C128 DEVELOPER'S PACKAGE

FOR COMMODORE 6502 DEVELOPMENT

OCTOBER 1987

Commodore®

AMIGA®

C128 DEVELOPER'S PACKAGE

FOR COMMODORE 6502 DEVELOPMENT

OCTOBER 1987

WRITTEN BY: Hedley Davis
Fred Bowen
Dan Baker
Connie Kreuzer

DISTRIBUTION

DISCLAIMER

This Developer's Package, the information contained and the programs provided herein are provided "AS IS" without warranty of any kind, either express or implied, including, but not limited to the implied warranties of merchantability or fitness for a particular purpose.

The entire risk as to the accuracy, reliability, correctness and currentness of the information and programs is assumed by the user thereof and not by Commodore Electronics Limited.

In no event shall Commodore Electronics Limited be liable for any indirect, consequential or incidental damages (including, but not limited to loss of profits, loss of business, loss of data or damage to property) arising out of the use of the information or programs provided herein, even if Commodore Electronics Limited has been advised of the possibility of such damages.

COPYRIGHT

TRADEMARK ACKNOWLEDGEMENTS

P/N 315820-01

TABLE OF CONTENTS

INTRODUCTION

SECTION I – C128 MODE

SECTION II – C64 and C128 MODES

INTRODUCTION

WHAT'S IN THE PACKAGE

The C128 Developer's Package is suitable for both large and small development projects. The package works best with systems having more than one disk drive and an 80-column text display, but minimal systems are supported as well.

The C128 Developer's Package includes this manual and two disks containing information and programs for use in developing 6502-based assembly language software. The manual and disks are separately described on the following pages.

ABOUT THIS MANUAL

This manual is divided into two sections. Section I, C128 Mode, contains information that is applicable to only the C128 in C128 mode. Section II, C64 and C128 Modes, contains information that is applicable to both the C64 and the C128.

SECTION I - C128 MODE

This section includes Chapters 1-4.

Chapter 1 - ED128 Full Screen Editor - describes ED128, a full screen editor for the C128, similar in function to the DEC® editor, EDT. This editor can be used for all text preparation tasks and functions in both ASCII and PETASCII.

Chapter 2 - HCD65 MACRO Assembler - describes HCD65, a powerful 6502 Macro assembler similar to the assembler used to assemble the C128 operating system. This assembler supports conditionals, local labels, many directives, cross references, etc.

Chapter 3 - C128 Object File Loader - describes a simple fast object file loader for the C128. This loader reads MOS-HEX object files created by the assembler into RAM memory, where they may be saved via the monitor to disk as executable binary files.

Chapter 4 - **C128 ROM Differences** - this chapter describes the differences between the C128 and 1571 ROM revisions. You can use this information to write better programs for all C128s.

SECTION II - C64 and C128 MODES

This section includes Chapters 5-13.

Chapter 5 - **C64 Tools** - describes three tools: a sprite editor, a sound editor, and a character editor. These tools run only on the C64 or C128 in C64 mode. You can use the tools to develop applications for the C128 or C64, but the tools themselves do not run in C128 mode.

Chapter 6 - **64 Fast Load #1** - lists the source code for a routine to speed up disk loading (2X normal LOAD) on a 1541 or 1571 disk drive. On return the carry flag is set if there is an error. The loader resides at $C000-$C318 or can be reassembled.

Chapter 7 - **64 Fast Load #2** - lists the source code for a routine to speed up disk loading (3X normal LOAD) on a 1541 or 1571 disk drive. This routine includes a simple user interface. These routines can load to any address between $0800-$FFFF.

Chapter 8 - **64 Fast Load #3** - lists the source code for a routine to speed up disk loading (2-3X normal LOAD) on a 1541, 1571 or 1581 disk drive.This routine includes a simple user interface.

Chapter 9 - RAM Expansion Code - lists the source code of three helpful routines for the C64 or C128 with the 1700, 1750, and 1764 RAM expansion cards. This code determines the size of the RAM card and performs general purpose STASH and FETCH routines.

Chapter 10 - 1351 Mouse Drivers - lists the source code for the two 1351 mouse driver routines for the C64 and C128.

Chapter 11 - 1571 Burst Load - lists the source code for a set of subroutines that support the 1571 Burst commands.

Chapter 12 - 1581 Burst Load - lists the source code for a set of subroutines that support the 1581 Burst commands.

Chapter 13 - BASIC 7.0 Math - describes the routines used in the C128 BASIC 7.0 floating point math package including the table of jump vectors.

ABOUT THE DISKS

This package includes two disks for program development.

Disk 1, side 1 contains:

ED128 Editor
HCD65 Macro Assembler
Loader and other 128 Tools

To access the programs on this disk, type **RUN"*"**. This will bring you to the main menu. For more information, see chapters 1-3 of this manual.

Disk 1, side 2 contains:

RAM Expansion Routines
1351 Mouse Routines
64 Tools

You can run the following programs on this disk:

C128CRUNCHER	CRUNCHER
C128UNCRUNCHER	UNCRUNCH
CHARED	ZAPLOAD 64
SPRED	AUTO-RUN 64
SIDMON1.41	

The remainder of the data on this disk supports these programs. For more information, see chapters 5, 9, and 10 of this manual.

Disk 2, side 1 contains:

1571/1581 Burst Routines

You can run the following programs on this disk:

1571 BURST.BAS
BURST EXAMPL.BAS

The remainder of the data on this disk supports these programs. For more information, see chapters 11 and 12 of this manual.

Disk 2, side 2 contains:

C64 Fast Loaders

To run the programs on this disk, run the .BIN files. For more information, see chapters 6-8 of this manual.

The following is a list of all the files on your disks.

DISK 1, SIDE 1

```
0  "DEVPAK128 072187" DP 2A
6     "STARTUP.072087"    PRG
1     "EDT.BAS"           PRG
33    "EDT.BIN V2.1"      PRG
27    "HCD65.BAS V3.3"    PRG
42    "HCD65.BIN V3.3"    PRG
6     "LOADER.BIN V0825"  PRG
18    "FILECOPY"          PRG
2     "FILECOPY.BIN"      PRG
19    "UNI-COPY"          PRG
18    "BACKUP 1571"       PRG
3     "BACKUP.BIN"        PRG
18    "BACKUP 1581"       PRG
10    "AUTOBOOT 128"      PRG
37    "SECTOR EDITOR"     PRG
21    "SHOW BAM"          PRG
5     "DOSWEDGE.BAS"      PRG
4     "DOSWEDGE128.BIN"   PRG
4     "DOSWEDGE64.BIN"    PRG
8     "RAMDOS.BAS"        PRG
28    "RAMDOS128.BIN3.5"  PRG
28    "RAMDOS64.BIN3.5"   PRG
6     "RAMTEST.BAS"       PRG
16    "RAMTEST.BIN"       PRG
1     "SNIFF.BIN"         PRG
30    "COMPRESS 128"      PRG
3     "COMPRESS.BIN"      PRG
1     "HELP.BAS"          PRG
17    "HELP.BIN"          PRG
251 BLOCKS FREE.
```

DISK 1, SIDE 2

```
0  "RAM-MOUSE-TOOLS V1.2A"
1     "-------------------"  PRG
1     "RAM EXPANSION"        PRG
1     "-------------------"  PRG
24    "RAMEXP.SRC"           SEQ
3     "RAMEXP.OBJ"           SEQ
1     "RAMEXP.BIN"           PRG
64    "RAMEXP.LST"           SEQ
1     "-------------------"  PRG
1     "1351 MOUSE #1"        PRG
1     "-------------------"  PRG
2     "MOUSE128.BAS"         PRG
1     "MOUSE128.BIN"         PRG
12    "MOUSE128.SRC"         SEQ
1     "MOUSE64.BAS"          PRG
2     "MOUSE64.BIN"          PRG
11    "MOUSE64.SRC"          SEQ
3     "MOUSE.POINTER"        PRG
1     "-------------------"  PRG
1     "1351 MOUSE #2"        PRG
1     "-------------------"  PRG
2     "M1351.64.BAS"         PRG
2     "M1351.64.BIN"         PRG
11    "M1351.64.SRC"         SEQ
4     "M1351.128.BAS"        PRG
2     "M1351.128.BIN"        PRG
12    "M1351.128.SRC"        SEQ
1     "-------------------"  PRG
1     "TOOLS FOR THE 64"     PRG
1     "-------------------"  PRG
28    "C128CRUNCHER"         PRG
9     "C128UNCRUNCHER"       PRG
38    "CHARED"               PRG
1     "CHAR-ML"              PRG
1     "SPRED%                PRG
64    "SSPED.8000"           PRG
18    "SIDMON1.41"           PRG
11    "CRUNCHER"             PRG
8     "UNCRUNCH"             PRG
10    "ZAPLOAD 64"           PRG
7     "AUTO-RUN 64"          PRG
300 BLOCKS FREE.
```

DISK 2, SIDE 1

```
0 "1571/1541 BURST V 80 2A"
74    "X571 BURST.SRC"    SEQ
206   "1571 BURST.LST"    SEQ
28    "1571 BURST.BAS"    PRG
3     "1571 BURST.BIN"    PRG
1     "----------------"  PRG
26    "BURST EXAMPL.BAS"  PRG
101   "BURST SUBS.SRC"    SEQ
222   "BURST SUBS.LST"    SEQ
4     "BURST SUBS.BIN"    PRG
0 BLOCKS FREE.
```

DISK 2, SIDE 2

```
0 "64 FAST LOADERS V FL 2A"
1     "EDT.BAS"           PRG
33    "EDT.BIN"           PRG
1     "----------------"  PRG
1     "C64 FAST LOAD #1"  PRG
1     "----------------"  PRG
19    "FLOAD1541.SRC"     SEQ
31    "FLOAD.C64.SRC"     SEQ
2     "FLOAD1541.OBJ"     PRG
3     "FLOAD.C64.OBJ"     PRG
4     "FL.C000-C318"      PRG
121   "LARGE C64 FILE"    PRG
17    "INSTRUCTIONS"      SEQ
1     "EXAMPLE"           PRG
1     "----------------"  PRG
1     "C64 FAST LOAD #2"  PRG
1     "----------------"  PRG
184   "BZAP.SRC"          SEQ
9     "BZAP.BIN"          PRG
1     "----------------"  PRG
1     "C64 FAST LOAD #3"  PRG
1     "----------------"  PRG
190   "FAST3.SRC"         SEQ
10    "FAST3.BIN"         PRG
30 BLOCKS FREE.
```

OTHER REFERENCE SOURCES

In addition to this Developer's Package, you may want to consult some of the following reference sources:

- Commodore 128 System Guide
- Commodore 1571 Disk Drive User's Guide
- Commodore 1541 Disk Drive User's Guide
- Commodore 1581 Disk Drive User's Guide
- Commodore 64 Programmer's Reference Guide
  (available from Howard W. Sams & Co., Inc)
- Commodore 128 Programmer's Reference Guide
  (available from Bantam Books)
- Commodore Magazine
  (published monthly by Commodore Magazine Inc., 1200 Wilson Drive, West Chester, PA 19380, U.S.A. U.S. subscriber rate is $35.40 per year; overseas subscriber rate is $65.00 per year. Questions concerning subscription should be directed to Commodore Magazine Subscription Department, Box 651, Holmes, Pennsylvania 19043.)
- CP/M Plus™ User's Guide
  (available from Digital Research Inc.)
- CP/M Plus Programmer's User's Guide
  (available from Digital Research Inc.)
- CP/M Plus™ System Guide
  (available from Digital Research Inc.)
- Quantumlink
  (the Commodore personal computer network)

SECTION I
C128 MODE

CHAPTER 1

ED128 FULL SCREEN EDITOR

ABOUT THIS CHAPTER

This chapter explains how ED128 works. If you are not familiar with EDT or a similar editor, it is suggested that you read all of this chapter. Do the operations as you read them; experiment with a copy of your file.

If you are familiar with EDT, or a similar editor, then read the first section of this chapter, Getting Started. Then you can experiment, referring when neccessary to the ED128 help screens. (Press <**HELP**> to access the help screens.) Use this chapter as a reference. You may find the Beginning Commands section particularly helpful.

This chapter is divided into the following sections:

INTRODUCTION TO ED128

ED128 is a full screen editor for the C128 which is similar in function to the DEC standard screen editor, EDT. This powerful, easy-to-use editor can be used for all text preparation tasks. ED128 allows you to:

- get help
- insert or delete characters, words, lines, blocks of text
- move forwards or backward to the next char, word, line, or page, the beginning or end of file, the next occurrence of a search string
- enter commands to perform functions such as loading a file, inserting a file into text, saving a file, displaying a disk directory, printing a file, sending commands to a disk drive

GETTING STARTED

ED128 is started by inserting Disk 1, side 1 into your system disk drive (unit 8) and typing **RUN"*"**. Then choose EDITOR from the menu. This program allocates memory for the editor, loads in the binary editor image, and invokes the machine code portion of the editor. Note that if the editor is booted from the 80-column screen, the system goes into fast mode automatically.

The top 23 lines on the screen form the editor's window into the text file. The bottom two lines on the screen are used by the computer for reporting status information.

When ED128 is first started, the screen appears blank except for the top line, and the bottom two lines.

The top line displays "[EOF]". This is a special indicator to show you where the end of the file is. The top 23 lines are the text editing area. Since no file has been loaded and no text has been entered, the only thing displayed is the [EOF] indicator and the cursor.

The bottom two lines are used to display status information, to report editor errors and disk errors, and to enter commands, repeat counts, or search strings. When the program is first started, the bottom line also displays the version number of this copy of the editor.

To enter text, simply type the text in the normal fashion. Do not use the keypad keys because they perform special functions.

TEXT DISPLAY

ED128 has three methods for displaying a character. It chooses one of these three methods based on the value of the character.

Normal:

For normal characters, ED128 simply displays the character in the appropriate position.

Inverse:

For text characters not in the normal character set, ED128 remaps the character onto a normal character, and displays the character in inverse mode.

Numerically:

For control characters which have no explicit meaning to ED128, there is a special four-character sequence used to display the character. For example, the form feed character has a decimal value of 12 and a hexadecimal value of $0C. If ED128 encounters a form feed in displayed text, it will show its hexadecimal value enclosed in angle brackets; i.e., ED128 will show "<0C>" instead of a form feed. This applies to all unprintable characters.

If text extends beyond the right hand column, ED128 shows a special checked character in the rightmost column to indicate that additional text extends beyond the right side of the display window.

KEYPAD COMMANDS

One of the main features of ED128 is its special use of the keypad keys on the right hand side of the keyboard. Essentially, each keypad key has been defined as a special function key giving the editor many functions available directly from the keypad. Certain keypad functions depend on others; therefore it is essential that you read this section in order to fully understand and use the remaining material in this document. Refer to the following keymap diagram as you read about the commands.

<HELP>

The HELP function is invoked by pressing the HELP key on the top line of the keyboard, or by pressing <F3> on the keypad. It causes help screens to be displayed. The first screen is a diagram of the keypad showing which keys perform which functions. The second screen lists the available screens and their syntax.

Function Keys

F1 **GOLD**	F3 HELP HELP	F5 FNDNXT FIND	F7 DELLIN UDELLIN

Numerical Keypad

7 PAGE COMMAND	8 SECT FILL	9 APPEND REPLACE	+ DELWRD UDELWRD
4 FORWARD BOTTOM	5 BACKWRD TOP	6 CUT PASTE	- DELCHR UDELCHR
1 WORD CASE	2 EOL DEL EOL	3 CHAR SPCINS	ENTER SUBS
0 BOL OPEN LINE		. MARK UNMARK	

This illustration shows the functions of the keypad and function keys while in the editor.

<GOLD>

The GOLD key <F1> is what is known as a "dead" key. By itself it performs no function. Pressing the GOLD key causes the system to interpret the meaning of the next key(s) pressed differently. This is different from the SHIFT, CONTROL, and C= keys which have to be pressed **while** the key to be modified is pressed. The system remembers that you pressed the GOLD key until an appropriate function is entered. You can think of the GOLD key as an automatic shift key.

The GOLD key has two main functions. The primary function is to select an alternate function for one of the keypad keys. For example, if you press <F7>, the delete line function is implemented. If you press the <GOLD><F7>, the undelete line function is implemented.

The second function of the GOLD key is to enter repeat counts. ED128 allows you to request that a certain function automatically be repeated. To enter a repeat count, press the GOLD key, followed by a decimal number using the digits on the normal keyboard, followed by the keypad command you wish implemented. Note that the repeat count is displayed on the bottom line of the display, and when the command is entered, the repeat count counts down as each iteration of the command is implemented.

Example: to delete 10 lines, press **<GOLD>**, then type **10** using the normal typewriter keys. Press the delete line key (F7).

GOLD functions can be repeated as follows:

Example: To undelete a line 10 times, press **<GOLD>** key. then type **10** using the normal typewriter keys. Press **<GOLD><F7>**.

Certain commands ignore repeat counts. For example, if you try to convert the case of text 20 times, the editor will only do it once.

All of the normal keyboard keys (except the digits 0-9) are considered to be special command keys, and therefore can be repeated. For example typing **<GOLD>20<RETURN>** causes 20 carriage returns to be entered into the file.

COMMAND <GOLD><F7>

To display the command prompt press **<GOLD><F7>**. You can type commands at the command prompt for disk access and other special functions. See the BEGINNING COMMANDS section for additional information.

<ENTER>

The enter key is used to terminate commands and search strings.

SPCINS <keypad 3>

The SPCINS (SPeCial INSert) command allows any character code to be inserted into the text buffer. (Form Feeds, for example). To use SPCINS, press the GOLD key and then type in the decimal value of a character (0-255) as if you were entering a repeat count. Then press <GOLD><keypad 3>. The SPCINS command then examines the repeat count and inserts a character of that value into the text at the cursor. The SPCINS command cannot be repeated.

MOVING AROUND IN THE TEXT

SCROLLING

The top 23 lines of the display are used as a window into your text. This window moves both vertically and horizontally through the text as you move the cursor around. ED128 follows specific rules for this operation.

VERTICAL SCROLLING - Vertical scrolling is done whenever the cursor approaches the top or bottom of the screen. As the cursor moves towards the bottom, ED128 will bring more text into view to prevent you from ever editing on the bottom line. This allows you to always see a few lines of text both in front of and behind the cursor. If there is no more text in the buffer, then ED128 will allow the cursor to move to the bottom of the screen. Similar rules apply to the top of the screen.

HORIZONTAL SCROLLING - Horizontal scrolling is performed whenever the cursor attempts to move off of the screen horizontally. Unlike vertical scrolling which moves in single line increments, horizontal scrolling is done on tab (8 column) boundaries. Pressing control cursor left or control cursor right instructs the editor to perform a horizontal scroll. This will be executed so long as the requested scroll does not move the cursor off the screen.

CURSOR KEYS

The cursor keys are used to move around in the text buffer. They will take you to the next position where there is text. If you are at the end of a line and press the cursor right key, the cursor will move down and left to the start of the next line because the cursor can only go where text exists. If you wish to move beyond where text exists, type spaces or tabs to move the cursor.

In addition to the cursor keys, there are nine keypad commands you can use to move text in the buffer.

FORWARD / BACKWARD <keypad 4> / <keypad 5>

Several commands move forwards or backwards though the text. The FORWARD and BACKWARD commands are used to set the direction of this movement. This direction is called the current direction throughout this text.

PAGE <keypad 7>

The PAGE command moves the cursor in the current direction to the next occurrence of a form feed in the text buffer.

BOL <keypad 0>

BOL (Beginning Of Line) moves the cursor to the start of the current line. If the cursor is already at the start of the line, then BOL moves the cursor to the start of the next line in the current direction.

EOL <keypad 2>

EOL (End Of Line) moves the cursor to the end of the current line. If the cursor is already at the start of the line, then BOL moves the cursor to the start of the next line in the current direction.

SECT <keypad 8>

SECT (SECTion) moves the cursor 16 lines in the current direction leaving it positioned at the start of a line.

WORD <keypad 1>

WORD moves the cursor to the start of the next word in the current direction. Note that the tab character and the carriage return character are considered to be words.

CHAR <keypad 3>

CHAR (CHARACTER) moves the cursor to the next character in the current direction.

TOP <GOLD><keypad 5>

TOP moves the cursor to the top (beginning) of the buffer.

BOT <GOLD><keypad 4>

BOT (BOTtom) moves the cursor to the bottom (end) of the buffer.

SEARCHING FOR TEXT

Once you are able to move around in the text, you can use a number of commands to edit text. These include:

FIND <GOLD><F5>

The FIND command allows you to search the buffer for specific words. Pressing <GOLD><FIND> displays a prompt on the bottom line of the screen. Type in the string you wish to search for. Press <ENTER> and the editor looks in the current direction for that string. If the string is not found, an error message is displayed. Note that such searches locate both upper and lowercase characters.

FNDNXT <F5>

FNDNXT (FiND NeXT) moves the cursor in the current direction to the next occurrence of the search string. (See FIND).

DELETING AND MOVING TEXT

The following three commands work on a single character at a time.

DELETE (main keyboard)
deletes the character immediately prior to the cursor and backs the cursor up to where that character was.

DELCHR <keypad ->
(DELete CHaR) deletes the character under the cursor and shifts the rest of the line left to take its place.

UDELCHR <GOLD><keypad ->
(UnDELete CHaR) restores the character previously deleted by DELETE or DELCHR. This last deleted character can be restored (duplicated) many times; the editor never forgets what the last deleted character was.

The following commands work with words up to 255 characters in length. Applying these commands to longer words causes error messages.

DELWRD <keypad +>
DELete WoRD) deletes from the cursor position to the start of the next word, including any spaces separating the words. The rest of the line is shifted to take the deleted word's place.

UDELWRD <GOLD><keypad +>
(UnDELete WoRD) undoes a DELWRD. This command can be repeated as the editor never forgets what the last deleted word was.

The following commands work on a single line at a time.

DELLIN <F7>
(DELete LINe) deletes all the text from the cursor to the end of the current line including the carriage return. In addition to removing text, this has the effect of joining the following line to the current line.

UDELLIN <GOLD><F7>

(UnDELete Line) undeletes all the text removed by DELEOL or DELLIN. This command can be repeated.

DELEOL <GOLD><keypad 2>

(DELete to End Of Line) deletes all text from the cursor to the end of the line. It does not remove the carriage return.

BEGINNING COMMANDS

Type the following editing commands at the command line prompt. To display this prompt while editing, press **<GOLD><keypad 7>** as shown on the keypad help diagram or press **<CONTROL>Z**.

You may enter these commands by typing only the first letter of the command unless otherwise specified. Commands must be terminated by pressing <ENTER>. If you press <RETURN> a carriage return is entered in as part of the command.

The following is a list of ED128 commands and their definitions.

LOAD

LOAD is used to load a text file into memory. The previous contents of the text buffer are lost. If the command is followed by a unit number, then that unit is used. If the file is loaded successfully, then the filename specified and the drive used become the default drive and filename.

INCLUDE

INCLUDE is just like LOAD except that it does not overwrite the default filename, and it inserts the contents of the specified file into the main text buffer immediately following the cursor.

SAVE

SAVE is used to save a file from memory to disk. If no unit number is specified, then the default unit number is used. If no filename is specified, the editor tries to save the file under the default filename. If that file already exists on the disk, the SAVE operation fails, and an appropriate error message is displayed. It is legal to specify a filename consisting of a single character '@'. In this case the file is saved, overwriting any other file present with the same name.

QUIT

QUIT is used to unconditionally exit the editor. No attempt at saving the current buffer contents is made. You must type the entire word "QUIT" when using this command. Use QUIT with care; anything in the editor buffer is lost forever when you QUIT.

EXIT

EXIT does two things. First, it saves the file on disk, with replace enabled. If that is successful, then EXIT exits the editor. If the implied save is unsuccessful, then an error message is displayed, and the editing session continues. EXIT has the same default options as LOAD, INCLUDE, and SAVE.

TYPE

TYPE is used for printing files to serial bus printers. It accepts two arguments. The first is the unit number which must always be present. The second is the secondary address. If the secondary address is not specified, then none is used. This is to allow usage of certain printers that do not allow the use of secondary addresses. Pressing STOP while printing aborts the command.

The TYPE command simply dumps the contents of the buffer to the specified output device. No translation takes place, tabs are not expanded, and line feeds are not inserted. If your printer requires special characters to be sent to it in order to specify letter quality, or 132-column mode, these characters can be easily set up at the head of the file using the special insert function, SPCINS. If your printer requires line feeds, insert them at the start of every line. The SUBSTITUTE function with repeat counts can make this a simple task.

DISK DIRECTORY COMMAND

'$' is a single character command used to display disk directories. It defaults to the load disk unless a unit number is specified. Selective directories are also possible.

DISK STATUS & COMMANDS

'@' is a single character used to display disk status and to send disk commands from the editor. Disk status is normally displayed after every disk operation, so this command is rarely used. The disk status command defaults to the load unit unless another unit number is specified. To send a disk a command from the editor, type the @ command followed by the disk command. Refer to your disk drive user's manual for further information.

GOTO LINE COMMAND

If you enter a decimal number as a command, the cursor will be located that many lines from the start of the main buffer. If a lesser number of lines are present, the cursor is put at the end of the buffer.

STATUS LINE COMMANDS

'O', 'ON' or 'OFF' are commands for controlling whether status information is displayed on the bottom two lines of the screen. Using the 'O' command causes the status line to toggle. Note that even if the status line is disabled, error messages are still displayed there.

KEYMAP COMMAND

The keymap command switches the functions of the CONTROL and STOP keys. This change is not displayed and the current state can only be determined by using those keys. On VT100 style keyboards, the TAB key is positioned where the CONTROL key is on the C128, and the CONTROL key is where the STOP key is. This command simply changes the function of these two keys for ergonomic purposes.

NOTE: The gray TAB key on the top row will always produce a tab. The STOP key will function both as <STOP> and <CONTROL>.

PETSCII & ASCII COMMANDS

These commands toggle the editor between the PETSCII & ASCII modes of operation. There are two differences between these modes:

1) The characters are displayed differently per the two different standards. Characters which are not part of the standards are remapped to other characters and displayed as inverse text. If the editor is running on the 80-column screen, then the full ASCII character set is displayable. On the 40-column screen, ASCII characters which have no PETSCII equivalent are displayed as the corresponding special PETSCII character. See tables 1 and 2 for details.

2) Different character codes are entered from the keyboard. This has the effect of canceling the display changes. If you type a lower case 'a' in PETSCII or ASCII mode, then a lower case 'a' is displayed even though a different character code is entered into the buffer.

The following tables outline the display techniques and the keyboard codes generated for both PETSCII and ASCII mode.

Table 1 -- PETSCII MODE

CHARACTER	KEYBOARD MAP		TYPE	PRINT FORMAT
$00-1F 0-31	control	normal PETSCII	(control)	control exp
$20-3F 32-63	numerics & punc		(text)	numerics
$40-5F 64-95	lower_case		(text)	lower case
$60-7F 96-127	never received from keyboard		(inv text)	inv upper case
$80-9F 128-159	more control	normal	(control)	control exp
$A0-BF 160-191	pet graphics	normal	(text)	pet graphics
$C0-DF 192-223	upper case text	normal	(text)	upper case
$E0-FF 224-225	KEYPAD KEYS	(SPECIAL)	(control)	inv pet graphics

Table 2 -- ASCII MODE

CHARACTER	KEYBOARD MAP	TYPE	PRINT FORMAT
$00-1F 0-31	control (except CR and TAB)	(control)	control exp
$20-3F 32-63	numerics & punc	(text)	numerics
$40-5F 64-95	upper_case	(text)	upper case
$60-7F 96-127	lower case	(text)	lower case
$80-9F 128-159	more control normal	(control)	control exp
$A0-BF 160-191	never received from keyboard	(inv text)	inv numerics
$C0-DF 192-223	never received from keyboard	(inv text)	inv upper case
$E0-FF 224-225	KEYPAD KEYS (SPECIAL)	(control)	inv lower case

MARKED TEXT COMMANDS

ED128 has many commands that deal with an area of selected text. There are two ways to select text:

1) with the MARK command

2) with the cursor at the start of a search string

MARK <keypad .>

Specification of a region of text is done by moving the cursor to one end of the text, pressing MARK (the decimal point key on the keypad), and then moving the cursor to the other end of the text. All the text in the marked area will be displayed as inverse characters. At this point, if an operation such as CUT is invoked, the operation takes place (the highlighted text is cut) and the inverse mode is canceled.

UNMARK <GOLD><keypad .>

The UNMARK command cancels inverse mode and restores normal editor operation.

By themselves, MARK and UNMARK do not change any text. They simply indicate a block of text to be operated on by the following commands.

CUT <keypad 6>

The CUT command removes the marked text from the main buffer and places it in a special buffer called the paste buffer. This buffer is the same size as the main buffer, therefore a large region of text may be moved to the paste buffer.

PASTE <GOLD><keypad 6>

The PASTE command copies the contents of the paste buffer into the main buffer at the cursor location. Any text which is cut can be pasted back into the main text buffer. The editor remembers what is in the PASTE buffer; it may be copied as often as you wish.

APPEND <keypad 9>

The APPEND command removes the marked text from the main buffer and appends it to the end of the text in the paste buffer.

REPLACE <GOLD><keypad 9>

The REPLACE command deletes the marked text area, and replaces it with the contents of the paste buffer.

SUBS <GOLD><enter>

The SUBS (SUBStitute) command is used with the FIND command. If the cursor is at an occurrence of the current search string, SUBS will delete that instance of the search string and place the contents of the paste buffer in its place. SUBS then automatically searches through the text for the next occurrence of the search string. This is useful for changing all occurrences of a word to another word.

CASE <GOLD><keypad 1>

CASE (change CASE) toggles the case of all the letters in the selected region. Lower case letters become upper case letters; upper case letters become lower case letters. All other characters are unaffected.

FILTER COMMANDS

Filter commands are used to perform an operation on every character within a MARKED text area. (See marking text). Each command reacts differently in PETSCII mode and ASCII mode. The four filter commands are:

FMASK

When the editor is in ASCII mode, characters in the buffer with bit 7 set are displayed as inverse characters. PETSCII mode also displays normal nonstandard characters in inverse mode. The FMASK command examines each character in the marked region and converts all those which are inverted to their normal noninverted counterpart.

FPTOA

The FPTOA command converts PETSCII text to ASCII text. This command must be entered while in PETASCII mode.

FATOP

The FATOP command converts ASCII text to PETSCII text. This command must be entered while in ASCII mode.

MONITOR COMMAND

The monitor command is provided for extremely advanced users who have a need for it. THE IMPROPER USE OF THIS COMMAND MAY RESULT IN DISASTROUS EFFECTS INCLUDING SYSTEM CRASH, LOSS OF THE CURRENT BUFFER CONTENTS, OR EVEN LOSS OF DISK DATA. It is necessary to type the entire word "MONITOR" to invoke the machine language monitor. Control can be returned to the editor by using the X command from within the monitor. In this case, a short delay will occur while the editor recounts all of the carriage returns in the buffer.

ED128 LIMITATIONS

Several limitations exist in ED128. For the most part, these limitations are inconsequential. Extensive error checking is performed, and should these limits be exceeded, appropriate error messages are displayed. The limitations are

Amount of text in main buffer:
47 Kbytes

Amount of text in paste buffer:
47 Kbytes

Amount of text in search string:
255 bytes

Amount of text in deleted word buffer:
255 bytes

Amount of text in deleted line buffer:
255 bytes

Maximum repeat count:
9999 iterations

Maximum characters on a line:
47 Kbytes.

Maximum editable line length:
255 chars

CHAPTER 2

HCD65 MACRO ASSEMBLER

ABOUT THIS CHAPTER

This chapter explains how HCD65 works. To fully understand it is suggested that you read all of this chapter. Do the operations as you read them; experiment with your file.

This chapter is divided into the following sections:

INTRODUCTION TO HCD65

HCD65 is powerful macro assembler with a rich set of directives to support the most demanding assembly needs. It supports the full MOS65xx instruction set and provides facilities for user customization. The macro capability helps to eliminate some of the tedium associated with assembly language programming. This assembler is similar in nature to the assembler used by Commodore systems software engineers to develop all of the code for the C128 product line. In fact, one benchmark used for testing this assembler was to assemble the C128 source files on the C128.

INPUT FILES

HCD65 accepts sequential-type text files as source input. These files must come from disk.

OUTPUT FILES

HCD65 has three types of output files:

LIST file
OBJECT file
ERROR file

Each of these files uses a specific channel specified by a basic startup program. This allows you to direct the output for these files to whichever output device is appropriate. The output for these files may also be inhibited, or all files may be merged so that their output is combined.

TEMPORARY FILES

HCD65 uses a sequential type disk file for holding cross reference information. You may cause this file not to be used (therefore no cross reference is generated), or you may specify which disk drive is to be used for this file.

FEATURES

HCD65 contains a rich set of conditional assembly directives. Conditionals may be nested up to 10 deep.

HCD65 has a powerful macro facility which is not just a conventional text expansion macro facility. The macro facility allows macros to redefine opcodes, and it allows macros which define, or redefine other macros. Combined with the conditional assembly features, macros provide a powerful tool for code generation. Macros can be nested to any depth, but are practically limited by the amount of memory in the machine.

HCD65 consists of two distinct programs. The first is a BASIC Shell that you can customize. This program is responsible for setting up the machine language assembler. The BASIC Shell is responsible for providing the user interface and setting up the many assembler parameterers required by the machine code portion of the assembler.

The BASIC Shell control program sets up communications registers to the machine code, and then passes control to the machine language program. The machine language performs the actual assembly, and returns control to BASIC Shell which then performs some final housekeeping.

By using the BASIC Shell, you have control over the following:

- The source file to be assembled.
- The disk drive(s) to be searched for source files.
- The output device to send the listing to.
- The output device to send error reports to.
- The output device to send the object file to.
- The disk drive to use for the cross reference temporary file.
- How many characters wide the listing should be.
- What to print for the date on each listing page.

GETTING STARTED

HCD65 is started by inserting Disk 1, side 1 into your system disk drive (unit 8) and typing **RUN"*"**. Then choose ASSEMBLER from the menu.

The assembler prompts you for the name of the main source file to be assembled. After you answer this prompt, the assembler provides several groups of options for your selection. If none of these preprogrammed configurations is suitable for your needs, you may simply press <RETURN> instead of selecting an option. At this point, the assembler leads you through a series of questions about every aspect of its operation. Respond to the various prompts to configure the assembler to your needs.

CASE SENSITIVITY

The assembler is case insensitive for all non-quoted strings. This includes symbol names, label names, macro names, opcodes, and directives. The list file shows the source statements in the case they are presented to the assembler in. The symbol table and cross references list all symbols in the same case. Subtitles, program names, and other uniquely textual items are not case sensitive. However, the DOS is case sensitive, therefore, INCLUDE filenames must be specified in the correct case.

CONSTANTS

Internally, HCD65 uses 16-bit values to represent constants. Constants may be expressed in one of four radices or may be created using literal strings.

Hexadecimal Constants

Hexadecimal constants are represented by one to four hexadecimal characters following a dollar sign (i.e., $1A7F).

Decimal Constants

Decimal constants are represented by one to five decimal digits. No leading radix character is needed.

Octal Constants

Octal constants are represented by an octal number preceded by the "@" symbol. (i.e., @17777).

Binary Constants

Binary constants are represented by a binary number preceded by the "%" symbol. (i.e., %01010100111).

Literal Constants

Literal constants are represented by one or two ASCII or PETSCII characters enclosed in matching single quote characters. In certain situations, only the first single quote is neccesary. Certain differences exist in the way the .BYTE directive handles quoted strings. See that directive's explanation. In the case where two literal characters are enclosed in quotes, the assembler places the first character in the low order field in the resulting 16-bit value.

INPUT FILE FORMAT

There are four fields recognized by the assembler. Each is optional and they typically are delimited by spaces or tabs.

Label_Field Operator_Field Argument_Field Comment_Field

Comment lines are marked with a semicolon as the first non-white character (white characters are spaces and tabs). Any characters appearing after a semicolon (except a semicolon that appears in quotes) are considered to be comments and are ignored.

Any text starting in the first column which is non blank and is not a comment is considered to be a label or symbol except in the case where there is a macro directive on that line. In that case the text is considered to be the macro name.

The text in the second field defines how the assembler will interpret that line. Mnemonics, assembler directives, and macro calls are all placed here.

The third field generally contains arguments to the second field. In cases where the operation does not expect arguments, the third field is considered to be a comment field.

Anything after the third field is considered to be a comment.

SYMBOLS AND LABELS

Three type of symbols are supported by the assembler.

Global symbols

Global labels

Local labels

Global symbols and labels represent 16-bit values. A symbol name is a string alphanumeric characters. None of the characters can be the characters which the assembler recognizes as an expression delimiter (i.e., quote marks, spaces, expression operators, radix operators, etc). In addition, the first character in a symbol name cannot be a digit from 0-9, as those characters are indicative of local labels.

Global symbol or global label names may be of any length, although only the first 32 characters are significant and all other characters are truncated for cross reference and symbol table purposes.

Examples of valid symbols:

the_routine_is_here

a3485734583488

periods.are.legal

this_symbol_so_long_that_it_is_the_same_as_the_next

this_symbol_so_long_that_it_is_the_same_as_the_previous

Examples of illegal symbols:

here+nop ; this is an expression with two symbols

123ksdhjfks ; this starts with a digit

NOTE: The asterisk "*" is a special symbol. It represents the current program counter. It may be assigned a value and it may be evaluated.

Global labels differ from global symbols in that the symbols can be redefined many times during assembly. Labels can only be defined once.

A symbol definition must be made explicitly using the equals sign ('='). Any time a potential symbol appears in the label field, and is not explicitly made a symbol using the equal sign, it becomes a label. Further attempts to define it result in assembly error generation.

Local labels take the form of one to three decimal digits with the value of 1-255 immediately followed by a dollar sign.

Examples of local labels:

```
100$      ; this is legal
001$      ; this is the same as the next
1$        ; this is the same as the previous
999$      ; this is illegal ( 1-255 ).
```

The range over which a local label is defined is delimited by two things:

1) global labels
2) the .local directive.

Example:

```
test      jsr 10$   ; ok
10$       bne 20$   ; ok
20$       bpl 30$   ; not ok, 30$ not defined here
test2     nop       ; the label here delimits the 30$
30$       bne 10$   ; ok, this 10$ is the one below.
10$       nop       ; ok, this is a different 10$ than
                    ; the one on the second line.
          .local
          jmp 30$   ; not ok, the .local directive limits
                    ; the range of the 30$
```

ASSIGNING VALUES TO SYMBOLS

Symbols may given a value in two ways.

Appearance in the label field.

A symbol which appears in the label field becomes a label except in 2 cases:

1) The symbol is being assigned a value using the '=' sign. (see below)

2) The symbol appears on a line with a .MACRO directive. In this case the symbol becomes the name of macro.

Explicit assignment using the equals sign

A symbol may be assigned a value using the equal sign.
For example:

```
nine = $09 ; assign the hex value 9 to the symbol "nine"
```

EXPRESSIONS

Expression processing in HCD65xx accepts a large number of operators. Expressions are evaluated left to right except in the following cases:

Highest Priority operators:

Unary +	truth operator
Unary -	two's complement operator
!N	one's complement operator (logical not)
<	low byte operator (returns low byte of value)
>	high byte operator (returns high byte of value)

Second Priority operators:

*	16 bit multiply, returns low order 16-bit result
!.	logical AND
!+	logical OR
!X	logical Exclusive OR

Lowest Priority operators:

Binary +	16 bit addition, carry discarded
Binary -	16 bit subtraction, borrow ignored.

For Example:

```
$1234      = $1234
>$1234     = $0012
>$1234+1   = $0013
1+>$1234   = $0013
5-1-1      = $0003
-1         = $FFFF
>-1        = $00FF
!N$000F    = $FFF0
```

MACROS

The macro facility provides text expansion and substitution capability. Macros are defined by enclosing a block of of text in .MACRO and .ENDM directives. All of the text between the two directives composes the body of the macro. When the .MACRO directive is encountered by the assembler, there must be a macro name in the label field. That name is used to call the macro. Whenever the assembler finds text in the operator field which is not preceded by a period (indicating a directive), it checks to see if that text is a macro name before checking to see if that text is an opcode mnemonic.

When a macro call is encountered in the source file, the text for the body of the macro with that name is substituted into the input stream where the macro call was. Therefore, one macro call can result in many lines of code for the assembler to handle.

Example of macro definition:

```
;the following macro clears the x and a registers.
clr_reg   .macro
          lda #0
          ldx #0
          .endm
```

Example of macro call:

```
label     nop
          clr_reg
          nop
```

Resulting assembled code:

```
label     nop
          lda #0
          ldx #0
          nop
```

Macros are expanded in a preprocessor before the input lines are fed to the assembler. They work entirely on a text substitution basis.

Macros allow arguments to be used. Dummy arguments, separated by commas, are declared as arguments to the MACRO directive. Many arguments are allowed, the upper limit being determined by the maximum length of the input string. When a macro is called, the input text is expanded verbatim except where dummy arguments occur. In this case, the arguments following the call to the macro are substituted into the positions where the dummy arguments were in the macro definition.

Example macro for loading a and x with immediate data:

```
definition
        ldi     .macro arg
                lda #>arg
                ldx #<arg
                .endm

call
        label   ldi $1234

expansion
        label   lda #>$1234
                ldx #<$1234
```

Because the macro facility only understands text streams, but does not understand about specific fields, the user must be careful in the selection of dummy arguments. Using short arguments can lead to unexpected expansions. For example, this following source code performs unexpectedly:

```
definition:
        ;store value in arg into memory at loc.
        setloc  .macro a
                lda #a
                sta loc
                .endm

call:
        label   nop
                setloc $0F
                nop
```

expansion:

```
label   nop
        ld$0F #$0F
        st$0F loc
        nop
```

This clearly erroneous expansion occurred because the macro facility blindy substitutes macro args wherever dummy args occur in the macro definition. One solution is to use longer dummy argument names. Another solution preferred by many users is to precede simple dummy argument names with a rarely used character. The percent symbol is a good choice.

definition:

```
        ;store value in arg into memory at loc.
setloc  .macro %a
        lda #%a
        sta loc
        .endm
```

call:

```
label   nop
        setloc $0F
        nop
```

expansion:

```
label   nop
        lda #$0F
        sta loc
        nop
```

As previously stated, macros can have many arguments. Blank dummy arguments can also occur in macro definitions. Arguments occurring in macro calls which have no corresponding dummy argument in the macro definition are simply ignored. Blank arguments occurring in macro calls evaluate as nothing when the macro is expanded.

For example:

Definition:

```
ld_chrs   .macro %a,%b
          lda #'%a'
          ldx #'%b'
          .endm
```

Call:

```
label     nop
          ld_chrs 1,,3
          nop
```

Expansion:

```
label     nop
          lda #'1'
          ldx #''
          .endm
```

Occasionally it is desirable to pass arguments to macros which contain characters like spaces or commas. Because these are normally stripped by the parser to delimit lines, or delimit arguments, this would seem impossible.

However, such an argument may be passed by placing them it inside angle brackets. Then, when the macro is to be expanded, the arguments are scanned for matching sets of angle brackets ("<",">"). If these are found, all the text between the angle brackets is passed as a single macro argument after the brackets are stripped away.

For example:

Definition:

```
ld_chrs   .macro %a,%b
          lda #'%a'
          ldx #'%b'
          .endm
```

Call:

```
label     nop
          ld_chrs 1,<,>
          nop
```

Expansion:

```
label   nop
        lda #'1'
        ldx #','
        nop
```

Many conditionals are supplied expressly for using inside of macros to allow detection of null fields, undefined symbols, etc. Other directives (.REPT, .IRPC, IRP) are actually special cases of macros and work using many .MACRO like principles.

LISTING FORMAT

The listing output has many features. The listing is paginated with several lines of information at the top of each page. Here is an example of a portion of a single page. Note that this page shows a call to the macro LD_CHRS used as an example in the previous sections describing macros.

```
MY_PROGRAM          HCD65XX   V1.0      12-DEC-1985                PAGE 10
UTILITY SUBROUTINE                                    0:UTILITIES.SRC,S,R

ERROR     ADDR      CODE      SEQ       SOURCE STATEMENT
          8122      EA        7000      LABEL     NOP
V         8123      A9 02     7001                LDA #$1002
                              7002+               LD_CHRS 1,<,>
          8125      A9 31     7003A               LDA #'1'
          8127      A2 2C     7004A               LDX #','
          8129      EA        7005                NOP
```

The top line contains four things:

1) The program name as defined by the .NAME directive.
2) Information identifying the assembler and version number.
3) The date information as described by the BASIC startup file.
4) The page number.

The second line contains two things:

1) The text defined by the last .SUBTTL directive.
2) The current source file being read to generate this page.

The third line contains headers for the columns output by the assembler. Source code listing lines have several fields.

Error field:

The error field is the first one on the line. It is placed there so that lines with error may be easily found. In general, assembly errors are indicated by a single letter in the error field on the line the error occurred on. If a line exhibits multiple errors, then several letters will appear there. One such error is shown in the sample listing indicating that a "VALUE" error occurred. This is because that line is attempting to load the eight-bit accumulator with a sixteen-bit value.

Address field:

The address field generally indicates the address of any object code bytes which are listed on that line.

Object field:

The object field generally indicates any object code bytes which were generated by the current line. It may also contain a 16-bit value preceded by an equal sign. This indicates the value of some expression which was evaluated on the line (for use in conditional directives, or which a symbol is equated to).

Sequence field:

The sequence field indicates the line number of the current source line. If a macro is called on this line, the sequence field is followed by a "+" sign. If the line was generated by a macro, then the sequence field is followed by a single letter from A-Z. This letter indicates the depth of macro expansion generating the current line.

Source Statement Field

The source statement field contains the verbatim source code for the current line. No beatification is performed. Tabs are displayed normally because the sequence field starts on a tab stop.

LISTING CONTROL DIRECTIVES

Many directives are supplied for controlling the format of the list file output.

NOTE: The list file output control only applies to normal listing output. If you define the object file as being the same as the list file, then the object file lines will appear in the listing as they are output by the assembler, independent of the settings imposed by any listing directives.

Each of the listing control directives is discussed here. Note that all source lines containing the directives are inhibited from the listing. Listing control directives are meant to control the listing, not add to it.

.NAME text
.NAM text

The .NAME directive is used to inform the assembler of the name of the code being assembled. When the .NAME directive is encountered, all text following the .NAME directive is copied to a buffer and is printed on each page header. The source line containing the .NAME directive is not listed. If the name is too long, it is truncated.

.SUBTTL text

The .SUBTTL directive is used to inform the assembler of a subtitle for the current listing pages. When the .SUBTTL directive is encountered, all text following the .SUBTTL directive is copied to a buffer and is printed on each page header. The source line containing the .SUBTTL directive is not listed. If the name is too long, it is truncated.

.PAGE
.PAG

The .PAGE directive forces the listing file to the top of a next page if it is not currently there.

.SKIP <optional expression>
.SKI <optional expression>
.SPACE <optional expression>

The SKIP directives are used to insert blank lines into the listing. If the skip directive is followed by a number, that many blank lines are inserted.

.FORMLN expression

The .FORMLN directive sets the number of lines per page. It controls how many lines are generated between page headers. The actual number of lines per page generated is the value specified by .FORMLN plus six. If the expression evaluates to zero, then page headers are inhibited. Other unreasonable values generate an error.

.LIST
.NLIST

These directives toggle a switch controlling whether listing output is enabled. Lines which generate errors override this setting. This switch is also overridden for the symbol table and cross reference outputs.

.CLIST
.NCLIST

These directives control whether lines of conditional assembly code, which are not truly being assembled, are listed or not. Normally the assembler lists such lines.

.MLIST
.NMLIST
.BLIST

These directives control how macro expansion lines are listed. .MLIST (the default setting) lists all macro expansion lines. .NMLIST inhibits all macro expansion lines. .BLIST lists all macro expansions which cause object code to be generated.

.GEN .NOGEN

Sometimes, a line of source code generates more bytes of object code than can fit on a single listing line. In this case, such bytes are listed on as many additional lines as neccesary. .NOGEN causes these additional lines to be inhibited. .GEN simply reenables them.

INPUT CONTROL DIRECTIVES

Input control directives are used for controlling which files are grouped together for assembly.

.INCLUDE filename

The include file is used to combine files within the assembly. Essentially, the assembler substitutes the entire contents of the named source file for the .INCLUDE statement. Note that the assembler forces a convention that all source files end in ".SRC". If the filename does not end in ".SRC", the assembler will append it to the filename before attempting to open the file.

Be careful in using of this statement not to open too many input files (exceeding the drive's capacity). See Limitations of 1541/1571 DOS for more information.

Also, be careful not to create circular linkages with this directive. This will result in ridiculously long assembly times.

.END

The .END directive terminates the assembly process and forces the assembler to ignore all further source lines.

CODE GENERATION DIRECTIVES

Several directives exist for generating bytes of object code which normal assembly code will not create.

.WORD args
.WOR args

The .WORD directive accepts a series of comma terminated expressions as arguments. Each argument is evaluated in order, and two bytes of object code are generated for each argument. The bytes are created in the usual low byte/high byte format, but are listed as 16-bit values.

.DBYTE args

The .DBYTE directive is just like the .WORD directive except the the bytes are created in high byte/low byte format, and each byte is listed individually in the listing.

.BYTE args
.BYT args

The .BYTE directive also accepts a series of comma-terminated arguments. Each argument may contain either a normal byte valued expression resulting in a single byte of object code or a quoted string. In the byte directive, quoted strings must be enclosed in matching sets of single or double quotes. All characters between the matching quotes are treated as a literal and create one byte each.

MACRO DIRECTIVES

.MACRO dummy1,dummy2,....dummyN
.ENDM

The .MACRO and .ENDM directives are extensively discussed in the section called "MACROS".

REPEAT DIRECTIVES

BEFORE YOU READ THIS SECTION READ THE SECTION ABOUT MACROS.

The repeat directives are used for repeating sections of code which are mostly or entirely identical. Using these directives is similar to using the .MACRO directive. That is, each usage of a repeat directive consists of the line with the directive, a body of code to be repeated, and a .ENDR directive. Note that internally .ENDM and .ENDR are treated identically and therefore may be interchanged.

.ENDR

The .ENDR directive is used to mark the end of a section of code which is being using in a repeat directive.

.REPT expression

The .REPT directive is the simplest repeat directive. It is used to create several sections of code which are identical. It accepts a single expression as its argument. That expression controls the number of times the body of repeat code is repeated.

For example, the following code causes the line with the .BYTE statement to be repeated 5 times resulting in twenty bytes of object code being generated.

```
.REPT 5
.BYTE 1,2,3,4
.ENDR
```

.IRP dummy_arguement,<0-N optional arguments>

The .IRP directive is used to define a temporary macro with a single dummy argument, then call it a variable number of times with a predefined set of arguments. The first argument to the .IRP directive is the dummy argument used in the body of the macro definition. Each remaining argument causes the body of the .IRP macro to be expanded once with that argument substituted for the dummy argument.

For example, the following .IRP

```
.IRP %DUMMY,THIS,IS,A,TEST
.BYTE "%DUMMY",0
.ENDR
```

generates the following code:

```
.BYTE "THIS",0
.BYTE "IS",0
.BYTE "A",0
.BYTE "TEST",0
```

.IRPC dummy_arguement,substititution_string

.IRPC is a macro similar to .IRP in nature. Instead of accepting a series of arguments to be iteratively substituted, it accepts one argument (after the dummy argument). During each iteration of the loop, one character from the argument is substituted for the dummy argument.

For example, the following .IRPC

```
.IRPC %DUMMY,ABC
.BYTE "%DUMMY",$%DUMMY%DUMMY
.ENDR
```

Generates the following code:

```
.BYTE "A",$AA
.BYTE "B",$BB
.BYTE "C",$CC
```

Because all repeat directives are actually special case macros, they can be nested to any depth. The use of angle brackets to pass unusual arguments is also supported. See the section on MACROS for additional information.

CONDITIONAL DIRECTIVES

Conditional directives are a powerful means of controlling the assembler. They allow intelligent selection between sets of source code based on several types of conditions including the numeric value of expressions, whether symbols are defined, whether strings are blank, and the identicality of strings,

Here is an example of a piece conditional source code.

```
.IFE SYMBOL
lda #0     ; this line is assembled if SYMBOL = 0
.ELSE
lda #1     ; this line is assembled if SYMBOL <> 0
.ELSE
lda #0     ; this line is assembled if SYMBOL = 0
.ENDIF
```

The main features of a conditional are the conditional directive itself and the .ENDIF directive terminating the range of the conditional.

In between these two lines is the conditional body. Normally the conditional body is assembled if the question the conditional directive asks is true. The .ELSE directive may be used to toggle the relative "TRUTH" of the conditional assembly thereby allowing the conditional to select between sections of source code to be assembled. All parts of the conditional other than the .ENDIF and the CONDITIONAL line itself are optional.

The use of undefined symbols in the expression for the conditional results in an error, (except for .IFDEF) and the conditional makes a choice as to whether to evaluate true or false.

There are several numeric conditionals. Each of these accepts a single expression as its argument. Numeric evaluation is considered to be a 16-bit twos complement.

.IFE expression evaluates true if expression is 0.
.IFN expression evaluates true if expression <> 0.
.IFGE expression evaluates true if expression is >= 0.
.IFGT expression evaluates true if expression is > 0.
.IFLT expression evaluates true if expression is < 0.
.IFLE expression evaluates true if expression is =< 0.

There are several textual conditionals. These are essentially useless if used alone. However, when combined with macros they can make many tasks easier. They can, for example, be used to detect the presence or absence of arguments.

.IFB <string>
.IFNB <string>

The .IFB directive evaluates true if its argument is found to be blank. Because of its nature, it is strongly recommended that this argument be delimited by the enclosing angle brackets as discussed in the section describing MACROS. .IFNB is simply the inverse of .IFB

.IFIDN <string1>,<string2>
.IFNIDN <string1>,<string2>

.IFIDN evalauates true if the two argument strings are identical. This operation is case sensitive. .IFNIDN evaluates true if they are not identical.

.IFDEF <symbol_name>
.IFNDEF <symbol_name>

.IFDEF evaluates true if the argument is both a legal symbol name, and has been previously defined in the source file. .IFNDEF evaluates true if the argument is either not a legal symbol name, or has not previously been defined.

MISCELLANEOUS DIRECTIVES

.LOCAL

The .LOCAL directive is used to delimit the range of local labels. It enables this operation without forcing the unneccesary act of thinking up yet another label name.

.MESSG <text>

The .MESSG forces the following text to be echoed out the error channel during pass2. Its primary use is inside of conditionals to present error messages for code overflow, etc.

.RMB <number>

(Reserve Memory Byte) .RMB is functionally identical to "*=*+". It advances the program counter without generating object code. <number> specifies the number of bytes to reserve.

ERROR REPORTING

There are three types of errors reported by the assembler. Errors are issued to both the error channel and to the listing.

FATAL ERRORS

Fatal errors prevent the assembler from continuing the assembly. These include running out of macro expansion space, and read errors from disk in the middle of a file. Fatal errors, which result in assembly termination, are accompanied by explanatory error messages.

SYSTEM ERRORS

System errors include inability to find an include file, to properly access the cross reference file, and other such non fatal errors.

ASSEMBLY ERRORS

Assembly errors are those related to the source file content. They can usually be associated with a single erroneous line of source code. As such, assembly errors are reported in the listing on the same line with the offending source. If the error channel is different from the listing channel, then the offending line is also echoed to the error channel.

Assembly errors occur for a variety of reasons. Each one has a specific error code printed in the first few columns of the listing output. The error codes and their definitions are listed below.

A Address error. Indicates bad address valued expression was evaluated. May indicate branch out of range.

B Balance error. Quotes, or angle brackets are mispaired on this line.

E Expression error. Invalid syntax in an expression. This error is more serious than a syntax error. Occurs when invalid expressions are used in critical places (like * = undefined_symbol).

F Field error. Something is missing on the line.

J Indicates that the address space is filled and that the resulting object code has wrapped from $FFFF to $0000 and a byte was created at $0000.

M Multiply-defined symbol. A symbol is defined more than once (where this is illegal). All but the first definition are ignored.

N Nesting error. Unexpected .ELSE, .ENDIF, .ENDR, or .ENDM detected.

O Undefined opcode or macro call used on this line.

P Phase error. Indicates the value of label was different in pass 2 than in pass 1. This may indicate a source file (disk) problem or some sort of illegal forward reference. The assembler is confused.

Q Questionable syntax. Indicates a syntax error which the assembler has resolved by some (probably incorrect) assumption.

S Syntax error. Generated for all sorts of syntactical errors.

U Undefined symbol. The assembler attempted to evaluate an expression which has an undefined symbol in it.

V Value error. An operand value was out of range. Typically generated when a 16-bit value is placed in an 8-bit field. Also flags attempts to branch out of range.

W Wasted byte warning. Generated when the assembler is forced to use an absolute addressing mode where a zero page addressing mode would suffice. This warning is typically created by forward references.

Z Division by zero error. Generated when an expression requests the assembler to divide by zero.

@ Symbol table overflow. The symbol table is full and a symbol on this line cannot be written to the symbol table. All references to this symbol will result in undefined symbol errors.

? Internal error checking has conflicting results. This error occurs when the assembler detects an error which by design, should not occur. This is indicative of a bug; however chances are that some construct on the line is questionable. This error can usually be eliminated by rearranging the line.

* Too many error codes were generated for this line for the assembler to list them all.

THE BASIC SHELL PROGRAM

The BASIC shell program is totally customizable by the user to perform whatever assembly functions are desired. The machine language program runs adapting to the available memory left over by the text for the BASIC Shell program. Because all variables may be overwritten by the assembly, it is neccesary to execute a CLR statement immediately after the SYS call to the assembler.

The assembler's machine code resides from $1700-$3FFF in bank 1. The only memory regions which are guaranteed to be found intact after an assembly are zero page, the basic text, and memory above the top of basic pointer in bank0.

MACHINE CODE INTERFACE:

The machine code interface uses a set of memory locations at $1300 to pass parameters to the machine code.

An important thing to remember is that placing garbage in these locations can cause the assembler to malfunction. The assembler is a tool to debug your source files, and is therefore very forgiving to information found there. It is not a tool to debug the BASIC shell.

CHANNELS AND SECONDARY ADDRESSES

YOU MAY ONLY USE SECONDARY ADDRESSES FROM 2-7 FOR FILES THAT ARE OPEN TO DISKS BY THE BASIC SHELL PROGRAM.

YOU MAY ONLY USE LOGICAL FILE NUMBERS FROM 2-7 FOR CHANNELS OPENED BY THE BASIC SHELL PROGRAM.

A channel number of zero inhibits output to that channel.

Output channels may be RS232, screen, or serial buss devices. (I/O to cassette is disallowed because memory from the cassette buffer is raided by the assembler for free memory.) All that is required of BASIC is to open and close these channels for this assembler.

LIST_CHANNEL **$1300**

The object channel parameter informs the assembler which logical file number to output the list file data to.

ERROR_CHANNEL **$1301**

The object channel parameter informs the assembler which logical file number to output the error file data to.

If the error channel is different from the listing channel, The error channel will show all listing lines on which errors are detected, and will also show which files are currently being assembled.If the error channel is the same as the listing channel, then the error channel is effectively inhibited because all errors are reported to the listing channel.

Note that information regarding the current include file is inhibited in the listing because this information is displayed in the .INCLUDE statement.

OBJECT_CHANNEL **$1302**

The object channel parameter informs the assembler which logical file number to output the object file data to.

SOURCE_DEVICE_LOW **$1303**

SOURCE_DEVICE_HIGH $1304

HCD65xx can be configured to look for source files on one disk, and if they are not there, search the next disk for the file. These two parameters determine which disks will be searched for source files. The assembler always searches in order of increasing device number.

XREF_DEVICE $1305

The xref device is the device number of the disk drive that the temporary cross reference file is to be created on. The assembler manages this binary file.

LIST_CHANNEL_WIDTH $1306

The list channel width is the maximum number of columns to print on a listing line. Listing lines greater than this value are truncated. Setting this value too high results in improper paging.

SOURCE_FILE_NAME $1307-1317

The source file name is a 17-byte area which should contain the null terminated source file name. As with include files, it is not neccesary to append a .SRC to the end of the file, as this is done internally.

DATE_STRING $1317-$1338

The data string is a 33-byte area which is contained in a null terminated section of text. This text is printed in the listing, at the top of every page, in the date field.

SUPPORTED INSTRUCTION SET

This assembler supports the full 65xx instruction set. The following opcode table lists the opcodes and their associated mnemonics.

```
opcode
.   immediate
.   .   absolute addressing mode
.   .   .   zero page
.   .   .   .   accumulator
.   .   .   .   .   implied
.   .   .   .   .   .   indirect,x
.   .   .   .   .   .   .   indirect,y
.   .   .   .   .   .   .   .   zero page,x
.   .   .   .   .   .   .   .   .absolute,x
.   .   .   .   .   .   .   .   .   .   .relative
.   .   .   .   .   .   .   .   .   .   .   .   indirect
.   .   .   .   .   .   .   .   .   .   .   .   .zero page,y
ADC $69 $6D $65 ... ... $61 $71 $75 $7D $79 ... ... ...
AND $29 $2D $25 ... ... $21 $31 $35 $3D $39 ... ... ...
ASL ... $0E $06 $0A ... ... ... $16 $1E ... ... ... ...
BCC ... ... ... ... ... ... ... ... ... ... $90 ... ...
BCS ... ... ... ... ... ... ... ... ... ... $B0 ... ...
BEQ ... ... ... ... ... ... ... ... ... ... $F0 ... ...
BIT ... $2C $24 ... ... ... ... ... ... ... ... ... ...
BMI ... ... ... ... ... ... ... ... ... ... $30 ... ...
BNE ... ... ... ... ... ... ... ... ... ... $D0 ... ...
BPL ... ... ... ... ... ... ... ... ... ... $10 ... ...
BRK ... ... ... ... $00 ... ... ... ... ... ... ... ...
```

```
opcode
.   immediate
.   .   absolute addressing mode
.   .   .   zero page
.   .   .   .   accumulator
.   .   .   .   .   implied
.   .   .   .   .   .   indirect,x
.   .   .   .   .   .   .   indirect,y
.   .   .   .   .   .   .   .   zero page,x
.   .   .   .   .   .   .   .   .absolute,x
.   .   .   .   .   .   .   .   .   .   .relative
.   .   .   .   .   .   .   .   .   .   .   .   indirect
.   .   .   .   .   .   .   .   .   .   .   .   .zero page,y
BVC ... ... ... ... ... ... ... ... ... ... $50 ... ...
BVS ... ... ... ... ... ... ... ... ... ... $70 ... ...
CLC ... ... ... ... $18 ... ... ... ... ... ... ... ...
CLD ... ... ... ... $D8 ... ... ... ... ... ... ... ...
CLI ... ... ... ... $58 ... ... ... ... ... ... ... ...
CLV ... ... ... ... $B8 ... ... ... ... ... ... ... ...
CMP $C9 $CD $C5 ... ... $C1 $D1 $D5 $DD $D9 ... ... ...
CPX $E0 $EC $E4 ... ... ... ... ... ... ... ... ... ...
CPY $C0 $CC $C4 ... ... ... ... ... ... ... ... ... ...
DEC ... $CE $C6 ... ... ... ... $D6 $DE ... ... ... ...
DEX ... ... ... ... $CA ... ... ... ... ... ... ... ...
DEY ... ... ... ... $88 ... ... ... ... ... ... ... ...
EOR $49 $4D $45 ... ... $41 $51 $55 $5D $59 ... ... ...
INC ... $EE $E6 ... ... ... ... $F6 $FE ... ... ... ...
INX ... ... ... ... $E8 ... ... ... ... ... ... ... ...
INY ... ... ... ... $C8 ... ... ... ... ... ... ... ...
JMP ... $4C ... ... ... ... ... ... ... ... ... $6C ...
JSR ... $20 ... ... ... ... ... ... ... ... ... ... ...
LDA $A9 $AD $A5 ... ... $A1 $B1 $B5 $BD $B9 ... ... ...
LDX $A2 $AE $A6 ... ... ... ... ... ... $BE ... ... $B6
LDY $A0 $AC $A4 ... ... ... ... $B4 $BC ... ... ... ...
LSR ... $4E $46 $4A ... ... ... $56 $5E ... ... ... ...
NOP ... ... ... ... $EA ... ... ... ... ... ... ... ...
ORA $09 $0D $05 ... ... $01 $11 $15 $1D $19 ... ... ...
```

```
opcode
.   immediate
.   .   absolute addressing mode
.   .   .   zero page
.   .   .   .   accumulator
.   .   .   .   .   implied
.   .   .   .   .   .   indirect,x
.   .   .   .   .   .   .   indirect,y
.   .   .   .   .   .   .   .   zero page,x
.   .   .   .   .   .   .   .   .absolute,x
.   .   .   .   .   .   .   .   .   .relative
.   .   .   .   .   .   .   .   .   .   .   indirect
.   .   .   .   .   .   .   .   .   .   .   .zero page,y
PHA ... ... ... ... $48 ... ... ... ... ... ... ... ...
PHP ... ... ... ... $08 ... ... ... ... ... ... ... ...
PLA ... ... ... ... $68 ... ... ... ... ... ... ... ...
PLP ... ... ... ... $28 ... ... ... ... ... ... ... ...
ROL ... $2E $26 $2A ... ... ... $36 $3E ... ... ... ...
ROR ... $6E $66 $6A ... ... ... $76 $7E ... ... ... ...
RTI ... ... ... ... $40 ... ... ... ... ... ... ... ...
RTS ... ... ... ... $60 ... ... ... ... ... ... ... ...
SBC $E9 $ED $E5 ... ... $E1 $F1 $F5 $FD $F9 ... ... ...
SEC ... ... ... ... $38 ... ... ... ... ... ... ... ...
SED ... ... ... ... $F8 ... ... ... ... ... ... ... ...
SEI ... ... ... ... $78 ... ... ... ... ... ... ... ...
STA ... $8D $85 ... ... $81 $91 $95 $9D $99 ... ... ...
STX ... $8E $86 ... ... ... ... ... ... ... ... ... $96
STY ... $8C $84 ... ... ... ... $94 ... ... ... ... ...
TAX ... ... ... ... $AA ... ... ... ... ... ... ... ...
TAY ... ... ... ... $A8 ... ... ... ... ... ... ... ...
TSX ... ... ... ... $BA ... ... ... ... ... ... ... ...
TXA ... ... ... ... $8A ... ... ... ... ... ... ... ...
TXS ... ... ... ... $9A ... ... ... ... ... ... ... ...
TYA ... ... ... ... $98 ... ... ... ... ... ... ... ...
```

ASSEMBLER LIMITATIONS

Symbol table size:
The symbol table is stored in bank 1. Its size is 61k bytes. Each symbol requires 8 bytes plus the number of characters in the symbol name. Local symbols always require 11 bytes.

Input buffer size.
All defined macros, any macro expansions, and the input buffers (page length per file), are stored in a heap immediately following the BASIC shell (starting at $4000) and terminating at $8000.

The input buffer is also used for sorting during the cross reference phase. The assembler will perform as many passes through the temporary file as are neccesary to complete the cross reference.

Maximum line length
The assembler truncates input lines at 132 characters. Internal macro expansion lines are also terminated at 132 characters. No errors are reported for truncation.

Discarded control characters
The assembler accepts PETSCII or ASCII text. All control characters ($00-$1F) are removed when the file is read in except the tab and carriage return characters. Tabs are treated in the expected fashion.

Maximum source file size.
Maximum list file size.
Maximum object file size.
Maximum cross reference file size.
Maximum error file size.
These things are limited by the available disk capacity, the amount of paper in your printer, and indirectly, by the symbol table size.

1541/1571 DOS LIMITATIONS

The 15xx series of disk drives have certain limitations which must actively be observed by users of this assembler. Primarily, these relate to specific limitations within the drives which cannot, or should not, be accounted for within the scope of the assembler proper. These are outlined below.

NEVER cause the assembler to attempt to open more than three files on any given drive. If you do, then output files will show a marked tendency to become intermixed, confusing both DOS and the assembler.

You cannot request two output files to a single drive. Note that requesting a cross reference (xref) requires a temporary cross reference file to be generated on your drive, and that this counts as an output file.

Legal combinations for a single drive output file with multiple input files (has >INCLUDES) are:	Legal combinations for a single drive output file with one input file (no .INCLUDES) are:
object file only	object + xref files
list file only	object + list files
xref file only	object + error files
error file only	xref + list files
	xref + error files
	list + error files

Note: Opening more than two files for simultaneous read access on a given drive is trapped by the drive and results in the third file being ignored. Note that the assembler buffers input files and therefore it may be possible to to do this if the buffering works out correctly.

CHAPTER 3

C128 OBJECT FILE LOADER

INTRODUCTION TO THE LOADER

The C128 Object File Loader is a program used to load object files created by various assemblers into memory.

Typically, object files are loaded using the loader and then saved using the ROMmed C128 machine code monitor.

The loader resides in bank zero from $1300-$17FF. Attempts to load files into this space will result in an error termination.

If you want to load object code into this space, load the object into another memory location and then use the machine language monitor to transfer the binary image into its proper location before saving the file to disk.

GETTING STARTED

Start the loader by inserting Disk 1, side 1 into your system disk drive (unit 8) and typing **RUN"*"**. Then choose LOADER from the menu.

The loader prompts you for three pieces of information.

1. **OBJECT FILE NAME**

   This is the name of an object file in MOS technology format. This file may be ASCII or PETSCII in upper or lowercase.

2. OBJECT FILE DRIVE

This is the unit number of the drive where the object file resides.

3. LOAD ADDRESS

Object files contain information about the address(es) where data is supposed to be loaded. Normally, the loader loads object files into bank 0, at the address(es) specified by the object file.

If you want to load the information into another region of memory, specify the address of the first byte of the region. The format of the load address is one to five hex digits in the format accepted by the machine language monitor.

In function, the load address parameter is identical to the OFFSET parameter found in other loaders. In use, the load address parameter is easier to use because it eliminates the need to calculate offsets, and allows you to think about where you want the data to go.

During the load process, the loader displays the regions of memory the loader is writing information to.

Errors that occur during the load process are displayed on the screen; they are self-explanatory.

MOS TECHNOLOGY OBJECT FILE FORMAT

The object file format used by all Commodore assemblers and loaders is the MOS (Metal Oxide Semiconductor) Technology format.

The MOS format consists of a series of DATA records followed by an end of file (EOF) record. Each record consists of a semicolon followed by a set of ASCII or PETSCII characters. All characters between records are ignored.

DATA records are organized as follows:

;NNAAAADDDDDDDDDDCCCC

Where

;	denotes the start of a record.
NN	is a two-character hexadecimal representation of the (non-zero) number of data bytes in this record.
AAAA	is the four-character hexadecimal representation of the sixteen-bit address for the data in this record.
DD	Each DD is a two-character hexadecimal representation of an eight-bit value to be loaded at the appropriate address.
CCCC	represents the checksum of eight-bit bytes in this record.

EOF records are organized as follows:

;00RRRRCCCC

Where

;00	indicates this is an EOF record.
RRRR	represents the total number of records in this file.
CCCC	represents the checksum of the record count.

CHAPTER 4

C128 ROM Differences

The following chapter contains information of interest to both developers of C128 compatible software and developers of special application programs for serial disk drives.

To maintain compatability with all versions of the C128, you should read this section carefully. Some of the differences occur in the C128 itself, others occur in the 1571 disk drive. Only programs which call ROM routines directly will be affected by ROM changes. Applications which call DOS or Kernel routines from the top level or through a jump table are not affected.

This chapter is divided into five sections:

1571 ROM Changes

1. The Set_Overflow flag was not disabled when exiting the 1571 controller. This is the cause of many seemingly random and difficult-to-reproduce problems. This particularly explains most of the Relative file problems. The Set_Overflow flag is now disabled on exit.

2. The TSTATN routine caused 'DEVICE NOT PRESENT' errors because the IRQ source was never cleared. This has been changed.

3. The BAM swapping code has been changed. A BAM swap occurs when all the buffers have been allocated by an application. The DOS then frees up the BAM buffer by marking it out of memory. Later when re-reading the BAM, the side-one BAM is also read in. Hence, if the side-1 BAM had been 'dirty', it would be corrupted by the BAM swap. A new swap flag has been added at RAM location $1B6. The DOS will rebuild the side 1 BAM upon a reread. This usually occurs with multiple files open and sectors being allocated on both sides of the disk.

4. Previously BAM allocation on side one would cause the BAM image to be written every access. This has been changed.

5. The SAVE-@ (SAVE with replace) variable, NODRV, is now 16-bit addressable. The STLBUF routine now steals the buffer locked by drive one.

6. Previously an active collect in 1541 emulation mode would write a zero to the double sided flag in the BAM. This has been changed.

7. Applications which addressed tracks beyond 35 (on any side) previously used incorrect bit cell densities because the table TRKNUM only listed up to track 35. The tables TRACKN and WORKTABLE replace TRKNUM and WORKTBL, respectively, and extend the tables to track 40.

8. A 1541 ROM revision changed the variable TIM from $3A to $20 which caused problems for some applications. TIM is the IRQ rate ($3A = 15 ms). It is once again $3A, like the original (-05) 1541 ROM.

9. USEDTS returned a 'BLOCK NOT AVAILABLE' status when the number of blocks free was equal to 3. This has been changed.

10. Previously during a BURST GCR FORMAT the activity led was not activated. This has been changed.

11. The 1571 BURST LOAD routine would not load 'Locked Files'. This has been changed.

12. Previously while loading files using the BURST LOAD routine, retries were not performed. This has been changed.

13. Motor acceleration time for the MFM controller was set long, which affected performance when reading and writing in MFM format. This has been changed.

14. Previously determining whether a diskette was double-sided or single-sided GCR would take a long time due to valid sync pulses found on 'flippy diskettes' and MFM diskettes. This has been changed.

15. SPINP interrupts from SP (fast serial input) were not enabled optimally. This has been changed, but has no affect on the operation of the serial bus.

16. Previously if a copy was performed addressing drive one, the error channel would return status '00,OK,00,00'. This has been changed so that an error is returned.

17. Previously the ROM test did not check the first page in ROM memory. This has been changed.

18. The ROM checksum at $8000 and $8001 is now $F2, $68.

19. The ROM signature at $C000 is now $38.

C/128 ROM Changes

The current C/128 ROM set bears the following part numbers. (The PCB socket number is valid only for original PCBs).

1. #318018-04 --> BASIC LOW
   ($4000-$7FFF, U33) checksum= 9A40

2. #318019-04 --> BASIC HIGH, MONITOR
   ($8000-$BFFF, U34) checksum= 6F80

3. #318020-05 --> EDITOR, KERNEL, CP/M
   ($C000-$FFFF, U35) checksum= EEC4

4. #318022-02 --> BASIC, MONITOR
   ($4000-$BFFF, U34) checksum= 09C0

5. #318023-02 --> EDITOR, KERNEL, CP/M
   ($C000-$FFFF, U32) checksum= F324

(Note: #5 also contains the C/64 ROM code.)

Each 16KB ROM block contains a small patch area for changes, and all patches described below have been accomplished such that any particular change will never affect more than one ROM. Similarly each ROM contains a revision status byte (at $7FFE, $BFFE, and $CFFE) which software can test to determine the version of the host system. The "original" ROMs contain $00 in these locations and the ROMs described herein contain $01. Each ROM has had several changes, as summarized on the following pages.

The following are the differences in the revisions of the BASIC LOW ROM ($4000-$7FFF) of the C128.

1. LIST and DELETE commands - Previously they did not report as errors certain non-numeric characters passed as arguments, such as 'LIST A'. This has been corrected totally in-line by adjusting a relative branch in the 'RANGE' routine.

2. CIRCLE command - Previously an unspecified Y - radius defaulted to the X-radius value. However, the X-radius value may have already been scaled for the X-axis before the Y-radius default value was calculated. This has been changed totally in-line by scaling the radii after the defaults have been established.

3. RS-232 STatus - Previously accessing ST after RS-232 I/O resulted in an incorrect status being returned from, and a zero written to, location $10A14, the BASIC variable area. This was caused by BASIC calling the Kernel routine 'READSS' with RAM bank 1 in context. This has been changed in-line.

4. CHAR command - Previously using CHAR with the 80-column text screen (GRAPHIC mode 5) resulted in RAM, not I/O, access at locations $D600 and $D601 of RAM bank 0 (the BASIC text bank) due to BASIC calling the Editor PLOT routine without the I/O block in context. This has been changed with two patch subroutines.

5. RENUMBER command - Previously the pass 2 routine, which was to pre-scan BASIC text and report 'out of memory' errors, did not find all errors. This has been changed using a patch subroutine.

6. DELETE command - Previously this command did not limit-check itself when moving down BASIC text. This has been changed with a patch subrouitne. In addition, the DELETE command exited to MAIN via 'JMP', effectively ending the evaluation of the current command string. This has been corrected by substituting an 'RTS', allowing direct commands like 'DELETE 10: PRINT"DELETED LINE 10"' to work correctly.

7. PLAY command - Previously the SID frequency tables were not exactly NTSC concert pitch. Also, there was no provision for adjusting the frequency for PAL systems. This has been corrected by changing the (NTSC) frequency tables, creating new PAL tables, and utilizing patch code to select from the appropriate table as determined by the Kernel PAL_NTSC flag.

8. BASIC ERROR handler - Previously the error-handler did not clear pending string temporaries when an error was TRAPped. This has been changed with a patch to reset TEMPT to TEMPST.

9. The powerup copyright notice has been updated to 1986, which will serve as an immediate visual indication of the ROM version. Also, a new notice has been placed at $7FC0.

10. The ROM signature at location $7FFC and $7FFD (lo/hi) is $8DEF.

11. The ROM revision byte at location $7FFE, has incremented from $00 to $01.

12. The ROM checksum byte at location $7FFF, has changed from $4C to $61.

The following are the differences in the revisions of the BASIC HIGH/MONITOR ROM ($8000-$BFFF) of the C128.

13. RSPRITE and RSPPOS functions - Previously these functions accepted sprite numbers in the range 9-16. without returning an error. This has been changed in-line. An illegal quantity error is reported for numbers outside this range.

14. PRINT USING command - Previously there was an anomaly involving the use of floating money symbols ('$') and commas. The command 'PRINT USING "#,##$.##"; 123.45', for example, resulted in the output '$,123.45'. This has been changed utilizing a patch subroutine which checks for the '$,' occurrence and substitutes a fill character (_$) whenever found.

15. Relative Coordinates - Previously, a negative relative co-ordinate for a graphic command resulted in an illegal quantity error. This has been changed totally in-line by substituting a different subroutine call. This change affects the BASIC commands LOCATE, DRAW, PAINT, BOX, CIRCLE, GSHAPE, and SSHAPE. This change also allows negative absolute coordinates to be accepted (previously they resulted in an illegal quantity error), although the legal range remains a 16-bit value.

16. DOPEN and APPEND commands - Previously it was possible to open two or more disk channels with the same logical file number without incurring an error report. This has been changed totally in-line.

17. MATH package - A change in the (F)MULT routine caused some rounding errors (such as 2^15 = 32768.0001). This has been resolved in line by changing (F)MULT in a new way.

18. A copyright notice has been placed, starting at $BFC0.

19. The ROM signature at location $BFFC and $BFFD (lo/hi) is $CDC8.

20. The ROM revision byte at location $BFFE. has incremented from $00 to $01.

21. The ROM checksum byte at location $BFFF, has changed from $3A to $C5.

The following are the differences in the revisions of the EDITOR / KERNEL / CP/M ROM ($C000-$FFFF) of the C128.

22. CAPS LOCK Q - Previously the keyboard matrix decode table caused a lower-case 'Q' to be passed when the keyboard was in CAPS LOCK mode. The table has been changed. A new value has been substituted for upper-case 'Q'.

23. FUNCTION KEYs - Previously the funcion key handler, part of the SCNKEY routine at CKIT2, did not detect a funtion key string pending. This has been changed via a patch routine, which will ignore new function key depressions until the string in progress has been output (i.e., KYNDX = 0). Also, DOPFKY now exits via SCNRTS, instead of simply RTSing.

24. IOINIT system initialization - Previously the RS-232 pseudo-6551 registers were not initialized because these values are given by the user whenever RS-232 channels are OPENed. However on the 64, these locations were cleared by other system routines. So nothing bad happened if you did not initialize as required. IOINIT on the 128 now works the same way as the 64. The registers are now initialized to default to: no parity, full duplex, 3-line, 1-stop bit, 8-bit words and 300 baud, via a patch subroutine.

25. IOINIT PAL system initialization - Adjustments have been made to the 8563 initialization values for PAL systems. The PAL horizontal total (register 0) changes from $7E to $7F. The PAL vertical total (register 4) changes from $27 to $26. These changes shift the cycle time from 20.320us to 20.032us. This change required a patch subroutine, and a change to VDCTBL.

26. BASIN system call - Previously attempting input from a logical channel to the screen (e.g., via INPUT#) resulted in line too long errors. This has been changed utilizing a subroutine patch to preserve bit-7 of CRSW, which serves as a flag to the Editor that a virtual end-of-line has been reached. Also, TBLX is copied to LINTMP to correctly locate the current cursor line for the Editor. Please note that switching between the 40 and 80-column text screens, opening and closing windows, or clearing text screens can confuse logical screen channels. The Editor variable LINTMP ($A30) is a global, not a local, variable. Users can POKE LINTMP with the logical screen line number before INPUT#'s.

27. OPEN RS-232 system call - Previously it was possible to receive a carry-set status, normally indicating an error, when no error existed after OPENing an RS-232 channel. This has been changed totally in-line by a modification to the code which checks for the proper X-line hardware states.

28. LOAD system call - Previously the normal load (a.k.a. SLOW) routines did not preserve the starting address of any LOADs, which made the BASIC 'BOOT "file"' command form malfunction on the 1541. This has been changed via a patch subroutine, which saves the starting address of all LOADed files at SAL and SAH, the same place the fast (a.k.a. BURST) load routines do.

29. DMA system call - Previously the Kernel forced the I/O block into the user's memory configuration at all times, which is no longer necessary and limits the functionality of the RAM expansion cartridge. This has been changed by a ROM patch routine, which affects Kernel DMA system calls and the BASIC FETCH, STASH, and SWAP command.

Also, previously it was possible for an IRQ to occur between the 'arm DMA' and 'trigger DMA' sequences, which caused DMA to occur with the system configuration in context regardless of desired configuration. The instructions, 'PHP/SEI...PLP' have been inserted around the 'JSR' to DMA RAM code at $3F0. Applications using the DMA code at $3F0 should do likewise.

Another change has been made made to allow DMA access to all RAM banks by using the VIC bank pointer found in the MMU RAM configuration register ($D506, VA16=bit-6 and VA17=bit-7). Applications using the Kernel routine at $FF50 will inherit these changes automatically. Please note that NMI interrupts can interfere with DMA acces as they cannot be masked.

30. A copyright notice has been placed, starting at $CFC0.

31. The ROM location $CFF8 is reserved for national character ROM checksums. This does not apply to US ROMs, which contain $FF here.

32. The ROM location $CFF9 is now reserved for country codes. The US ROMs contain $FF here.

Country Code	Value in $CFF9
Germany/Austria	$00
France/Belgium	$02
Switzerland	$03
Finland/Sweden	$04
Norway	$05
Italy	$06
Denmark	$07
Spain	$08
U.S./U.K.	$FF

33. The ROM location $CFFA and $CFFB (lo/hi) contain the national character set signature. This does not apply to US ROMs, which contain $FFFF here.

34. The ROM signature at location $CFFC and $CFFD (lo/hi) is $8F76.

35. The ROM revision byte at location $CFFE, has incremented from $00 to $01.

36. The ROM checksum byte at location $CFFF, has changed from $C3 to $3C.

37. The Kernel revision byte at location $FF80 has incremented from $00 to $01.

1. DMA interface - It should also be noted that DMA hardware is unreliable at 2MHz clock speeds and consequently the user must insure 1MHz (SLOW) mode is used before any DMA operations are performed. Also, NMI interrupts cannot be masked and should be disabled or somehow avoided. RS-232 operations use NMIs; the remote should be XOFFed or the channel disabled before DMA operations are performed.

2. IRQ handler - It is possible for the Kernel IRQ handler to perform a keyscan when the IRQ is not the Kernel's. This was not changed to avoid problems with older software which may be taking advantage of the unintentional keyscans.

3. IRQ and NMI handlers - The Kernel forces the system bank into context before taking the RAM indirect vectors to the actual interrupt handler.

4. SAVE-to-disk - It is not possible to SAVE the last byte of any memory bank (e.g., RAM at $FFFF), because the SAVE routine requires you to specify the end of the area to be SAVEd as the ending address PLUS ONE ($FFFF+1 -> $0000). This is a problem found on all CBM 65xx systems.

5. SAVE-to-cassette - It is not possible to save the last page of any memory bank (e.g., RAM at $FF00-$FFFF) to tape. The tape handler hangs with the motor running until the user STOPs it. This is a problem found on all CBM 65xx systems except the Plus-4.

6. SAVE and LOAD - While program SAVEs correctly save the 16-bit starting address for future LOADs, they do not save the memory bank.

7. STOP/RESET monitor entry - It is not possible to enter the Monitor directly via the STOP/RESET sequence from BASIC and then eXit back to BASIC without incurring a 'cold' BASIC initialization. The alternative, taking the BASIC 'warm' start route, would result in a system crash if BASIC had not been properly initialized. If BASIC was running before the STOP/RESET, you may instead POKE $C1 into location $A04 (INITSTATUS) and then eXit.

8. Monitor 'H' (hunt) command - Because the editor performs various translations on data read from the screen, it is not possible to Hunt for certain CBM characters, such as pi and all reverse-field characters.

9. BOX command - Because of the algorithm used, BOX has a range -16384 to +16383 (unscaled). The algorithm uses parameters that are twice those given and divides down the result before plotting. Thus it is possible for very large (unscaled) positive coordinates to result in large negative plots. To avoid this use SCALEing, user range-checking, or avoid BOX and use either DRAW or CIRCLE commands instead.

10. RDOT, PEN, and RSPPOS functions - These BASIC functions return the current pixel cursor, lightpen and sprite positions, respectively, but the values they return are unSCALEd. Changing this is easy but would cause problems for existing applications as well as being incompatible with the C64 VSP and the PLUS-4.

11. FNDEF and GRAPHIC modes - After defining a user function, anything that results in program relocation must be avoided, such as GRAPHIC 'n' or GRAPHIC CLR. Follow the general general rule: define GRAPHIC screens first (then SCALE), then define functions.

12. GETKEY function - The command 'GETKEY A' will accept some non-numeric keys, such as 'E', colon, comma, period, '+', This also occurs with the command 'GET A'.

13. PUDEF and PRINT USING - In some USING format fields such as "$,###.##", the leading '$' or commas are not interpreted as they are considered not part of the numeric field.

14. OUT of MEMORY ERROR - It is possible to hang the system with this error from a running program when there is insufficient memory to contain the string representation of the original line number where the error occurred.

18. MATH package - The binding of operators is such that unary minuses are evaluated after powers. This results in NO error when equations of the form (-4^.5) are evaluated (square root of a negative number).

19. BASIC DOS commands, such as DOPEN and APPEND, limit filenames to 16 characters maximum. However, when the name string includes the filetype, such as "LONGLONGLONGLONG,P", BASIC reports a FILENAME TOO LONG error.

20. CIRCLE command - In multicolor mode, CIRCLE calculates the default Y-radius based upon twice the X-radius. This is done in order to maintain compatibility among the C64 VSP, PLUS-4, and C128.

21. TAPEs written in FAST mode are occasionally hard to read in SLOW mode. Users should take care to use tapes only in SLOW (1MHz) mode so that the tapes can be read on PETs, 8032s, C64s, etc.

1541-II ROM Differences

The following list describes the differences between the ROMs in the 1541 (-05) and 1541-II (251968-03). Please note that this ROM will not work in older 1541 boards which require different sized ROMs.

1. SAVE-@ (SAVE with replace) - The variable NODRV has been changed to a 16-bit addressable variable. The STLBUF routine steals the buffer locked by drive one.

2. Device Not Present - TSTATN now clears IRQ which makes serial bus "device not present" errors more reliable.

3. IRQ disable - Decimal mode is now set (SED) before disabling IRQs (SEI).

4. DOS patches - Block read (B-R), formatting, relative file handling and disk-full error handling have been changed slightly to enhance reliability.

5. A new copyright notice has been added.

6. The ROM checksum adjustment byte at $C001 is now $E0.

7. The ROM checksum adjustment byte at $FFE5 is now $EB.

1541C ROM Differences

The following list describes the differences between the ROMs in the 1541 (-05) and 1541C (251968-02). Please note that this ROM will not work in older 1541 boards which require different sized ROMs.

1. Density table - The bit cell density table, TRKNUM, has been extended from 35 to 40 to enhance reliability.

2. SAVE-@ (SAVE with replace) - The variable NODRV has been changed to a 16-bit addressable variable. The STLBUF routine steals the buffer locked by drive one.

3. Device Not Present - TSTATN now clears IRQ which makes serial bus "device not present" errors more reliable.

4. IRQ disable - Decimal mode is now set (SED) before disabling IRQs (SEI).

5. DOS patches - Block read (B-R), stack operation, relative file handling and disk-full error handling have been changed slightly to enhance reliability.

6. A new copyright notice has been added.

7. The ROM checksum adjustment byte at $C001 is now $46.

SX-64 ROM Changes

Following is a list of the differences between the C64 and SX-64 kernel ROMs. The BASIC ROMs of these two machines are the same.

There are quite a few modifications to the kernel ROM however:

Kernel Address		Alteration
$E479-$E493	(58489-58515)	The power up message has been changed to show SX-64 instead of Commodore, and v2.0 instead of v2.
$E4AC	(58540)	Also in power up message.
$E4D3-$E4DC	(58579-58588)	Added routine replacing blank memory.
$E535	(58677)	New cursor color (blue instead of lt. blue).
$E57C-$E5F5	(58748-58869)	Set screen pointers routine.
$E5EF	(58863)	Input from keyboard routine (branch).
$E5F4-$E5F5	(58868-58869)	New address for Shift-Run/Stop.
$E622-$E623	(58914-58915)	New address for retreat cursor routine.
$EA07-$EA12	(59911-59922)	Change to clear screen-line routine.
$ECD9-$ECDA	(60633-60634)	New default screen and border colors.
$EF94-$EF96	(61332-61334)	Jump from routine to added routine at location $E4D3 (58879).
$F0D8-$F0E6	(61656-61670)	New default string for Shift-Run/Stop: LOAD"*",8 <ret> RUN <ret>.
$F387	(62343)	Changed branch in open file routine.
$F4B7	(62647)	Changed low byte of a JSR in load routine.
$F5F9	(62969)	Changed branch in save routine.

SECTION II
C64 and C128 MODES

CHAPTER 5

C64 Tools

SPRED: The Sprite Editor

INTRODUCTION

This program allows you to easily create and modify sprites on the 64 or in the 64 mode of the 128. Many features which allow easy manipulation of the sprites are included. There are also commands that will help you use the sprites that you have created in other programs.

GETTING STARTED

If you are using the 128, go into 64 mode. Run the program by typing LOAD "SPRED",8 and Return. Once the program has loaded, type RUN and press Return.

The Screen Format

The screen is divided into 4 work areas. The large box on the left of your screen is the sprite editing area. This box is 24 x 21 and shows the complete sprite that you have created.

The work area in the upper right corner of your screen is the status box and shows the current settings for your colors, color mode, x-y expansion and the range for sprite loading and saving.

The lower right corner of you screen shows the sprite as it would actually appear in you application program.

The work area at the bottom of your screen is the command box. Also shown here are the row and column position of the cursor in the sprite, the cursor color, sprite number, sprite mode and other information.

CREATING A SPRITE

To create a sprite, move the cursor around in the sprite editing area with the cursor keys. Use the period-key to turn on a pixel or the space bar to turn off a pixel. Change colors with the F1 and F7 keys.

If you make a mistake and get stuck in a command prompt, just hit the Run/Stop key. Run/Stop means oops!

If you forget a command, just press the H key and a help menu will appear which shows all the commands that the Sprite Editor will accept. When you are done looking at the help menu, press Run/Stop to exit.

Up to 208 sprites may be created and managed at once by the Sprite Editor. To edit a particular sprite press E followed by the number of the sprite.

Saving a Sprite

To save a sprite, press F6. A sprite may be saved as binary data, as 6502 assembly source code or as BASIC data statements.

Press B to save the sprites as binary data which can be loaded back into the editor later. You must enter either a sprite number or a load address. The extension ".B" will be added to the file name you provide.

Press S to save the sprites as assembly source code byte statements. The extension ".S" will be added to the file name you provide.

Or press D to save the sprites as BASIC data statements. You can append the data statements to an already existing program or save them alone. You must enter the BASIC starting line number and line increment. If you save the sprites as BASIC data, the extension ".D" will be added to the file name.

Whatever method you use to save a sprite, you must enter a file name. Also be sure that the range parameter is set correctly in the status box. If the range parameter is set to 000-003, only sprites 0-3 will be saved.

If you want to save your work and come back to do more editing later, you must save your sprites as a binary type file.

Loading a Sprite

To load a sprite, press F8. Only binary sprite files can be reloaded into the Sprite Editor. Next enter the type of load you want to do by pressing R, A or L.

Press R to load the sprites only into the currently selected range. Press A to load all the sprites that were saved starting at the beginning of the current range. Or press S to select the sprites you want to load. If you press S, you will be able to accept or reject each sprite as it is loading.

NOTE: You do not need to type the file name extension to re-load your sprites.

The Directory

To view the directory, press F4 and then enter $ (dollar sign), Return. When you press F4, a DOS command line appears. In addition to viewing the directory, you can send disk commands such as scratch or rename to your disk drive. For instance to scratch a file, enter S0:<file name>.

EXTENDED MODES

Most of the time you will want to edit just one sprite at a time. In that case you would use the single sprite mode of the Sprite Editor. But the Sprite Editor will allow you to join or overlay sprites as well. In addition, the extended view mode of the Sprite Editor will allow you to create simple animation sequences.

Joined Sprite Mode

To activate the joined sprite mode, press J. SPRED will prompt you to enter the size of the sprite you want to create . Joined sprite mode will allow you to make a compound sprite up to 4 times as big as a normal sprite. It does this by attaching the sprite data of two or more consecutive sprites at the edges.

A joined sprite may be as big as 48 x 42. You cannot see a sprite this big all at once in the sprite editing area. However you can use the cursor keys to reposition the edit area. The edit area will scroll as needed to any position over your compound sprite.

Overlaid Sprite Mode

To activate overlaid sprite mode, press O. SPRED will prompt you to enter the number of sprites to be overlaid. Overlaid sprite mode will allow you to create a special compound sprite in which one sprite appears directly on top of another. This technique provides a way to create a sprite with more colors than are usually allowed.

This trick is accomplished by selecting one color set for the sprite on top and a differing set of colors for the spriteunderneath. Wherever you can see through the sprite on top (that is, wherever a sprite pixel is set to the background color), you will be able to see the colors of the sprite underneath.

ANIMATION

You can use the sprite editor to create simple animation sequences. Each sprite that you define will represent one frame in your animation. Your sequence can have up to 208 frames, although smooth animation can be created with as few as four frames.

To view the animation, use R to set the range of your sequence. Then press V and select the animation type - either oscillate or rollover. Rollover will show each frame in your sequence and then start over at the first frame. Oscillate will show each frame in your sequence and then reverse the order back to the first frame. Next select the number of frames/sprite. This setting controls speed of playback. The bigger the number, the slower the playback.

Remember, the V command will playback only the frames specified by the range parameter. If your range is set to 000-000, you are not going to see anything!

Extended View Mode

To activate extended view mode, press Shift-V. Extended view mode will allow you to create animation sequences that control up to eight different sprites in each frame with up to 638 frames. This is the only way to create an animation sequence which contains a joined or overlaid sprite.

When you activate extended view mode, the screen changes. The sprite editing area disappears and becomes the animation playback window. A set of sequence data is shown in the lower right of your screen.

Each sequence frame may contain up to eight sprites. Hence, there are eight slots in the sequence data area. Enter the sprite number (0-207) and the x and y coordinates of the sprite in one of these eight slots. It is not necessary to fill all the slots.

To enter the frame sequence data, first select a sprite number by pressing E and entering a sprite number, 0-207. Next enter a number, 1-8, for the slot in the sequence data area that you wish to fill. The sprite number will appear in the # column of the sequence data slot you have selected. Now use the cursor keys to position the sprite at the x,y coordinate that you want. When you press the cursor keys, your sprite will move in the animation playback window.

When you have finished defining the sequence data for one frame press Shift + (plus sign). You may then enter the sequence data for the next frame. When all your frames are defined, use V to playback the animation sequence as described above under "Animation".

SPRED Command Summary

F1 - Increment thru Sprite colors
F3 - Increment thru Multicolor 1
F5 - Increment thru Multicolor 2
F7 - Toggle thru Cursor colors

F2 - Change background color
F4 - Directory and disk commands
F6 - Save Sprite range to disk
F8 - Load Sprites from disk

X - Toggle expansion in X
Y - Toggle expansion in Y
M - Toggle Multicolor mode

S - Enter single Sprite mode
J - Enter joined Sprite mode
O - Enter overlay Sprite mode

\+ - Go to next sprite
\- - Go to previous sprite
E - Edit Sprite #
R - Set Sprite range

Space - Erase at cursor
Period - Fill at cursor or draw a line if origin is set
L - Set line origin
Shift L - Cancel line

C - Copy a Sprite
Shift C - Copy a range of sprites
Shift S - Swap current Sprite

V - View a Sprite sequence (Animate)
Shift V - Activate extended view mode

I - Insert a column
D - Delete a column
Shift I - Insert a row
Shift D - Delete a row
INS/DEL - Insert or delete space

1 - Edit Sprite 1 in overlay mode
2 - Edit Sprite 2 in overlay mode
3 - Edit Sprite 3 in overlay mode
4 - Edit Sprite 4 in overlay mode
^ - Redisplay grid in overlay mode

Home - Move cursor to top left corner of grid
Home Home - Move cursor to top left of whole Sprite
Shift Home - Clear the Sprite
Return - Move cursor to left side

G - Toggle global mode
H - Help
< - Toggle repeat key
B - Change border color
N - Toggle number/color for status display

F - Flip a Sprite on y-axis
Shift F - Flip a Sprite on x-axis
Shift A - And a Sprite
Shift O - Or a Sprite
Shift R - Reverse a Sprite

Run/Stop - Return to edit mode

Extended View Mode Commands

Shift V - Return to single sprite mode
E - Edit a given sprite
Shift E - Edit a given sprite

1-8 - Enter sequence data for sprite priorities 1-8
0 - Turn current sprite off (except #1)
R - Set sequence range
Z - Zero range of sequences
Shift Z - Zero all sequences

\+ - Move to next Sprite
\- - Move to previous Sprite
Shift + - Move to next sequence
Shift - - Move to previous sequence

C - Copy given sequence to to the current one
Shift C - Copy sequence range
F8 - Load or Save sequences
V - View current range of sequences

CHARACTER EDITOR
(CHRED)

INTRODUCTION

This program allows editing of single characters, or editing four characters at a time (a 16 bit by 16 bit character). When editing expanded charactes, all character editor commands are adjusted to operate on the expanded character. More details on expansion can be found under the X(pand) command.

SCREEN FORMAT

The screen is basically divided into five areas by the character editor program. The first is the large character creation box. Here the character is formed by using the "*" and " " symbols. In this box, an entire character is represented.

The second area is the information box. Current character parameters are displayed here. Also, when a command needs a parameter to operate, a cursor will appear at the appropriate place in this box.

The third area is the strip below the creation box. Here is where the entire current character set is displayed. Also, the character currently being edited is highlighted in black.

The fourth area is the display area, where the current character appears.

The fifth area is the command menu, where most of the commands are displayed.

General Notes: The first letter of a command is enough. When a command requires an input (like character number), a cursor will appear in the information box. Type the answer there, ending it with a return. If you type an incorrect letter, use the delete key. If your response is illegal, the program will ignore it, keeping the old value of the parameter.

THE INDFORMATION BOX

The information box contains the following information:

	PARAMETER	POSSIBLE VALUES
1.	Character number	0-63
2.	Name of current file	any 5 letters
3.	Assumed address	any 4 digit hex number
4.	Range	0-63
5.	Type of character displayed	HIRES or MULTI
6.	Foreground color	any of the 16 colors
7.	Multi-color register 0	any of the 16 colors
8.	Multi-color register 1	any of the 16 colors
9.	X expand	YES or NO

10. The bottom blank line of the information box is used for other command inputs that don't need to be continually displayed.

COMMANDS

1. E(dit) : requests a character number from 0 to 63. That character becomes the current character. The current appearance of the character is displayed both in the creation box and the current display area.
2. N(ext character) : selects the next character as the current character.
3. T(ype) : selects between displaying the current character as a hires (h) or a multi-color (m) character.
4. M(ove here) : asks for a character number. That character is copied into the current character. Note that this operation destroys the current character.
5. C(olor) : allows choice of colors. A cursor will appear at each of the color parameters in turn. Type the three letter abbreviation for the color of your choice for each register in turn.
6. X(pand) : selects either 8 bit by 8 bit characters or 16 bit by 16 bit characters. Answer YES or NO (or Y or N) to select expansion. When expansion is selected, four contiguous characters will be treated as a single entity for the purpose of all commands. The creation box will be expanded. The characters are edited in the following format.

   1 3
   2 4

7. F(ont) : masking function from the Commodore 64 ROM. The program asks if you want to mask all the characters. Answering Y replaces the current character set with the upper case character set from ROM. If you answer N (or just hit RETURN), the program will ask for a character number (from 0 to 63). That character from ROM will be copied into the current character.
8. R(ange) : sets a range of characters for use with SAVE, LOAD, BYTE, and DISPLAY commands. The form is ##:## (i.e. 0:12), for first and last characters to be affected by an operation.
9. O(r) : like move, but doesn't erase the current character first.
10. A(ddress) : this command selects the assumed address of the character data. This is used when the characters are saved. If they are reloaded by a BASIC load command, that is where they will load.
11. S(ave) : asks for a file name, then saves the current range of characters to disk, as a program file using the assumed address.
12. L(oad) : asks for a file name, then loads the current range of characters from disk into the work area from that program file.
13. B(yte) : asks for a file name, then saves the current range of characters to disk, as a sequential file compatible with Commodore's Assembler Development System.
14. Q(uit) : exits the program.
15. V(alue) : displays the values of the bytes which make up the current character. These values appear next in the character display area. This display lasts until any key is hit. Then the normal character display reappears.
16. H(ex flag) : toggles whether the V(alue) display will be in decimal or in hex.
17. F3 : step through possible screen colors.

EDITING COMMAND KEYS

These commands operate on the current character in the creation box.

1. * : places a dot at the current cursor position.
2. SPACE BAR : places a space (removes a dot) from the current cursor position.
3. CRSR RIGHT : moves the cursor one position to the right.
4. CRSR LEFT : moves the cursor one position to the left.
5. CRSR UP : moves the cursor one line up.
6. CRSR DOWN : moves the cursor one line down.
7. RETURN : moves the cursor to the start of the current line.
8. HOME : moves the cursor to the top left corner of the character.
9. CLR : erases the current character.
10. RVS : reverses the current character.
11. INST : moves all dots on the current one place to the right.
12. DEL : moves all dots on the current line one place to the left.
13. + : moves all lines from the current line to the bottom line one line down.
14. - : moves all lines from the current line to the top line, one line up.

COLOR CODES

BLK	WHT	RED	CYN	PUR	GRN	BLU	YEL
ORN	BRN	RD2	GY1	GY2	GN1	BL2	GY3

SIDMON 64

SIDMON is a program that allows you to create sounds using the 6581 Sound Interface Device of the Commodore 64. It is difficult to experiment with SID sound parameters by poking from BASIC. SIDMON does all the poking for you making it easier to create the sound you want.

SIDMON commands are usually the first letter of the register name. In some cases the letter is already taken by another command, so the second letter is used instead. The proper command letter will be highlighted in reverse on the screen.

The most important command is Gate. Press G + to hear the sound you have created. This "gates on" the currently selected voice. Press G - to turn the sound off.

The sound you hear depends on the data values in the various SID registers. Here is a summary of those registers.

Frequency and Envelope Control Registers

Frequency - This register sets the number of sound waves per second produced by SID. A high number in this register produces a high pitched tone. A small number produces a lower tone.

Pulse Width - This register forms a number which linearly controls the pulse width when a pulse wave form is selected.

pulse width

Attack - The attack register determines how rapidly the output of the voice rises to peak amplitude after the voice is gated on.

Decay - The decay cycle follows the attack cycle. The number in the decay register determines how rapidly the voice falls from peak amplitude to the selected sustain level.

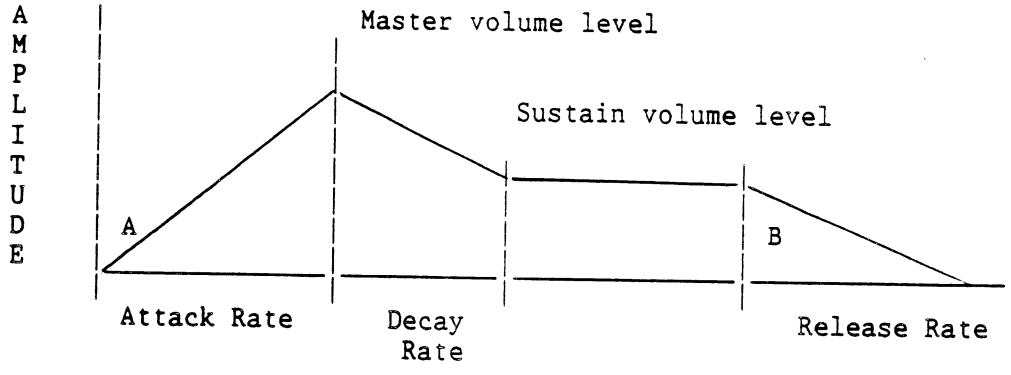
Sustain - The sustain register determines the amplitude level of the sustain portion of a sound. The sustain will last as long voice is gated on.

Release - Release is the final phase of a sound. The number in this register determines how rapidly the voice falls from peak amplitude to zero after a voice is gated off.

Master Volume - The master volume register selects one of 16 overall volume levels for the final composite sound. Volume 0 is off. Volume 16 is loudest.

The ADSR enevlope registers determine how a sound changes in amplitude (volume) over time. By adjusting the values in these registers you can adjust the tonal quality of a sound.

The ADSR Envelope



Voice Control Register Bits

Gate - When the gate bit for a voice is set to one, the attack-decay-sustain cycle begins. When the gate bit is reset to zero, the release cycle begins. To hear a sound you must "gate on". Press the G key followed by the + key.

Sync - When sync is set to one, the frequency of the current voice is synchronized with another (1 with 3, 2 with 1, 3 with 2). This allows more complex harmonics and "hard sync" effects.

Ring - When the ring bit is set to one, the triangle waveform of the current voice is replaced with a ring modulated combination of the current voice with another (1 with 3, 2 with 1, 3 with 2). Ring modulation is used for bell and gong sounds.

Triangle - The triangle bit selects the tringle waveform (⩘) for the current voice. The triangle waveform has a mellow, flute-like quality.

Sawtooth - The sawtooth bit selects a sawtooth waveform (◢◢) which has a brassy timbre.

Pulse - The pulse bit selects the pulse waveform (⎍) for the current voice. The timbre of the pulse waveform will vary depending on the contents of the pulse width register from a reedy, nasal sound to a bright, hollow sound.

Noise - The noise bit selects a random signal (*) for the waveform which changes at the frequency of the current voice. This can be used to produce hissing or rumbling noises.

Filter Control Registers

Filter - When set to zero, the filter is disabled. When set to a one the selected voice is processed through a filter which alters its harmonic content. SID has only one filter. You may route any or all of the voices through it.

Resonance - This register controls the resonance of a filter. Resonance is a peaking effect which emphasizes frequency components at the filter's cutoff frequency. A higher number will create a sharper tone.

Cutoff - The data in this register will linearly control the cutoff frequency of the filter on the current voice.

Low Pass - When set to a one, this selects a low pass filter. All frequency components above the cutoff will be attenuated. This procudes a full bodied sound.

Band Pass - When set to a one, this selects a band pass filter. All frequency components above and below the cutoff will be attenuated. This procudes a thin, open sound.

High Pass - When set to a one, this selects a high pass filter. All frequency components below the cutoff will be attenuated. This procudes a tiny, buzzy sound.

Cut 3 - When set to a one, this removes voice 3 from the audio path. You should set the cut-3 bit when using voice 3 for modulation purpose to prevent unwanted output from voice 3.

SIDMON Command Keys

To use SIDMON, set the parameters you want in the SID registers and use G + to "gate on" the voice you want to hear. All the SIDMON commands are one or two-key combinations.

The command keys are highlighted in reverse on the SIDMON screen. When you press a command key it appears on the "MODE" line at the top of your screen. Below is a list of SIDMON commands.

CLR/Home - This completely resets SIDMON. All SID registers will be set to their default values. Any changes made are lost.

F1, F3, F5 - The function keys select the voice that you are currently working on. F1 is voice one. F3 is voice 2. F5 is voice 3. The current voice is displayed on the "VOICE" line at the bottom of the SIDMON screen.

Plus/minus - The plus/minus (+/-) keys lower and raise the values in the register selected. For instance to raise the frequency of the current voice, press F followed by +.

M and D - The M and D keys are used to multiply and divide the value in a register by 2. These keys work only on frequency and pulse width. For instance to double the frequency, press F followed by M.

F - The F key is used to select the frequency of the current voice. Follow this key with either +, -, M or D to change the frequency.

P - The P key is used to select the pulse width of the pulse (or square) waveform. Folow this key with +, -, M or D. Remember, pulse width has no effect unless you use the pulse waveform.

ADSR - These keys select Attack, Decay, Sustain and Release. Follow the A, D, S or R keys with + or - to change their value.

G - The G key is used to "gate on" or "gate off" a voice. To hear the voice you have set up, press G followed by +.

Y and I - These keys select sync and ring. Press Y followed by + to turn on sync for the current voice. Press I followed by + to turn on ring modulation for the current voice.

0-4 - The number keys 0 to 4 select the type of waveform for the current voice. Press 1 for triangle. Press 2 for sawtooth. Press 3 for pulse. Press 4 for noise. The 0 key will turn off all the waveforms. Only one wave type can be tuned on at a time.

T - The T key is used to send the output of the current voice the SID filter. Press T followed by + to enable the filter.

U - The U key is used to set the cutoff frequency of the SID filter. Press U followed by + or minus to adjust this register. Remember the cutoff frequency has no effect unless the filter is enbled for the current voice.

E - Resonance is set by pressing the E key followed by + or -. Resonance values have no effect unless the filter is enabled.

V - Use the V key to adjust the overall volume level. Press V followed by + or - to raise or lower the volume.

As an experiment, try re-creating this sound sample:

```
MODE:        ***  SIDMON 1.4  ***
FREQUENCY    PULSE WIDTH
 4968          1024
 5000          2048
 5032          3136

ATK DEC SUS RLS GATE SYNC RING     *
 1   3   13  10   1    0    0    .!..
 1   3   13  10   1    0    0    !...
 1   3   13  10   1    0    0    ..!.

FILTER:        RESONANCE: 0
 0
 0
 0

                          +/- UP/DN/ON/OFF
LOW  BAND  HIGH  CUT       D  DIVIDE BY 2
 0    0     0     0       M  MULTIPLY BY 2

MASTER VOLUME: 15

CUTOFF FREQUENCY: 0

VOICE (F1,F3,F5): 1
```

CHAPTER 6

64 Fast Load #1

Hardware: C64 with a 1541 or 1571

1) 2x speed up, no screen blanking
2) Full error checking
3) Loader resides at $c000-$c318 or can be reassembled.

Normal LOAD time for 112 blocks	Fast Load #1 LOAD time for 112 blocks
1 min 17 sec	39 sec

Known bugs: Will not work with MPS-1200 printer

Fast load #1 works by downloading code to the disk drive. This gives us a quick and dirty DOS to load files. The general purpose Commodore DOS is not needed. Instead, we can work with the drive's job queue directly and use our own serial handshake.

Once all the routines are in place, the user calls the fast load. The first track and sector of the file to be loaded should be in the x and y registers respectively.

On return the carry flag is set if there was a problem with the load. If the load was successful, the carry flag is clear.

The code that does the loading resides in the $c000 block. Because of this, the load will fail if the load address of the file is in the $c000 block. So if you want to load a file into this area , you will have to reassemble the fast load to run in a different area of memory.

```
INS*      PAGE 0001

LINE# LOC    CODE        LINE

00001  0000              ;SCDO."INSTRUCTIONS"
00002  0000              ;2/15/85
00003  0000              ;
00004  0000              ; ***********************************************
00005  0000              ; **        INSTRUCTIONS FOR INSTALLING        **
00006  0000              ; **           1541-FASTLOAD ROUTINE           **
00007  0000              ; ***********************************************
00008  0000              ;
00009  0000              ;                 BY  MATT BLAIS
00010  0000              ;
00011  0000              ;
00012  0000              ; ******************************
00013  0000              ; **     FILE DESCRIPTION     **
00014  0000              ; ******************************
00015  0000              ;
00016  0000              ;  'FLOAD64.S'
00017  0000              ;
00018  0000              ;  FLOAD64.S IS THE 6502-ASSEMBLER SOURCE CODE FOR THE
00019  0000              ;  C-64 END OF THE FAST-LOADER.  IT IS CURRENTLY ASSEM-
00020  0000              ;  BLED AT $C000, AND USES ZERO-PAGE MEMORY STARTING AT $0002.
00021  0000              ;  IT CAN BE RE-ASSEMBLED TO RUN ANYWHERE IN MEMORY.
00022  0000              ;
00023  0000              ;
00024  0000              ;
00025  0000              ;  'FLOAD1541.S'
00026  0000              ;
00027  0000              ;  FLOAD1541.S IS THE 6502-ASSEMBLER SOURCE CODE FOR THE
00028  0000              ;  1541 END OF THE FAST-LOADER.  IT IS AND MUST STAY AT
00029  0000              ;  $0500 IN THE DRIVE.  THE ASSEMBLED CODE GETS STUCK
00030  0000              ;  ONTO THE END OF THE C-64 PROGRAM, AND WHEN THE C-64
00031  0000              ;  PROGRAM RUNS, IT TRANSFERS THE DRIVE CODE INTO THE
00032  0000              ;  DRIVE.
00033  0000              ;
00034  0000              ;
00035  0000              ;
00036  0000              ;  'FL64C.OBJ',  'FL15.OBJ'
00037  0000              ;
00038  0000              ;  THESE ARE ASSEMBLER OUTPUT FILES AND ARE ONLY USED
00039  0000              ;  TO CREATE THE BINARY IMAGES OF THE PROGRAMS.
00040  0000              ;
00041  0000              ;
00042  0000              ;
00043  0000              ;  'FL64.BIN'
00044  0000              ;
00045  0000              ;  THIS IS THE ASSEMBLED C-64 CODE FOR THE FAST-LOADER,
00046  0000              ;  CURRENTLY SITTING AT $C000.
00047  0000              ;
00048  0000              ;
00049  0000              ;
00050  0000              ;  'FL15.BIN'
00051  0000              ;
00052  0000              ;  THIS IS THE ASSEMBLED 1541 CODE, AT $0500.  TO COMPLETE
00053  0000              ;  THE FAST LOADER, THIS CODE IS APPENDED ONTO THE END OF
00054  0000              ;  THE C-64 CODE.
00055  0000              ;
```

```
INS*      PAGE 0001

LINE# LOC   CODE        LINE

00001 0000              ;SCDO,"INSTRUCTIONS"
00002 0000              ;2/15/85
00003 0000              ;
00004 0000              ; ***********************************************
00005 0000              ; **       INSTRUCTIONS FOR INSTALLING         **
00006 0000              ; **          1541-FASTLOAD ROUTINE            **
00007 0000              ; ***********************************************
00008 0000              ;
00009 0000              ;                BY  MATT BLAIS
00010 0000              ;
00011 0000              ;
00012 0000              ; ******************************
00013 0000              ; **     FILE DESCRIPTION     **
00014 0000              ; ******************************
00015 0000              ;
00016 0000              ;  'FLOAD64.S'
00017 0000              ;
00018 0000              ;  FLOAD64.S IS THE 6502-ASSEMBLER SOURCE CODE FOR THE
00019 0000              ;  C-64 END OF THE FAST-LOADER.  IT IS CURRENTLY ASSEM-
00020 0000              ;  BLED AT $C000, AND USES ZERO-PAGE MEMORY STARTING AT $0002
00021 0000              ;  IT CAN BE RE-ASSEMBLED TO RUN ANYWHERE IN MEMORY.
00022 0000              ;
00023 0000              ;
00024 0000              ;
00025 0000              ;  'FLOAD1541.S'
00026 0000              ;
00027 0000              ;  FLOAD1541.S IS THE 6502-ASSEMBLER SOURCE CODE FOR THE
00028 0000              ;  1541 END OF THE FAST-LOADER.  IT IS AND MUST STAY AT
00029 0000              ;  $0500 IN THE DRIVE.  THE ASSEMBLED CODE GETS STUCK
00030 0000              ;  ONTO THE END OF THE C-64 PROGRAM, AND WHEN THE C-64
00031 0000              ;  PROGRAM RUNS, IT TRANSFERS THE DRIVE CODE INTO THE
00032 0000              ;  DRIVE.
00033 0000              ;
00034 0000              ;
00035 0000              ;
00036 0000              ;  'FL64C.OBJ',  'FL15.OBJ'
00037 0000              ;
00038 0000              ;  THESE ARE ASSEMBLER OUTPUT FILES AND ARE ONLY USED
00039 0000              ;  TO CREATE THE BINARY IMAGES OF THE PROGRAMS.
00040 0000              ;
00041 0000              ;
00042 0000              ;
00043 0000              ;  'FL64.BIN'
00044 0000              ;
00045 0000              ;  THIS IS THE ASSEMBLED C-64 CODE FOR THE FAST-LOADER,
00046 0000              ;  CURRENTLY SITTING AT $C000
00047 0000              ;
00048 0000              ;
00049 0000              ;
00050 0000              ;  'FL15.BIN'
00051 0000              ;
00052 0000              ;  THIS IS THE ASSEMBLED 1541 CODE, AT $0500.  TO COMPLETE
00053 0000              ;  THE FAST LOADER, THIS CODE IS APPENDED ONTO THE END OF
00054 0000              ;  THE C-64 CODE.
00055 0000              ;
```

```
LINE# LOC  CODE        LINE

00056 0000             ;
00057 0000             ;
00058 0000             ;
00059 0000             ;  'FL.C000'
00060 0000             ;
00061 0000             ;  THIS FILE IS THE C-64 CODE WITH THE ASSEMBLED 1541 CODE
00062 0000             ;  ALREADY APPENDED TO IT.  THIS IS A READY-TO-RUN BINARY
00063 0000             ;  FILE AT $C000.
00064 0000             ;
00065 0000             ;
00066 0000             ;
00067 0000             ;
00068 0000             ; **********************************
00069 0000             ; **     USING THE FAST-LOADER     **
00070 0000             ; **********************************
00071 0000             ;
00072 0000             ;
00073 0000             ;  THE FAST-LOADER CONSISTS OF TWO SEPARATE PROGRAMS:
00074 0000             ;  ONE PROGRAM IN THE C-64 CONTROLS THE LOADING PROCESS, AND
00075 0000             ;  ANOTHER PROGRAM IN THE 1541 SENDS OVER THE FILE THAT
00076 0000             ;  IS BEING LOADED.
00077 0000             ;
00078 0000             ;  THE 1541 CODE SITS IMMEDIATELY AT THE END OF THE C-64
00079 0000             ;  CODE, AND WHEN THE PROGRAM IS RUN, IT TRANSFERS THE 1541
00080 0000             ;  CODE INTO THE DISK DRIVE AND EXECUTES IT.
00081 0000             ;  TO USE THE FAST LOADER:
00082 0000             ;  FIRST LOAD THE FILE 'FL.C000' WHICH IS BOTH THE C-64 CODE
00083 0000             ;  AND THE DRIVE CODE.
00084 0000             ;  YOU MUST KNOW THE STARTING TRACK AND SECTOR OF THE
00085 0000             ;  FILE YOU WISH TO FAST-LOAD.  THE STARTING TRACK # GOES INTO
00086 0000             ;  THE .X REGISTER, AND THE SECTOR # INTO THE .Y REGISTER;
00087 0000             ;  IF YOU DON'T WANT TO USE THE DEFAULT TRACK AND SECTOR
00088 0000             ;  WHICH ARE SET UP IN THE FIRST TWO LINES OF THE PROGRAM,
00089 0000             ;  THEN ENTER THE CODE AT $C004 WITH THE .X AND .Y REGISTERS
00090 0000             ;  CONTAINING THE STARTING TRACK AND SECTOR.  ENTERING THE
00091 0000             ;  CODE AT $C004 WILL SET THE DRIVE NUMBER TO 8; IF YOU WANT
00092 0000             ;  TO USE A DIFFERENT DEVICE, LOAD THE ACCUMULATOR WITH THE
00093 0000             ;  DEVICE NUMBER (AND .X AND .Y WITH THE STARTING TRACK &
00094 0000             ;  SECTOR) AND ENTER THE CODE AT $C006.  IN ALL CASES, THE
00095 0000             ;  ENTRY SHOULD BE VIA A JSR INSTRUCTION.
00096 0000             ;
00097 0000             ;  THE FAST-LOADED FILE WILL ALWAYS LOAD
00098 0000             ;  TO THE ADDRESS SPECIFIED BY IT'S FIRST TWO
00099 0000             ;  BYTES WHEN YOU USE THIS FAST-LOAD PROGRAM!
00100 0000             ;
00101 0000             ;  ERROR-TRAPPING:
00102 0000             ;  WHEN THE FAST-LOAD IS FINISHED, IT WILL RETURN
00103 0000             ;  CONTROL TO THE CALLING PROGRAM.  IF THERE WERE
00104 0000             ;  NO ERRORS DURING THE LOADING PROCESS, THE CARRY
00105 0000             ;  FLAG WILL BE CLEAR AND THE ACCUMULATOR WILL BE
00106 0000             ;  ZERO.  IF THE CARRY IS SET (AND .A IS NON-ZERO),
00107 0000             ;  THEN AN ERROR OCCURRED.  THIS COULD BE EITHER A
00108 0000             ;  KERNAL ERROR (SUCH AS 'DEVICE NOT PRESENT'), OR
00109 0000             ;  A DISK READ ERROR.  IF A KERNAL ERROR OCCURRED,
00110 0000             ;  BIT-7 WILL BE SET IN THE ACCUMULATOR.
```

```
INS*      PAGE 0003

LINE# LOC   CODE        LINE

00111  0000             ;  USE THE KERNAL 'READST' ROUTINE TO DETERMINE
00112  0000             ;  WHAT ERROR OCCERRED   IF THE
00113  0000             ;  ERROR OCCURRED WHILE READING A DISK SECTOR, BIT-7
00114  0000             ;  OF THE ACCUMULATOR WILL BE CLEAR, AND THE VALUE
00115  0000             ;  CONTAINED IN .A WILL BE THE DISK JOB-QUEUE
00116  0000             ;  ERROR CODE (SEE TABLE).  THE TRACK AND SECTOR
00117  0000             ;  WHERE THE ERROR WAS DETECTED ARE IN THE VARIABLES
00118  0000             ;  'TRACK' AND 'SECTOR', WHICH ARE AT $0002 AND
00119  0000             ;  $0003 IN THE CURRENT VERSION OF THE FAST-LOADER
00120  0000             ;  THE DRIVE WILL NOT 'KNOCK' IF AN ERROR IS DETECTED
00121  0000             ;  DURING THE LOAD
00122  0000             ;
00123  0000             ;
00124  0000             ;  ****************************************
00125  0000             ;  **        TABLE OF ERROR CODES        **
00126  0000             ;  ****************************************
00127  0000             ;
00128  0000             ;  CODE    DOS ERROR #    MEANING
00129  0000             ;  ----    -----------    -------
00130  0000             ;    0          0        ( OK - NO ERROR )
00131  0000             ;    2         20        NO HEADER-BLOCK
00132  0000             ;    3         21        NO SYNC
00133  0000             ;    4         22        NO DATA-BLOCK
00134  0000             ;    5         23        DATA-BLOCK CHECKSUM ERROR
00135  0000             ;    9         27        HEADER-BLOCK CHECKSUM ERROR
00136  0000             ;   11         29        DISK ID MISMATCH
00137  0000             ;
00138  0000             ;
00139  0000             ; * END *
00139  0000             ;

ERRORS = 00000
END OF ASSEMBLY
```

```
FLOAD1541.S  . . . .PAGE 0001

LINE# LOC    CODE        LINE

00001  0000              ;SCDO:"FLOAD1541.S"
00002  0000              ;2/14/85
00003  0000              ;
00004  0000              ; ************************************************
00005  0000              ; ***  1541-END OF FAST-LOADER:  BY MATT BLAIS  *** V.2
00006  0000              ; ***          ( WITH ERROR-CHECKING )          ***
00007  0000              ; ************************************************
00008  0000              ;                                    SJO : AF
00009  0000              ;
00010  0000              ;  <<<  IN THE DISK DRIVE  >>>
00011  0000              ;
00012  0000              PB      = $1800          ; SERIAL I/O PORT.
00013  0000              JOBS    = $00            ; JOB QUEUE.
00014  0000              HDRS    = $06            ; T/S FOR JOB QUEUE.
00015  0000              ;
00016  0000                      *=$B5
00017  00B5              WAITRK  *=*+1            ; TRACK (,SEC) WAITING IN BUF2.
00018  00B6              WAISEC  *=*+1
00019  00B7              TRACK   *=*+1
00020  00B8              SECTOR  *=*+1
00021  00B9              DATA    *=*+1
00022  00BA              INDEX   *=*+1
00023  00BB              JOBST   *=*+1            ; STATUS OF READ JOB.
00024  00BC              TRIES   *=*+1
00025  00BD              ;
00026  00BD              ;
00027  00BD              ;*** I/O-PORT-BIT VALUES ***
00028  00BD              DATIN   = $01
00029  00BD              DATOUT  = $02
00030  00BD              CLKIN   = $04
00031  00BD              CLKOUT  = $08
00032  00BD              ATA     = $10            ; AUTO-ATN ENABLE (OUT).
00033  00BD              ATNIN   = $80
00034  00BD              ;
00035  00BD              RDCMD   = 1              ; READ-SECTOR COMMAND.
00036  00BD              OFFCMD  = 0              ; 'END' COMMAND.
00037  00BD              READ    = $80            ; JOB-QUEUE READ COMMAND.
00038  00BD              NMTRYS  = 5
00039  00BD              ;
00040  00BD              BUF1    = $0300
00041  00BD              BUF2    = $0400
00042  00BD              BUF3    = $0500
00043  00BD              ;
00044  00BD              ;
00045  00BD              ;
00046  00BD              ;       *** DISK CODE (INIT) AT $0500 ***
00047  00BD                      *=BUF3
00048  0500              ;
00049  0500              ;
00050  0500  A9 00       INTO    LDA #0
00051  0502  85 B5               STA WAITRK       ; NO SECTOR WAITING.
00052  0504  A9 10               LDA #ATA         ; CLEAR LINES.
00053  0506  8D 00 18            STA PB
00054  0509              ;
00055  0509              ;
```

```
FLOAD1541.S  . . . . PAGE 0002

LINE# LOC    CODE        LINE


00056  0509              ;     ***** GET COMMAND *****
00057  0509              ;
00058  0509  20 91 05    CMDWT  JSR GETBYT
00059  050C  58                 CLI
00060  050D  C9 01              CMP #RDCMD
00061  050F  F0 01              BEQ RDSC10       ; READ-SECTOR CMD?
00062  0511  60                 RTS
00063  0512              ;
00064  0512              ;
00065  0512  20 91 05    RDSC10 JSR GETBYT       ; *** READ-CMD: GET T,S ***
00066  0515  85 B7              STA TRACK
00067  0517  20 91 05           JSR GETBYT
00068  051A  85 B8              STA SECTOR
00069  051C              ;
00070  051C  C5 B6              CMP WAISEC       ; IS IT WAITING IN BUF2?
00071  051E  D0 4E              BNE RDBLK        ; (NO) GO READ IT IN
00072  0520  A5 B7              LDA TRACK
00073  0522  C5 B5              CMP WAITRK
00074  0524  D0 48              BNE RDBLK
00075  0526              ;
00076  0526  58          RDSC15 CLI
00077  0527  A6 01              LDX JOBS+1       ; WAIT FOR BUF-2 READ TO COMPLETE.
00078  0529  30 FB              BMI RDSC15
00079  052B  86 BB              STX JOBST        ; SAVE STATUS OF READ JOB.
00080  052D              ;
00081  052D  CA                 DEX              ; SUCCESSFUL READ?
00082  052E  F0 04              BEQ RDSC18
00083  0530  C6 BC              DEC TRIES        ; TRY AGAIN..
00084  0532  D0 46              BNE RDB010
00085  0534              ;
00086  0534  A2 00       RDSC18 LDX #0           ; TRANSFER BUF2 TO BUF1.
00087  0536  BD 00 04    RDSC20 LDA BUF2,X
00088  0539  9D 00 03           STA BUF1,X
00089  053C  E8                 INX
00090  053D  D0 F7              BNE RDSC20
00091  053F              ;
00092  053F  AD 00 03    RDSC30 LDA BUF1+0       ; DOUBLE-BUFFER: READ LINK SECTOR TO BUF2.
00093  0542  85 B5              STA WAITRK
00094  0544  F0 11              BEQ SNDBF1       ; IF TRACK=0, NO LINK.
00095  0546  85 08              STA HDRS+2       ; NEXT TRACK.
00096  0548              ;
00097  0548  AD 01 03           LDA BUF1+1
00098  054B  85 B6              STA WAISEC
00099  054D  85 09              STA HDRS+3       ; READ SECTOR TO BUF2.
00100  054F              ;
00101  054F  A9 05              LDA #NMTRYS      ; INIT TRY-CNTR.
00102  0551  85 BC              STA TRIES
00103  0553  A9 80              LDA #READ
00104  0555  85 01              STA JOBS+1
00105  0557              ;
00106  0557              ;                        ; *** SEND BUF1 ***
00107  0557  A2 00       SNDBF1 LDX #0
00108  0559  86 BA              STX INDEX
00109  055B  A5 BB              LDA JOBST
00110  055D  20 C4 05           JSR SNDBYT       ; SEND JOB STATUS FIRST.
```

```
LINE# LOC   CODE        LINE

00111  0560                ;
00112  0560  A6 BA         SNDO10 LDX INDEX
00113  0562  BD 00 03              LDA BUF1,X
00114  0565  20 C4 05              JSR SNDBYT      ; SEND A BYTE
00115  0568  E6 BA                 INC INDEX
00116  056A  D0 F4                 BNE SNDO10
00117  056C  F0 9B                 BEQ CMDWT       ; DONE: WAIT FOR NEXT COMMAND.
00118  056E                ;
00119  056E                ;
00120  056E  A5 B7         RDBLK  LDA TRACK       ; READ BLOCK TO BUF2, XFER TO B1.
00121  0570  85 08                 STA HDRS+2
00122  0572  A5 B8                 LDA SECTOR
00123  0574  85 09                 STA HDRS+3
00124  0576  A9 05                 LDA #NMTRYS     ; INIT TRY-COUNTER.
00125  0578  85 BC                 STA TRIES
00126  057A                ;
00127  057A  A9 80         RDBO10 LDA #READ       ; PUT READ CMD IN JOB QUEUE.
00128  057C  85 01                 STA JOBS+1
00129  057E                ;
00130  057E  D0 A6                 BNE RDSC15      ; WAIT TILL READ IS FIN, THEN XFER.
00131  0580                ;
00132  0580                ;
00133  0580                ;
00134  0580                ;       ********************************
00135  0580                ;       ***  SERIAL I/O ROUTINES  ***
00136  0580                ;       ***                       ***
00137  0580                ;       ********************************
00138  0580                ;
00139  0580                ;
00140  0580                ;*********************************
00141  0580                ;*  FAST I/O MISC. SUBROUTINES   *
00142  0580                ;*********************************
00143  0580                ;
00144  0580  AD 00 18      DEBNC  LDA PB
00145  0583  CD 00 18             CMP PB
00146  0586  D0 F8                BNE DEBNC
00147  0588  60                   RTS
00148  0589                ;
00149  0589                ;
00150  0589  20 80 05      CLKO   JSR DEBNC       ; WAIT FOR CLK-LO.
00151  058C  29 04                AND #CLKIN
00152  058E  D0 F9                BNE CLKO
00153  0590  60                   RTS
00154  0591                ;
00155  0591                ;
00156  0591                ;************************
00157  0591                ;*  GET A BYTE FROM 64  *
00158  0591                ;************************
00159  0591                ;
00160  0591  A9 80         GETBYT LDA #$80
00161  0593  85 B9                STA DATA
00162  0595                ;
00163  0595  20 80 05      GTBIT  JSR DEBNC       ; WAIT FOR CLK-HI.
00164  0598  A8                   TAY
00165  0599  29 04                AND #CLKIN
```

```
LINE# LOC   CODE        LINE

00166  059B  F0 F8            BEQ GTBIT
00167  059D  78               SEI
00168  059E                  ;
00169  059E  98               TYA               ; SET DATA-HI
00170  059F  09 02            ORA #DATOUT
00171  05A1  8D 00 18         STA PB
00172  05A4                  ;
00173  05A4  20 89 05         JSR CLKO          ; WAIT FOR CLOCK-LO
00174  05A7  AD 00 18         LDA PB
00175  05AA  29 FD            AND #$FF-DATOUT
00176  05AC  8D 00 18         STA PB            ; SET DATA-LO.
00177  05AF                  ;
00178  05AF  20 80 05    GTBOO JSR DEBNC         ; WAIT FOR CLOCK-HI.
00179  05B2  A8               TAY
00180  05B3  29 04            AND #CLKIN
00181  05B5  F0 F8            BEQ GTBOO
00182  05B7  20 89 05         JSR CLKO          ; WAIT FOR CLOCK-LO.
00183  05BA                  ;
00184  05BA  98               TYA
00185  05BB  4A               LSR A             ; READ IN A BIT.
00186  05BC  66 B9            ROR DATA
00187  05BE  90 D5            BCC GTBIT         ; (REPEAT FOR 8 BITS).
00188  05C0                  ;
00189  05C0  A5 B9            LDA DATA
00190  05C2  58               CLI
00191  05C3  60               RTS
00192  05C4                  ;
00193  05C4                  ;
00194  05C4                  ;************************
00195  05C4                  ;*  SEND A BYTE TO 64   *
00196  05C4                  ;************************
00197  05C4                  ;
00198  05C4  A2 08       SNDBYT LDX #8
00199  05C6  85 B9            STA DATA
00200  05C8                  ;
00201  05C8  AD 00 18    BITSND LDA PB            ; WAIT FOR CLOCK-HI.
00202  05CB  A8               TAY
00203  05CC  29 04            AND #CLKIN
00204  05CE  F0 F8            BEQ BITSND
00205  05D0  98               TYA
00206  05D1  09 02            ORA #DATOUT       ; SET DATA-HI.
00207  05D3  8D 00 18         STA PB
00208  05D6  78               SEI
00209  05D7                  ;
00210  05D7  AD 00 18    BSNDO  LDA PB            ; WAIT FOR CLOCK-LO.
00211  05DA  A8               TAY
00212  05DB  29 04            AND #CLKIN
00213  05DD  D0 F8            BNE BSNDO
00214  05DF                  ;
00215  05DF  98          BSND1  TYA               ; SET DATA-LO.
00216  05E0  29 FD            AND #$FF-DATOUT
00217  05E2  8D 00 18         STA PB
00218  05E5                  ;
00219  05E5  AD 00 18    BSND2A LDA PB            ; WAIT FOR CLOCK-HI.
00220  05E8  A8               TAY
```

```
LINE# LOC   CODE        LINE

00221  05E9  29 04            AND #CLKIN
00222  05EB  F0 F8            BEQ BSND2A
00223  05ED                 ;
00224  05ED  98              TYA             ; COMPUTE BIT.
00225  05EE  46 B9           LSR DATA
00226  05F0  90 02           BCC BSND2
00227  05F2  09 02           ORA #DATOUT
00228  05F4                 ;
00229  05F4  8D 00 18  BSND2 STA PB          ; SEND BIT.
00230  05F7                 ;
00231  05F7  AD 00 18  BSND3 LDA PB          ; WAIT FOR CLOCK-LOW.
00232  05FA  A8              TAY
00233  05FB  29 04           AND #CLKIN
00234  05FD  D0 F8           BNE BSND3
00235  05FF                 ;
00236  05FF  98              TYA             ; SET DATA-LOW.
00237  0600  29 FD           AND #$FF-DATOUT
00238  0602  8D 00 18        STA PB
00239  0605                 ;
00240  0605  CA              DEX             ; (REPEAT FOR 8 BITS).
00241  0606  D0 DD           BNE BSND2A
00242  0608  58              CLI
00243  0609  60              RTS
00244  060A                 ;
00245  060A                 ;
00246  060A                 .END

ERRORS = 00000

SYMBOL TABLE

SYMBOL VALUE
 ATA      0010    ATNIN    0080    BITSND   05C8    BSNDO    05D7
 BSND1    05DF    BSND2    05F4    BSND2A   05E5    BSND3    05F7
 BUF1     0300    BUF2     0400    BUF3     0500    CLKO     0589
 CLKIN    0004    CLKOUT   0008    CMDWT    0509    DATA     00B9
 DATIN    0001    DATOUT   0002    DEBNC    0580    GETBYT   0591
 GTBIT    0595    GTB00    05AF    HDRS     0006    INDEX    00BA
 INTO     0500    JOBS     0000    JOBST    00BB    NMTRYS   0005
 OFFCMD   0000    PB       1800    RDB010   057A    RDBLK    056E
 RDCMD    0001    RDSC10   0512    RDSC15   0526    RDSC18   0534
 RDSC20   0536    RDSC30   053F    READ     0080    SECTOR   00B8
 SND010   0560    SNDBF1   0557    SNDBYT   05C4    TRACK    00B7
 TRIES    00BC    WAISEC   00B6    WAITRK   00B5
END OF ASSEMBLY
```

```
FLOAD64.S       PAGE 0001

LINE# LOC    CODE        LINE

00001  0000              ;SCDO,"FLOAD64.S"
00002  0000              ;2/14/85
00003  0000              ;
00004  0000              ; ***********************************************
00005  0000              ; ***      C-64 END OF 1541 FAST LOADER       ***
00006  0000              ; ***         ( WITH ERROR-CHECKING )         *** V.2
00007  0000              ; ***********************************************
00008  0000              ;                                        SIO: AF
00009  0000              ;*** VARIABLES ***
00010  0000                    *=$02
00011  0002              ;
00012  0002              TRACK  *=*+1
00013  0003              SECTOR *=*+1
00014  0004              DATA   *=*+1
00015  0005              TOREAD *=*+1
00016  0006              PTR    *=*+2            ; (2 BYTES).
00017  0008              STAT   *=*+1
00018  0009              DEVC   *=*+1            ; DEVICE NUMBER.
00019  000A              ;
00020  000A              CI2PRA = $DD00
00021  000A              DATIN  = $80            ; PORT BITS.
00022  000A              CLKIN  = $40
00023  000A              DATOUT = $20
00024  000A              CLKOUT = $10
00025  000A              ;
00026  000A              ;
00027  000A              CHKIN  = $FFC6
00028  000A              CHKOUT = $FFC9
00029  000A              CHRIN  = $FFCF
00030  000A              CHROUT = $FFD2
00031  000A              CLRCHN = $FFCC
00032  000A              CLOSE  = $FFC3
00033  000A              SETNAM = $FFBD
00034  000A              SETLFS = $FFBA
00035  000A              OPEN   = $FFC0
00036  000A              ;
00037  000A              RDCMD  = 1              ; DISK COMMAND TO READ IN AND SEND A SECTOR
00038  000A              OFFCMD = 0              ; DISK COMMAND TO REVERT TO DOS CONTROL.
00039  000A              ;
00040  000A              ;
00041  000A              ;
00042  000A                    *=$C000; **** ENTRY TO INIT AND LOAD A FILE ****
00043  C000              ;
00044  C000  A2 17       FLDDEF LDX #23          ; DEFAULT TRACK & SECTOR.
00045  C002  A0 01              LDY #1
00046  C004  A9 08       FLD8   LDA #8           ; DEFAULT TO DRIVE #8.
00047  C006              ;
00048  C006  85 09       FLD    STA DEVC
00049  C008  20 32 C0           JSR FSTART       ; ** INIT DRIVE ** (NO PARAMS)
00050  C00B  B0 10              BCS CLSALL       ; ERROR ON OPEN.
00051  C00D              ;
00052  C00D  AD 02 DD           LDA CI2PRA+2     ; SAVE DDR VALUE.
00053  C010  48                 PHA
00054  C011  A9 30              LDA #DATOUT+CLKOUT ; NEW DDR.
00055  C013  8D 02 DD           STA CI2PRA+2
```

```
LINE# LOC   CODE        LINE

00056  C016  20 EC C0          JSR FLOAD       ; ** LOAD A FILE **  (X,Y = T,S)
00057  C019  68                 PLA
00058  C01A  8D 02 DD           STA CI2PRA+2    ; RESTORE DDR.
00059  C01D                     ;
00060  C01D  A9 02        CLSALL LDA #2          ; CLOSE FILES
00061  C01F  20 C3 FF           JSR CLOSE
00062  C022  A9 0F              LDA #15
00063  C024  20 C3 FF           JSR CLOSE       ; ***** EXIT *****
00064  C027                     ;
00065  C027  38                 SEC
00066  C028  A5 08              LDA STAT        ; STATUS (.A) SHOULD BE ZERO ON EXIT.
00067  C02A  C9 01              CMP #1
00068  C02C  D0 03              BNE EXIT
00069  C02E  18                 CLC             ; CARRY-CLEAR IF ALL OK.
00070  C02F  A9 00              LDA #0
00071  C031  60           EXIT  RTS
00072  C032                     ;
00073  C032                     ;
00074  C032                     ;
00075  C032                     ;*****************************************************************
00076  C032                     ;***  INIT :  READ DRIVE PROGRAM FROM SECTOR INTO DRIVE  ***
00077  C032                     ;*****************************************************************
00078  C032                     ;
00079  C032  86 02        FSTART STX TRACK       ;SAVE .X, .Y
00080  C034  84 03              STY SECTOR
00081  C036  A9 01              LDA #1
00082  C038  85 08              STA STAT        ; START WITH NO ERROR.
00083  C03A                     ;
00084  C03A  A9 0F              LDA #15
00085  C03C  20 C3 FF           JSR CLOSE
00086  C03F  A9 02              LDA #2
00087  C041  20 C3 FF           JSR CLOSE
00088  C044  A9 0F              LDA #15         ; OPEN 15,8,15,"I0"
00089  C046  A8                 TAY
00090  C047  A6 09              LDX DEVC
00091  C049  20 BA FF           JSR SETLFS
00092  C04C  A9 02              LDA #2
00093  C04E  A2 D9              LDX #<I0
00094  C050  A0 C0              LDY #>I0
00095  C052  20 BD FF           JSR SETNAM
00096  C055  20 C0 FF           JSR OPEN
00097  C058  90 06              BCC FINIT
00098  C05A                     ;
00099  C05A  09 80        ERRR  ORA #$80        ; I/O ERR: MAKE .A NON-ZERO.
00100  C05C  85 08        ERR1  STA STAT
00101  C05E  38                 SEC
00102  C05F  60                 RTS
00103  C060                     ;
00104  C060  A9 02        FINIT LDA #2          ;OPEN THE DATA CHANNEL 2,8,2,"#2".
00105  C062  A6 09              LDX DEVC
00106  C064  A8                 TAY
00107  C065  20 BA FF           JSR SETLFS
00108  C068  A9 02              LDA #2
00109  C06A  A2 DB              LDX #<DNAME1
00110  C06C  A0 C0              LDY #>DNAME1
```

```
LINE# LOC   CODE        LINE

00111  C06E  20 BD FF          JSR SETNAM
00112  C071  20 C0 FF          JSR OPEN
00113  C074  B0 E4             BCS ERRR
00114  C076                    ;
00115  C076  A2 0F             LDX #15          ; SEND 'B-P: 2 0'
00116  C078  20 C9 FF          JSR CHKOUT
00117  C07B  A0 00             LDY #0
00118  C07D  B9 DD C0    SBP01 LDA BFPNT,Y
00119  C080  F0 06             BEQ SBP05
00120  C082  20 D2 FF          JSR CHROUT
00121  C085  C8                INY
00122  C086  D0 F5             BNE SBP01
00123  C088  20 CC FF    SBP05 JSR CLRCHN
00124  C08B                    ;
00125  C08B  A2 02             LDX #2           ;SEND DRIVE CODE TO $0500.
00126  C08D  20 C9 FF          JSR CHKOUT       ; (256 BYTES)
00127  C090  B0 C8             BCS ERRR
00128  C092  A0 00             LDY #0
00129  C094  B9 0E C2    FNT0  LDA DRVCOD,Y
00130  C097  20 D2 FF          JSR CHROUT
00131  C09A  C8                INY
00132  C09B  D0 F7             BNE FNT0
00133  C09D  20 CC FF    FNT5  JSR CLRCHN       ;SEND EOI.
00134  C0A0                    ;
00135  C0A0  A2 0F             LDX #15
00136  C0A2  20 C9 FF          JSR CHKOUT       ; SEND REST OF CODE IN 'M-W' CMD.
00137  C0A5  A0 05             LDY #5
00138  C0A7  B9 E6 C0    MW0010 LDA MWR1,Y
00139  C0AA  20 D2 FF          JSR CHROUT
00140  C0AD  88                DEY
00141  C0AE  10 F7             BPL MW0010
00142  C0B0  A0 00             LDY #0           ; 31 MORE BYTES.
00143  C0B2  B9 0E C3    MW0020 LDA DRVCOD+256,Y
00144  C0B5  20 D2 FF          JSR CHROUT
00145  C0B8  C8                INY
00146  C0B9  CC E6 C0          CPY MWR1
00147  C0BC  D0 F4             BNE MW0020
00148  C0BE  20 CC FF          JSR CLRCHN
00149  C0C1                    ;
00150  C0C1  A2 0F             LDX #$0F         ;DRIVE: JMP TO $0500.
00151  C0C3  20 C9 FF          JSR CHKOUT
00152  C0C6  A9 55             LDA #'U
00153  C0C8  20 D2 FF          JSR CHROUT
00154  C0CB  A9 33             LDA #'3
00155  C0CD  20 D2 FF          JSR CHROUT
00156  C0D0  20 CC FF          JSR CLRCHN       ;DONE INIT.
00157  C0D3                    ;
00158  C0D3  A6 02             LDX TRACK
00159  C0D5  A4 03             LDY SECTOR       ;RECOVER .X, .Y
00160  C0D7  18                CLC
00161  C0D8  60                RTS
00162  C0D9                    ;
00163  C0D9                    ;
00164  C0D9  49 30       IO    .BYT 'IO'
00165  C0DB  23 32       DNAME1 .BYT '#2'
```

```
LINE# LOC   CODE        LINE

00166 C0DD  42 2D       BFPNT   .BYT 'B-P: 2 0',0
00166 C0E5  00
00167 C0E6  1F          MWR1    .BYT 31,6,0,'W-M'
00167 C0E7  06
00167 C0E8  00
00167 C0E9  57 2D 4D
00168 C0EC              ;
00169 C0EC              ;
00170 C0EC              ;
00171 C0EC              ;*************************************************************
00172 C0EC              ;**   LOAD A FILE:   GIVE STARTING TRACK=.X, SECTOR=.Y    **
00173 C0EC              ;*************************************************************
00174 C0EC              ; *
00175 C0EC              ; * DRIVE REVERTS TO NORMAL DOS CONTROL AFTER
00176 C0EC              ; *   FILE IS LOADED.
00177 C0EC              ; * DRIVE MUST BE INITIALIZED, READY FOR RDCMD.
00178 C0EC              ; *
00179 C0EC              ;
00180 C0EC  86 02       FLOAD   STX TRACK       ; SAVE FILE'S STARTING T,S.
00181 C0EE  84 03               STY SECTOR
00182 C0F0  AD 00 DD            LDA CI2PRA      ; CLEAR LINES.
00183 C0F3  29 CF               AND #$FF-CLKOUT-DATOUT
00184 C0F5  8D 00 DD            STA CI2PRA
00185 C0F8  2C 00 DD    FLO     BIT CI2PRA      ; WAIT FOR LINES TO CLEAR.
00186 C0FB  10 FB               BPL FLO
00187 C0FD  50 F9               BVC FLO
00188 C0FF              ;
00189 C0FF  A9 01               LDA #RDCMD      ; DISK COMMAND TO GET A SECTOR.
00190 C101  20 C0 C1            JSR SNDBYT
00191 C104  A5 02               LDA TRACK       ; GIVE FIRST T,S.
00192 C106  20 C0 C1            JSR SNDBYT
00193 C109  A5 03               LDA SECTOR
00194 C10B  20 C0 C1            JSR SNDBYT
00195 C10E              ;
00196 C10E  20 7F C1            JSR GETBYT      ; GET JOB STATUS FIRST.
00197 C111  85 08               STA STAT
00198 C113              ;
00199 C113  20 7F C1    FL1     JSR GETBYT
00200 C116  85 02               STA TRACK       ; GET T,S LINK (NEXT SECTOR).
00201 C118  20 7F C1            JSR GETBYT
00202 C11B  85 03               STA SECTOR
00203 C11D              ;
00204 C11D  20 7F C1            JSR GETBYT      ; GET FILE STARTING ADDRESS.
00205 C120  85 06               STA PTR
00206 C122  20 7F C1            JSR GETBYT
00207 C125  85 07               STA PTR+1
00208 C127              ;
00209 C127  A9 FC               LDA #252        ; 252 BYTES LEFT IN THIS BLOCK.
00210 C129  85 05               STA TOREAD
00211 C12B              ;
00212 C12B              ;                       ; GET A BYTE.
00213 C12B  20 7F C1    FL0010  JSR GETBYT
00214 C12E  A6 02               LDX TRACK       ; IS THIS LAST SECTOR?
00215 C130  D0 0A               BNE FL0015      ; (NO)
00216 C132  AA                  TAX
```

```
LINE# LOC   CODE        LINE

00217  C133  A5 03          LDA SECTOR      ; (YES) ARE WE PAST END OF FILE?
00218  C135  49 FF          EOR #$FF
00219  C137  C5 05          CMP TOREAD
00220  C139  B0 05          BCS FL0017      ; (YES) WAIT OUT REST OF SECTOR.
00221  C13B  8A              TXA
00222  C13C                ;
00223  C13C  A0 00       FL0015 LDY #0       ; (NO) STORE BYTE.
00224  C13E  91 06          STA (PTR),Y
00225  C140                ;
00226  C140  E6 06       FL0017 INC PTR      ; INCREMENT POINTER.
00227  C142  D0 02          BNE FL0020
00228  C144  E6 07          INC PTR+1
00229  C146                ;
00230  C146  C6 05       FL0020 DEC TOREAD   ; # BYTES LEFT IN THIS BLOCK.
00231  C148  D0 E1          BNE FL0010
00232  C14A                ;
00233  C14A                ;
00234  C14A  A5 08          LDA STAT        ; WAS READ OK?
00235  C14C  C9 01          CMP #1
00236  C14E  D0 04          BNE FLFIN       ; NO - DON'T FINISH LOAD.
00237  C150                ;
00238  C150  A5 02          LDA TRACK       ; WAS THIS LAST SECTOR OF FILE?
00239  C152  D0 06          BNE FL0030
00240  C154  A9 00       FLFIN  LDA #OFFCMD  ; (YES) SEND "OFF" COMMAND TO DRIVE.
00241  C156  20 C0 C1        JSR SNDBYT
00242  C159  60              RTS             ; EXIT THE LOAD ROUTINE ("STAT").
00243  C15A                ;
00244  C15A                ;
00245  C15A  A9 01       FL0030 LDA #RDCMD   ; (NO) READ ANOTHER SECTOR:
00246  C15C  20 C0 C1        JSR SNDBYT
00247  C15F  A5 02          LDA TRACK       ; SEND OUT NEXT TRACK, SECTOR.
00248  C161  20 C0 C1        JSR SNDBYT
00249  C164  A5 03          LDA SECTOR
00250  C166  20 C0 C1        JSR SNDBYT
00251  C169                ;
00252  C169  20 7F C1        JSR GETBYT      ; GET JOB STATUS.
00253  C16C  85 08           STA STAT
00254  C16E                ;
00255  C16E  20 7F C1        JSR GETBYT      ; GET NEW T,S LINK.
00256  C171  85 02           STA TRACK
00257  C173  20 7F C1        JSR GETBYT
00258  C176  85 03           STA SECTOR
00259  C178                ;
00260  C178  A9 FE          LDA #254        ; 254 BYTES LEFT IN BLOCK.
00261  C17A  85 05           STA TOREAD
00262  C17C  4C 2B C1        JMP FL0010
00263  C17F                ;
00264  C17F                ;
00265  C17F                ;
00266  C17F                ;       ****************************
00267  C17F                ;       ***   SERIAL ROUTINES    ***
00268  C17F                ;       ***    (ANDY FINKEL)     ***
00269  C17F                ;       ****************************
00270  C17F                ;
00271  C17F                ;
```

```
LINE# LOC   CODE        LINE

00272  C17F             ;
00273  C17F             ;*******************************
00274  C17F             ;*  GET BYTE FROM DISK DRIVE   *
00275  C17F             ;*******************************
00276  C17F             ;
00277  C17F  A9 80       GETBYT LDA #$80
00278  C181  85 04              STA DATA
00279  C183             ;
00280  C183  AD 00 DD    GETBIT LDA CI2PRA       ; SET CLOCK-HI.
00281  C186  09 10              ORA #CLKOUT
00282  C188  8D 00 DD           STA CI2PRA
00283  C18B             ;
00284  C18B  AD 00 DD    GETB00 LDA CI2PRA       ; WAIT FOR DATA-HI.
00285  C18E  A8                 TAY
00286  C18F  29 80              AND #DATIN
00287  C191  D0 F8              BNE GETB00
00288  C193             ;
00289  C193  98                 TYA             ; SET CLOCK-LO.
00290  C194  29 EF              AND #$FF-CLKOUT
00291  C196  8D 00 DD           STA CI2PRA
00292  C199             ;
00293  C199  AD 00 DD    GETB01 LDA CI2PRA       ; WAIT FOR DATA-LO.
00294  C19C  A8                 TAY
00295  C19D  29 80              AND #DATIN
00296  C19F  F0 F8              BEQ GETB01
00297  C1A1             ;
00298  C1A1  98          GETB02 TYA             ; SET CLK-HI.
00299  C1A2  09 10              ORA #CLKOUT
00300  C1A4  8D 00 DD           STA CI2PRA
00301  C1A7             ;
00302  C1A7  A2 07              LDX #7          ; ...WAIT FOR DRIVE TO SEND A BIT...
00303  C1A9  CA          GETB03 DEX
00304  C1AA  D0 FD              BNE GETB03
00305  C1AC             ;
00306  C1AC  AD 00 DD           LDA CI2PRA
00307  C1AF  A8                 TAY             ; GET DATA BIT.
00308  C1B0  29 EF              AND #$FF-CLKOUT
00309  C1B2  8D 00 DD           STA CI2PRA      ; SET CLK-LOW.
00310  C1B5  98                 TYA
00311  C1B6  0A                 ASL A
00312  C1B7  66 04              ROR DATA
00313  C1B9  90 E6              BCC GETB02      ; GET REST OF DATA BITS.
00314  C1BB             ;
00315  C1BB  A5 04              LDA DATA
00316  C1BD  49 FF              EOR #$FF
00317  C1BF  60                 RTS
00318  C1C0             ;
00319  C1C0             ;
00320  C1C0             ;*******************************
00321  C1C0             ;*  SEND BYTE TO DISK DRIVE    *
00322  C1C0             ;*******************************
00323  C1C0             ;
00324  C1C0  85 04       SNDBYT STA DATA
00325  C1C2  A2 08              LDX #8
00326  C1C4  20 FA C1    ABIT   JSR SCLK1       ; SET CLOCK-HI.
```

```
LINE# LOC   CODE        LINE

00327  C1C7                ;
00328  C1C7  20 F1 C1     WAITD  JSR SETTLE       ; WAIT FOR DATA-HI
00329  C1CA  29 80                AND #DATIN
00330  C1CC  D0 F9                BNE WAITD
00331  C1CE  20 03 C2             JSR SCLK0       ; SET CLOCK-LO
00332  C1D1                ;
00333  C1D1  20 F1 C1     SNDB01 JSR SETTLE       ; WAIT FOR DATA-LO
00334  C1D4  A8                   TAY
00335  C1D5  29 80                AND #DATIN
00336  C1D7  F0 F8                BEQ SNDB01
00337  C1D9                ;
00338  C1D9  98                   TYA
00339  C1DA  09 10                ORA #CLKOUT
00340  C1DC  46 04                LSR DATA
00341  C1DE  90 02                BCC SNDB02
00342  C1E0  09 20                ORA #DATOUT
00343  C1E2  8D 00 DD     SNDB02 STA CI2PRA       ; SEND A BIT WITH CLOCK-HI
00344  C1E5                ;
00345  C1E5  A0 07                LDY #7          ; . . . WAIT . . .
00346  C1E7  88           SNDB03 DEY
00347  C1E8  D0 FD                BNE SNDB03
00348  C1EA                ;
00349  C1EA  20 03 C2             JSR SCLK0       ; CLEAR CLOCK
00350  C1ED                ;
00351  C1ED  CA                   DEX
00352  C1EE  D0 D4                BNE ABIT        ; DO OTHER BITS IN BYTE
00353  C1F0  60                   RTS
00354  C1F1                ;
00355  C1F1                ;
00356  C1F1                ;**********************
00357  C1F1                ;*  DEBOUNCE IO PORT  *
00358  C1F1                ;**********************
00359  C1F1                ;
00360  C1F1  AD 00 DD     SETTLE LDA CI2PRA
00361  C1F4  CD 00 DD            CMP CI2PRA
00362  C1F7  D0 F8                BNE SETTLE
00363  C1F9  60                   RTS
00364  C1FA                ;
00365  C1FA                ;************************
00366  C1FA                ;*  SET CLOCK OUT HIGH  *
00367  C1FA                ;************************
00368  C1FA                ;
00369  C1FA  AD 00 DD     SCLK1  LDA CI2PRA
00370  C1FD  09 10                ORA #CLKOUT
00371  C1FF  29 DF                AND #$FF-DATOUT
00372  C201  D0 05                BNE PUTPRT
00373  C203                ;
00374  C203                ;***********************
00375  C203                ;*  SET CLOCK OUT LOW  *
00376  C203                ;***********************
00377  C203                ;
00378  C203  AD 00 DD     SCLK0  LDA CI2PRA
00379  C206  29 CF                AND #$FF-DATOUT-CLKOUT
00380  C208                ;
00381  C208  8D 00 DD     PUTPRT STA CI2PRA
```

```
FLOAD64.S  . . . PAGE 0008

LINE# LOC   CODE        LINE


00382  C20B  EA                 NOP
00383  C20C  EA                 NOP
00384  C20D  60                 RTS
00385  C20E                ;
00386  C20E                ;
00387  C20E                DRVCOD *=*+287          ; DISK DRIVE CODE AT END
00388  C32D                ;
00389  C32D                ;
00390  C32D                     .END

ERRORS = 00000


SYMBOL TABLE

SYMBOL VALUE
 ABIT     C1C4    BFPNT    CODD     CHKIN    FFC6    CHKOUT   FFC9
 CHRIN    FFCF    CHROUT   FFD2     CI2PRA   DD00    CLKIN    0040
 CLKOUT   0010    CLOSE    FFC3     CLRCHN   FFCC    CLSALL   C01D
 DATA     0004    DATIN    0080     DATOUT   0020    DEVC     0009
 DNAME1   C0DB    DRVCOD   C20E     ERR1     C05C    ERRR     C05A
 EXIT     C031    FINIT    C060     FL0      C0F8    FL1      C113
 FLD      C006    FLD8     C004     FLDDEF   C000    FLFIN    C154
 FL0010   C12B    FL0015   C13C     FL0017   C140    FL0020   C146
 FL0030   C15A    FLOAD    C0EC     FNT0     C094    FNT5     C09D
 FSTART   C032    GETBIT   C183     GETB00   C18B    GETB01   C199
 GETB02   C1A1    GETB03   C1A9     GETBYT   C17F    IO       C0D9
 MW0010   C0A7    MW0020   C0B2     MWR1     C0E6    OFFCMD   0000
 OPEN     FFC0    PTR      0006     PUTPRT   C208    RDCMD    0001
 SBP01    C07D    SBP05    C088     SCLK0    C203    SCLK1    C1FA
 SECTOR   0003    SETLFS   FFBA     SETNAM   FFBD    SETTLE   C1F1
 SNDB01   C1D1    SNDB02   C1E2     SNDB03   C1E7    SNDBYT   C1C0
 STAT     0008    TOREAD   0005     TRACK    0002    WAITD    C1C7

END OF ASSEMBLY
```

CHAPTER 7

64 Fast Load #2

Hardware: C64 with a 1541 or 1571

1) 3x speed up
2) simple user interface
3) Loads to any address between $0800 - $ffff

Normal LOAD time for 112 blocks	Fast Load #2 LOAD time for 112 blocks
1 min 17 sec	26 sec

Known bugs: None.

Fast load #2 includes 2 fast load routines. One that works with the screen on (fast) and one that works with the screen off (extra-fast). Some code gets downloaded to the drive. This allows us to work with the drive's job queue directly and to use our own serial handshake.

Once all the routines are in place, the fast routine is used to send the list of program files from the drive to the c64. The user then selects the file he wants to load.

The small section of code that does the actual loading is transferred to screen RAM and run. The screen will then blank as the extra-fast routine is used to fetch the user's file.

```
error addr  code          seq   source statement

                           1    ;=======================================================
                           2    ;these are the declarations for constants and variables that
                           3    ;will be used within the commodore 64 during program runs
                           4    ;=======================================================
                           5    ;
          =dd00            6    pb64   =$dd00     ;port containing serial i/o bits
          =0080            7    din64  =$80       ;data in line
          =0020            8    dout64 =$20       ;data out line
          =0040            9    cin64  =$40       ;clock in line
          =0010           10    cout64 =$10       ;clock out line
          =0008           11    atnout =$08       ;atn output (not used)
                          12    ;
                          13    ;kernal routines used in this program
                          14    ;
          =ffc6           15    chkin  =$ffc6     ;make file input
          =ffc9           16    ckout  =$ffc9     ;make file output
          =ffcc           17    clrchn =$ffcc     ;clear channels
          =ffe4           18    getin  =$ffe4     ;get a character from channel
          =ffd2           19    chrout =$ffd2     ;print a character to channel
          =ffba           20    setlfs =$ffba     ;set logical file number and device
          =ffbd           21    setnam =$ffbd     ;set file name for open
          =ffc0           22    open   =$ffc0     ;open a file
          =ffc3           23    close  =$ffc3     ;close a file
                          24    ;
                          25    ;zero page used by this program
                          26    ;
          =00fb           27    byte64 =$fb       ;incoming or outgoing byte during fast i/o
          =00fc           28    bcnt64 =$fc       ;counter for bytes to send or receive
          =00fd           29    lpb64  =$fd       ;copy of last state of the port for burst load
          =00fe           30    ptr    =$fe       ;pointer for data fetches or stores
          =006a           31    ptr2   =$6a       ;work pointer (in fac #2)
          =006c           32    vptr   =$6c       ;pointer to the screen
          =0062           33    cptr   =$62       ;pointer to the color (in fac #1)
          =0064           34    color  =$64       ;current color to use
          =0065           35    temp   =$65       ;temporary variable
          =002d           36    vartab =$2d       ;end of program pointer
                          37    ;
                          38    ;non zero page variables used by this program
                          39    ;
          =02a7           40    prgcnt =$02a7     ;number of programs on the disk
          =02a8           41    curprg =$02a8     ;current program that cursor is on
          =02a9           42    fprg   =$02a9     ;first program on the screen
          =02aa           43    curlin =$02aa     ; current line in menu that cursor is on
          =02ab           44    wrklin =$02ab     ;used during filename display
          =02ac           45    wrkprg =$02ac     ;used during filename display
          =02ad           46    dcntr  =$02ad     ;counter for bytes downloaded to disk
                          47    ;
                          48    ;=======================================================
                          49    ;these are the declarations for constants and variables that
                          50    ;will be used within the 1541 drive during program runs
                          51    ;=======================================================
                          52    ;
                          53    ;
          =1800           54    pb15   =$1800     ;port containing serial i/o bits
          =0001           55    din15  =$01       ;data in line
          =0002           56    dout15 =$02       ;data out line
          =0004           57    cin15  =$04       ;clock in line
```

```
error addr  code          seq   source statement

            =0008          58   cout15 =$08       ;clock out line
            =0080          59   atnin  =$80       ;atn input (not used)
                           60   ;
            =0000          61   jobs   =$00       ;job queue for the controller
            =0006          62   hdrs   =$06       ;arguments for each job (only first 4 are used)
                           63   ;
            =0080          64   read   =$80       ;these are the codes for each job to be performed
            =0090          65   write  =$90
            =00a0          66   wverfy =$a0
            =00b0          67   seek   =$b0
            =00b8          68   secsek =seek+$08
            =00c0          69   bump   =$c0
            =00d0          70   jumpc  =$d0
            =00e0          71   exec   =$e0
                           72   ;
            =0300          73   buf1   =$0300     ;addresses of each buffer in disk ram
            =0400          74   buf2   =$0400     ;first 2 buffers are used for data
            =0500          75   buf3   =$0500     ;this is where the static code lives
            =0600          76   buf4   =$0600     ;transient blocks of code are downloaded to here
            =0700          77   buf5   =$0700     ;this buffer is used for variable storage
                           78   ;
                           79   ;zero page storage used by the disk routines, uses hdrs associated
                           80   ;with buffers 3 through 5 because these are never used by the controller
                           81   ;
            =000a          82   bufptr =hdrs+$04  ;pointer to current active buffer
            =000c          83   joboff =hdrs+$06  ;index to current active job
            =000d          84   wrkoff =hdrs+$07  ;previous active job
            =000e          85   byte15 =hdrs+$08  ;current input/output byte
            =000f          86   bcnt15 =hdrs+$09  ;counter for multiple byte i/o
                           87   ;
                           88   ;non zero page storage used by the disk routines
                           89   ;
            =0780          90   *=buf5+$80        ;just use the upper half of buffer 5
                           91   ;
      0780 =0781           92   lpb15 *=*+1        ;copy of last port value during burst load
                           93   ;
                           94   ;=================================================================
                           95   ;this section of code starts the program by downloading the static
                           96   ;fast input output code to buffer 5 in the drive.  This fast i/o
                           97   ;routine is then used for all subsequent code transfers between the
                           98   ;drive and the 64.  Note, these i/o routines work with the screen
                           99   ;and interrupts turned on but aren't as fast as the burst transfers
                          100   ;=================================================================
                          101   ;
            =0801         102   *=$0801  (c64 BASIC start address)
                          103   ;
                          104   ;this data forms a basic line 10 that says sys2061
                          105   ;
      0801 0b 08 0a       106         .byte $0b,$08,$0a,0,$9e,'2061',0,0,0
      0804 00 9e 32
      0807 30 36 31
      080a 00 00 00
                          107   ;
      080d a9 0f          108   ad2061 lda #15      ;open a command channel to the drive
      080f a8             109         tay           ;but don't initialise, there's no need
      0810 a2 08          110         ldx #8        ;maybe this should be variable ++++++
      0812 20 ffba        111         jsr setlfs
```

```
error addr  code        seq   source statement

      0815 a9 00        112         lda #$00      ;no name to send
      0817 20 ffbd      113         jsr setnam
      081a 20 ffc0      114         jsr open
                        115   ;
      081d a9 a4        116         lda #<dcode   ;build a pointer to the disk code
      081f 85 fe        117         sta ptr
      0821 a9 08        118         lda #>dcode
      0823 85 ff        119         sta ptr+1
                        120   ;
      0825 a9 00        121         lda #<buf3    ;build address of destination into the memory...
      0827 8d 08a1      122         sta ddest     ;write command
      082a a9 05        123         lda #>buf3    ;this isn't really needed but it is safer in...
      082c 8d 08a2      124         sta ddest+1   ;case this code is called again
                        125   ;
      082f a9 d2        126         lda #<dclen   ;set up counter for length of disk code
      0831 8d 02ad      127         sta dcntr
      0834 a9 01        128         lda #>dclen
      0836 8d 02ae      129         sta dcntr+1
                        130   ;
                        131   ;this is the main loop to download code in 32 byte sections
                        132   ;
      0839 a2 0f        133   down00 ldx #15       ;make channel 15 output
      083b 20 ffc9      134         jsr ckout
      083e a2 00        135         ldx #$00      ;and now output the command string to the disk
                        136   ;
      0840 bd 089e      137   down10 lda memcmd,x
      0843 20 ffd2      138         jsr chrout
      0846 e8           139         inx
      0847 e0 06        140         cpx #6        ;have we sent all the bytes yet?
      0849 d0 f5        141         bne down10    ;nope, not yet
                        142   ;
      084b a0 00        143         ldy #$00      ;now output 32 bytes of the disk code
      084d b1 fe        144   down15 lda (ptr),y
      084f 20 ffd2      145         jsr chrout
      0852 c8           146         iny
      0853 c0 20        147         cpy #32
      0855 d0 f6        148         bne down15
                        149   ;
      0857 20 ffcc      150         jsr clrchn    ;let the disk execute this command
      085a 18           151         clc
      085b ad 08a1      152         lda ddest     ;move the destination pointer along by 32 bytes
      085e 69 20        153         adc #32
      0860 8d 08a1      154         sta ddest
      0863 90 03        155         bcc down16
      0865 ee 08a2      156         inc ddest+1
                        157   ;
      0868 18           158   down16 clc          ;move pointer to the source data on by 32 bytes
      0869 a5 fe        159         lda ptr
      086b 69 20        160         adc #32
      086d 85 fe        161         sta ptr
      086f 90 02        162         bcc down20    ;this can wrap over a page boundary
      0871 e6 ff        163         inc ptr+1
                        164   ;
      0873 ad 02ad      165   down20 lda dcntr     ;check if we have downloaded all data
      0876 38           166         sec
      0877 e9 20        167         sbc #32
      0879 8d 02ad      168         sta dcntr
```

```
error addr  code          seq   source statement

      087c b0 bb          169          bcs down00    ;still more data to send
      087e ad 02ae        170          lda dcntr+1   ;check the hi byte
      0881 f0 06          171          beq down25    ;all data has been sent
      0883 ce 02ae        172          dec dcntr+1
      0886 4c 0839        173          jmp down00
                          174    ;
                          175    ;
                          176    ;=============================================================
                          177    ;gets to here when all of the code has been downloaded, the routine
                          178    ;must now start the disk code running with a u3 command and give it
                          179    ;a little time to install itself before sending it commands.
                          180    ;=============================================================
                          181    ;
      0889 a2 0f          182    down25 ldx #15       ;make channel 15 output
      088b 20 ffc9        183           jsr ckout
      088e a9 55          184           lda #'u'      ;send a u3 command to execute code at $0500
      0890 20 ffd2        185           jsr chrout
      0893 a9 33          186           lda #'3'
      0895 20 ffd2        187           jsr chrout
      0898 20 ffcc        188           jsr clrchn    ;this will leave dout64 and cout64 = 0
      089b 4c 0b56        189           jmp menu      ;all initialised and ready to go so print the menu
                          190    ;
                          191    ;
      089e 4d 2d 57       192    memcmd .byte 'm-w'   ;command to do a memory write
      08a1 0500           193    ddest  .word buf3    ;variable, where to write to
      08a3 20             194           .byte 32      ;constant, number of bytes to be written
                          195    ;
                          196    ;
                          197    ;=============================================================
                          198    ;this is the section of code that gets downloaded to the drive on
                          199    ;power up of the program.  it contains fast transfer routines for
                          200    ;both 64 to 1541 and 1541 to 64.  these routines work with the
                          201    ;screen turned on and interrupts running and are used for sending
                          202    ;commands to the drive and returning errors to the 64.  There is
                          203    ;also a handshake that works with the 64's screen turned off for
                          204    ;doing super fast burst load of programs.
                          205    ;
                          206    ;the jsr commands use a computed destination address so that this
                          207    ;code can be assembled in line with the main 64 code but will have
                          208    ;the correct address when the code is in the disk buffers.
                          209    ;=============================================================
                          210    ;
      08a4 ad 1800        211    dcode  lda pb15              ;clear the clock and data lines to start
      08a7 29 f5          212           and #$ff-dout15-cout15
      08a9 8d 1800        213           sta pb15
                          214    ;
                          215    ;
                          216    ;=============================================================
                          217    ;this is the main loop that the drive code sits in.  it waits for
                          218    ;the 64 to send a command byte and calls the correct routine.
                          219    ;
                          220    ;command bytes are as follows:-
                          221    ;
                          222    ;0     turn off and return control to the dos
                          223    ;1     do a seek and return the status to the 64
                          224    ;2     execute the directory search and send routine
                          225    ;3     execute the burst load from a given start track and sector
```

```
error addr  code        seq   source statement

                        226   ;=======================================================
                        227   ;
      08ac 20 058b      228   cmdlp  jsr ibyt15-dcode+buf3    ;get a command from the 64
      08af 20 0511      229          jsr dojmp-dcode+buf3     ;call the correct routine
      08b2 4c 0508      230          jmp cmdlp-dcode+buf3     ;and get another command.
                        231   ;
      08b5 c9 01        232   dojmp  cmp #1           ;does the 64 want to do a seek on the disk ?
      08b7 f0 0b        233          beq doseek       ;yes
      08b9 c9 02        234          cmp #2           ;is this a directory command ?
      08bb f0 1a        235          beq dodir        ;yes
      08bd c9 03        236          cmp #3           ;is it a burst load command ?
      08bf f0 19        237          beq brstld       ;yes
      08c1 68           238          pla              ;no, any other value returns to dos
      08c2 68           239          pla              ;by scrapping return address to cmdlp
      08c3 60           240          rts              ;and returning to caller of this code
                        241   ;
                        242   ;
                        243   ;=======================================================
                        244   ;this code just does a seek on track 18 sector 0 and returns the
                        245   ;error code to the 64.  used to verify that a disk is in the drive
                        246   ;=======================================================
                        247   ;
      08c4 a9 12        248   doseek lda #18      ;set up the track and sector for the seek
      08c6 85 06        249          sta hdrs
      08c8 a9 00        250          lda #0
      08ca 85 07        251          sta hdrs+1
      08cc a9 b0        252          lda #seek
      08ce 85 00        253          sta jobs
                        254   ;
      08d0 a5 00        255   115    lda jobs             ;wait for the seek to finish
      08d2 30 fc        256          bmi 115              ;not done yet
      08d4 4c 0557      257          jmp obyt15-dcode+buf3   ;finish by sending status to the 64
                        258   ;
                        259   ;
                        260   ;=======================================================
                        261   ;call the directory read routine with a jump, (branch is too far)
                        262   ;=======================================================
                        263   ;
      08d7 4c 05e7      264   dodir  jmp bdir-dcode+buf3
                        265   ;
                        266   ;=======================================================
                        267   ;call the burst loading routine with a jump, (branch is too far)
                        268   ;=======================================================
                        269   ;
      08da 4c 065f      270   brstld jmp fload-dcode+buf3
                        271   ;
                        272   ;=======================================================
                        273   ;this is the frame handshake that starts transmission of 8 bits
                        274   ;in any direction.  It is called by the input and output routines to
                        275   ;put the 64 and 1541 code in synch for a faster handshake on bits
                        276   ;=======================================================
                        277   ;
      08dd ad 1800      278   fram15 lda pb15         ;wait for clkin = 1 (64 is ready for the byte)
      08e0 a8           279          tay              ;save the port value for later
      08e1 29 04        280          and #cin15
      08e3 f0 f8        281          beq fram15       ;64 isn't ready yet
      08e5 98           282          tya              ;acknowledge 64's ready signal with datout=1
```

```
error addr  code         seq   source statement

      08e6 09 02         283          ora #dout15      ;64 will see this as datin = 0
      08e8 8d 1800       284          sta pb15
      08eb 78            285          sei              ;disk can't have any irq's now
                         286   ;
      08ec ad 1800       287   fr0015 lda pb15         ;now wait for the 64 to acknowledge again
      08ef a8            288          tay              ;by resetting clkin to 0
      08f0 29 04         289          and #cin15
      08f2 d0 f8         290          bne fr0015       ;hasn't answered us yet
      08f4 98            291          tya              ;finalise frame handshake by setting datout=0
      08f5 29 fd         292          and #$ff-dout15  ;64 will see this as datin=1
      08f7 8d 1800       293          sta pb15
      08fa 60            294          rts              ;now go and do the bit handshake
                         295   ;
                         296   ;
                         297   ;=============================================================
                         298   ;this routine sends a byte of data to the 64 using a fast handshake
                         299   ;enter with .a=byte to send.
                         300   ;=============================================================
                         301   ;
      08fb 85 0e         302   obyt15 sta byte15       ;store the byte being sent
      08fd 8a            303          txa              ;save the .x register
      08fe 48            304          pha
      08ff 98            305          tya
      0900 48            306          pha                   ;save the .y register
      0901 a2 08         307          ldx #$08              ;keep a bit counter
      0903 20 0539       308          jsr fram15-dcode+buf3 ;go do a frame handshake
                         309   ;
                         310   ;the following handshake is performed for each bit that is sent to the 64
                         311   ;
      0906 ad 1800       312   ob1015 lda pb15         ;wait for clkin=1 (64 is ready for the bit)
      0909 a8            313          tay              ;save port value for later
      090a 29 04         314          and #cin15
      090c f0 f8         315          beq ob1015       ;64 isn't ready yet
                         316   ;
      090e 98            317          tya              ;64 expects the bit to be valid very soon!
      090f 46 0e         318          lsr byte15       ;datout is 0 at the moment so see what to send
      0911 b0 02         319          bcs ob2015       ;if a 1 is needed, send 0 to complement data
      0913 09 02         320          ora #dout15      ;a 0 is needed so send a 1 to complement the data
      0915 8d 1800       321   ob2015 sta pb15         ;present the 64 with it's data
                         322   ;
      0918 ad 1800       323   ob3015 lda pb15         ;wait for the 64 to say it has the data
      091b a8            324          tay              ;it will set clkin = 0 when it has
      091c 29 04         325          and #cin15
      091e d0 f8         326          bne ob3015       ;64 didn't pick it up yet
      0920 98            327          tya              ;set datout to a known state again (0)
      0921 29 fd         328          and #$ff-dout15
      0923 8d 1800       329          sta pb15
      0926 ca            330          dex              ;are there any more bits to send ?
      0927 d0 dd         331          bne ob1015       ;yes, so start the bit handshake again
      0929 58            332          cli              ;ok, all bits sent so allow irq's again
      092a 68            333          pla              ;restore .y register
      092b a8            334          tay
      092c 68            335          pla              ;restore .x register
      092d aa            336          tax
      092e 60            337          rts              ;bye bye
                         338   ;
                         339   ;
```

```
error addr  code        seq   source statement

                        340   ;=======================================================
                        341   ;this routine gets a byte of data from the 64 using a fast handshake
                        342   ;enter with .a=byte to send.
                        343   ;=======================================================
                        344   ;
     092f a9 01         345   ibyt15 lda #$01              ;put a flag bit into the data byte
     0931 85 0e         346          sta byte15            ;so we know when 8 bits have been sent
     0933 8a            347          txa                   ;save the .x register
     0934 48            348          pha
     0935 98            349          tya                   ;save the .y register
     0936 48            350          pha
     0937 20 0539       351          jsr fram15-dcode+buf3 ;go do a frame handshake
                        352   ;
                        353   ;the following handshake is performed for each bit that is sent by the 64
                        354   ;
     093a ad 1800       355   ib1015 lda pb15    ;wait for clkin=1 (64 has sent the data bit)
     093d a8            356          tay         ;save port value for later
     093e 29 04         357          and #cin15
     0940 f0 f8         358          beq ib1015  ;64 hasn't set it yet
     0942 98            359          tya         ;ok, get the bit back
     0943 4a            360          lsr a       ;move it to the carry
     0944 26 0e         361          rol byte15  ;and then into the data byte
                        362   ;
     0946 ad 1800       363   ib2015 lda pb15    ;wait for the 64 to set clkin to 0 again
     0949 29 04         364          and #cin15  ;note, any flag bit in the carry is preserved
     094b d0 f9         365          bne ib2015  ;not done it yet
     094d 90 eb         366          bcc ib1015  ;flag bit didn't drop off yet so get another bit
     094f 58            367          cli         ;ok, all bits fetched so allow irq's again
     0950 68            368          pla         ;restore .y
     0951 a8            369          tay
     0952 68            370          pla         ;restore .x
     0953 aa            371          tax
     0954 a5 0e         372          lda byte15  ;and return received byte in .a
     0956 60            373          rts
                        374   ;
                        375   ;
                        376   ;=======================================================
                        377   ;these are subroutines for starting reads on chained blocks.
                        378   ;call rfblok to read the first block into buf1 and call rnblok to
                        379   ;read the next block into the buffer that is not being used.  rnblok
                        380   ;takes its arguments from the current active buffer, if the first
                        381   ;byte is 0 then there is no block to chain to and nothing is done
                        382   ;=======================================================
                        383   ;
                        384   ;call rfblok with .x=track and .y=sector to be read
                        385   ;
     0957 8e 0400       386   rfblok stx buf2      ;store the first track we want to read
     095a 8c 0401       387          sty buf2+1    ;and the first sector
     095d a9 01         388          lda #1        ;fool rnblok into thinking buf2 is active
     095f 85 0c         389          sta joboff    ;drop through to the read next block routine
                        390   ;
                        391   ;this routine starts a block read into buf1 or buf2 depending on joboff
                        392   ;
     0961 a5 0c         393   rnblok lda joboff    ;joboff tells us where the track and sector are
     0963 85 0d         394          sta wrkoff    ;save current value as a work pointer
     0965 49 01         395          eor #1        ;make the other buffer the active one
     0967 85 0c         396          sta joboff    ;joboff now points to buffer where block will go
```

```
error addr  code        seq   source statement

      0969 18           397         clc           ;use old value of joboff as place to get t/s
      096a a5 0d        398         lda wrkoff
      096c 69 03        399         adc #>buf1    ;use the result to modify some code
      096e 8d 05db      400         sta gt-dcode+buf3+2
      0971 8d 05d6      401         sta gs-dcode+buf3+2
      0974 a5 0c        402         lda joboff    ;compute index to the correct header
      0976 0a           403         asl a
      0977 aa           404         tax           ;to set up track and sector for reading
      0978 ad 0301      405   gs    lda buf1+1    ;this gets modified - fetch the sector
      097b 95 07        406         sta hdrs+1,x
      097d ad 0300      407   gt    lda buf1      ;this gets modified too - fetch the track
      0980 95 06        408         sta hdrs,x
      0982 f0 06        409         beq rnbl00    ;track was 0, so nothing else to read
      0984 a6 0c        410         ldx joboff    ;now start a read into the correct buffer
      0986 a9 80        411         lda #read
      0988 95 00        412         sta jobs,x
      098a 60           413   rnbl00 rts          ;all done
                        414   ;
                        415   ;
                        416   ;=============================================
                        417   ;this routine reads the name of the disk and sends all
                        418   ;program names to the 64.  The send format is as follows:
                        419   ;
                        420   ;1     the error code when reading 18,0 - exit if <> 1
                        421   ;2     16 bytes that make up the name of the disk
                        422   ;
                        423   ;3     track and sector where file starts on the disk
                        424   ;      0 byte for track means directory read is complete, end.
                        425   ;4     send the name of the program (16 bytes)
                        426   ;5     go back to step 3
                        427   ;=============================================
                        428   ;
      098b a2 12        429   bdir  ldx #18             ;read 18,0 to find the disk name
      098d a0 00        430         ldy #0
      098f 20 05b3      431         jsr rfblok-dcode+buf3
                        432   ;
      0992 20 0648      433   bdir00 jsr snext-dcode+buf3  ;build pointer and start next read
      0995 48           434         pha                 ;save the controller return code
      0996 20 0557      435         jsr obyt15-dcode+buf3 ;send the code to the 64
      0999 68           436         pla                 ;get the code back
      099a c9 01        437         cmp #1              ;was our active block read ok ?
      099c f0 01        438         beq bdir05          ;yes so proceed to read the disk name
      099e 60           439         rts                 ;error, just return to main command loop
                        440   ;
      099f a0 90        441   bdir05 ldy #144           ;index to the disk name
      09a1 b1 0a        442   bdir10 lda (bufptr),y     ;get next character of the disk name
      09a3 20 0557      443         jsr obyt15-dcode+buf3 ;send it to the 64
      09a6 c8           444         iny
      09a7 c0 a0        445         cpy #160            ;have we sent the whole name ?
      09a9 d0 f6        446         bne bdir10          ;nope, not yet
                        447   ;
                        448   ;ok, the disk name has been sent so find all program types and send
                        449   ;the names and start t/s to the 64 for user selection
                        450   ;
      09ab 20 0648      451   bdir20 jsr snext-dcode+buf3  ;start the next block reading
      09ae c9 01        452         cmp #1              ;did the current block read ok ?
      09b0 f0 04        453         beq bdir25          ;yes
```

```
error addr  code        seq   source statement

      09b2 a9 00        454          lda #$00             ;error!, send 64 a zero byte and return
      09b4 f0 33        455          beq bdir80           ;bra
                        456   ;
      09b6 a9 02        457   bdir25 lda #$02         ;point to the first directory entry
      09b8 8d 0780      458          sta lpb15        ;keep the index
      09bb ac 0780      459   bdir30 ldy lpb15        ;get current index to the directory entries
      09be c8           460          iny              ;point to its track/sector field
      09bf b1 0a        461          lda (bufptr),y   ;if track = 0 then directory done
      09c1 f0 26        462          beq bdir80       ;yep, send a zero byte and exit
      09c3 88           463          dey              ;ok, a file is there so check if it's a prog
      09c4 b1 0a        464          lda (bufptr),y   ;get the type byte
      09c6 c9 82        465          cmp #$82         ;is it a closed program file ?
      09c8 d0 0e        466          bne bdir40       ;nope, go to the next entry
      09ca a9 12        467          lda #18          ;keep a byte counter
      09cc 85 0f        468          sta bcnt15
                        469   ;
      09ce c8           470   bdir35 iny              ;ok, send the next 18 bytes (t/s,name)
      09cf b1 0a        471          lda (bufptr),y
      09d1 20 0557      472          jsr obyt15-dcode+buf3
      09d4 c6 0f        473          dec bcnt15       ;have we sent it all ?
      09d6 d0 f6        474          bne bdir35       ;nope
                        475   ;
      09d8 ad 0780      476   bdir40 lda lpb15        ;point to the next directory entry
      09db 18           477          clc
      09dc 69 20        478          adc #32
      09de 8d 0780      479          sta lpb15
      09e1 90 d8        480          bcc bdir30       ;haven't finished yet
                        481   ;
      09e3 a0 00        482          ldy #$00         ;ok, finished this block, see if there's another
      09e5 b1 0a        483          lda (bufptr),y   ;if track link <> 0 then there is
      09e7 d0 c2        484          bne bdir20       ;start another read and send next buffer
                        485   ;
      09e9 4c 0557      486   bdir80 jmp obyt15-dcode+buf3 ;send a zero byte to finish
                        487   ;
                        488   ;
                        489   ;=================================================================
                        490   ;this routine waits for the job to finish on the buffer we are going
                        491   ;to read, and then starts another read on the other buffer (buf1 or 2)
                        492   ;=================================================================
                        493   ;
      09ec a9 00        494   snext  lda #$00              ;build pointer to the current buffer
      09ee 85 0a        495          sta bufptr
      09f0 a5 0c        496          lda joboff
      09f2 18           497          clc
      09f3 69 03        498          adc #>buf1
      09f5 85 0b        499          sta bufptr+1
      09f7 a6 0c        500          ldx joboff           ;wait for the current job to be finished
      09f9 b5 00        501   snex00 lda jobs,x
      09fb 30 fc        502          bmi snex00
      09fd 48           503          pha                  ;save the returned error code
      09fe 20 05bd      504          jsr rnblok-dcode+buf3 ;start a read on the next block
      0a01 68           505          pla                  ;get back the error code
      0a02 60           506          rts                  ;all done
                        507   ;
                        508   ;=================================================
                        509   ;this routine does the burst load of a program file, the 64
                        510   ;sends the start track and sector using the normal fast i/o
```

```
error addr  code        seq   source statement

                        511   ;routines and also receives the start address of the program
                        512   ;in the same manner
                        513   ;=======================================================
                        514   ;
      0a03 20 058b      515   fload  jsr ibyt15-dcode+buf3  ;get the start track
      0a06 aa           516          tax
      0a07 20 058b      517          jsr ibyt15-dcode+buf3  ;get the start sector
      0a0a a8           518          tay
      0a0b 20 05b3      519          jsr rfblok-dcode+buf3   ;read the first block into buf1
      0a0e 20 0648      520          jsr snext-dcode+buf3    ;and start the second block reading
      0a11 ad 0302      521          lda buf1+2              ;send the start address of the code
      0a14 20 0557      522          jsr obyt15-dcode+buf3
      0a17 ad 0303      523          lda buf1+3
      0a1a 20 0557      524          jsr obyt15-dcode+buf3
                        525   ;
      0a1d a0 04        526          ldy #4                  ;where to start the send from
      0a1f a2 fc        527          ldx #252                ;number of bytes being sent
      0a21 d0 0e        528          bne flo20               ;send the block starting at .y index
                        529   ;
                        530   ;this is the main loop that sends a block of code to the 64
                        531   ;
      0a23 a0 00        532   flo10  ldy #$00                ;assume its a full block
      0a25 a2 fe        533          ldx #254
      0a27 b1 0a        534          lda (bufptr),y          ;if track = 0 then it isnt
      0a29 d0 04        535          bne flo15               ;everything is ok
      0a2b c8           536          iny                     ;point to sector field for nmber of bytes to send
      0a2c b1 0a        537          lda (bufptr),y
      0a2e aa           538          tax
      0a2f a0 02        539   flo15  ldy #$02
      0a31 8a           540   flo20  txa                     ;tell 64 how many bytes
      0a32 20 06ab      541          jsr fstb15-dcode+buf3
                        542   ;
      0a35 b1 0a        543   flo30  lda (bufptr),y          ;now send all the bytes of data
      0a37 20 06ab      544          jsr fstb15-dcode+buf3
      0a3a c8           545          iny
      0a3b ca           546          dex
      0a3c d0 f7        547          bne flo30              ;more to send
                        548   ;
      0a3e a0 00        549          ldy #$00               ;see if there is another block to do
      0a40 b1 0a        550          lda (bufptr),y
      0a42 f0 07        551          beq flo50              ;nope, all done so finish by sending 0
      0a44 20 0648      552          jsr snext-dcode+buf3 ;start the next block reading
      0a47 c9 01        553          cmp #1                 ;did the new block read ok ?
      0a49 f0 d8        554          beq flo10              ;yes so get another block
                        555   ;
                        556   ;gets to here when the whole file has been sent to the 64 or an error
                        557   ;was encountered while reading program blocks
                        558   ;
      0a4b 68           559   flo50  pla              ;scrap return address to main command loop
      0a4c 68           560          pla
      0a4d a9 00        561          lda #$00         ;send 64 a zero byte to terminate
                        562   ;
                        563   ;
                        564   ;=======================================================
                        565   ;this is the code to do a fast handshake to the 64 with screen off
                        566   ;=======================================================
                        567   ;
```

```
error addr  code          seq   source statement

      0a4f 85 0e          568   fstb15 sta byte15       ;save the byte to be sent
      0a51 8a             569          txa              ;save .x
      0a52 48             570          pha
      0a53 a2 08          571          ldx #$08         ;keep a count of bits to be sent
      0a55 78             572          sei              ;and don't allow interrupts now
                          573   ;
      0a56 ad 1800        574   fs1500 lda pb15         ;get current value of the port
      0a59 06 0e          575          asl byte15       ;move the next data bit into the carry
      0a5b b0 05          576          bcs fs1510       ;nothing to do if bit is set (bus complements)
      0a5d 09 02          577          ora #dout15      ;send a 1 bit if 0 was required
      0a5f 8d 1800        578          sta pb15
      0a62 09 08          579   fs1510 ora #cout15      ;set datout = 1 to say data available
      0a64 8d 1800        580          sta pb15
      0a67 ea             581          nop              ;give the 64 time to find the data
      0a68 ea             582          nop              ;***** this runs very close to the edge....
      0a69 ea             583          nop              ;***** one more nop doesn't slow it down much
                          584   ;
      0a6a 29 f5          585          and #$ff-dout15-cout15   ;return port to known state
      0a6c 8d 1800        586          sta pb15
      0a6f ca             587          dex                      ;any more bits ?
      0a70 d0 e4          588          bne fs1500               ;yes
      0a72 58             589          cli                      ;irqs are ok now
      0a73 68             590          pla                      ;restore .x
      0a74 aa             591          tax
      0a75 60             592          rts                      ;all done
                          593   ;
                          594   ;
      0a76 00             595   dcend .byte 0 ;end of disk drive code
           =01d2          596   dclen = dcend-dcode
                          597   ;
                          598   ;
                          599   ;
                          600   ;put"0:c64code.src"
                          601   ;=======================================================
                          602   ;these are the fast i/o routines for the 64 that work with
                          603   ;the screen and interrupts turned on.
                          604   ;no computed offsets for the jsr addresses are required
                          605   ;because this code is assembled where it will be executed
                          606   ;=======================================================
                          607   ;
                          608   ;
                          609   ;=======================================================
                          610   ;this routine fetches a byte from the drive using a fast handshake
                          611   ;=======================================================
                          612   ;
      0a77 a9 80          613   ibyt64 lda #$80         ;put a flag bit into the data byte we will fetch
      0a79 85 fb          614          sta byte64
      0a7b 8a             615          txa              ;save .x
      0a7c 48             616          pha
      0a7d 98             617          tya              ;save .y
      0a7e 48             618          pha
      0a7f 20 0b31        619          jsr fram64       ;go do the byte handshake
                          620   ;
                          621   ;this section loops around a quick handshake to fetch 8 bits of data
                          622   ;
      0a82 ad dd00        623   ib1064 lda pb64         ;set clkout = 1 to say we want a bit now
      0a85 09 10          624          ora #cout64
```

```
error addr  code        seq   source statement

      0a87 8d dd00       625          sta pb64
                         626   ;
      0a8a a2 07         627          ldx #$07        ;give the disk time to present the data
      0a8c ca            628   ib2064 dex             ;with this small delay loop
      0a8d d0 fd         629          bne ib2064
                         630   ;
      0a8f ad dd00       631          lda pb64        ;ok disk! data should be valid by now
      0a92 29 ef         632          and #$ff-cout64 ;tell the disk we fetched it by setting clkout=0
      0a94 8d dd00       633          sta pb64
      0a97 0a            634          asl a           ;move the data bit into the data byte
      0a98 66 fb         635          ror byte64
      0a9a 90 e6         636          bcc ib1064      ;flag bit didn't drop out yet so get another bit
                         637   ;
      0a9c 68            638          pla         ;restore .y
      0a9d a8            639          tay
      0a9e 68            640          pla         ;restore .x
      0a9f aa            641          tax
      0aa0 a5 fb         642          lda byte64  ;fetch the assembled byte of data
      0aa2 60            643          rts ;bye bye
                         644   ;
                         645   ;
                         646   ;=============================================================
                         647   ;this routine sends a byte of data to the disk using the fast shake
                         648   ;the handshake is different to the get byte routine because the 64
                         649   ;can call the shots and depend on the 1541 to be waiting for data
                         650   ;at any given time.
                         651   ;=============================================================
                         652   ;
      0aa3 85 fb         653   obyt64 sta byte64   ;save the data byte to be sent
      0aa5 8a            654          txa ;save .x
      0aa6 48            655          pha
      0aa7 98            656          tya ;save .y
      0aa8 48            657          pha
      0aa9 a2 08         658          ldx #$08    ;keep a count of the bits to send
      0aab 20 0b31       659          jsr fram64  ;do the frame handshake for this byte
                         660   ;
                         661   ;this is the loop to handshake each bit over to the 1541
                         662   ;
      0aae ad dd00       663   ob1064 lda pb64    ;get the current value of the port
      0ab1 06 fb         664          asl byte64  ;move the next data bit into the carry
      0ab3 90 05         665          bcc ob2064  ;nothing to do, the bit is clear
      0ab5 09 20         666          ora #dout64 ;it's a 1 bit that needs to be sent
      0ab7 8d dd00       667          sta pb64    ;this is a fix (1526 drops bits if 2 are changed)
      0aba 09 10         668   ob2064 ora #cout64 ;set clkout to 1 to say data is there
      0abc 8d dd00       669          sta pb64
                         670   ;
      0abf ea            671          nop         ;give the drive time to find the bit
      0ac0 ea            672          nop
      0ac1 ea            673          nop
      0ac2 ea            674          nop
                         675   ;
      0ac3 29 cf         676          and #$ff-dout64-cout64
      0ac5 8d dd00       677          sta pb64    ;set the port back to a known state
      0ac8 ca            678          dex         ;are there any more bits to send
      0ac9 d0 e3         679          bne ob1064  ;yes
      0acb 68            680          pla         ;restore .y
      0acc a8            681          tay
```

```
error addr  code        seq   source statement

      0acd 68           682         pla         ;restore .x
      0ace aa           683         tax
      0acf 60           684         rts         ;bye bye
                        685   ;=================================================
                        686   ;this is the real fast zap load routine that fetches a program
                        687   ;from the drive with a very fast handshake.  The 64 has to be
                        688   ;watching the bus at all times, therefore it must have the
                        689   ;video chip turned off and interrupts disabled.
                        690   ;
                        691   ;this code is loaded into the screen at $0402 so that it cannot
                        692   ;be killed by programs loading over it (most cases)
                        693   ;=================================================
                        694   ;
      0ad0 ad 0400      695   sctop  lda $0400   ;build a pointer to the code
      0ad3 85 2d        696          sta vartab  ;use vartab so basic progs know where they finish
      0ad5 ad 0401      697          lda $0401
      0ad8 85 2e        698          sta vartab+1
                        699   ;
      0ada 20 044a      700   sct00  jsr fstb64-sctop+$0402    ;go get a super fast byte
      0add aa           701          tax         ;this is the number of bytes being sent
      0ade f0 18        702          beq gotprg  ;0 means we are all done
      0ae0 48           703          pha         ;save for later
      0ae1 a0 00        704          ldy #$00
                        705   ;
      0ae3 20 044a      706   sct10  jsr fstb64-sctop+$0402  ;get a byte of the code
      0ae6 91 2d        707          sta (vartab),y         ;store it in memory
      0ae8 c8           708          iny
      0ae9 ca           709          dex                    ;any more bytes in this block ?
      0aea d0 f7        710          bne sct10              ;yes
                        711   ;
      0aec 18           712          clc                    ;update the pointer
      0aed 68           713          pla
      0aee 65 2d        714          adc vartab
      0af0 85 2d        715          sta vartab
      0af2 90 e6        716          bcc sct00
      0af4 e6 2e        717          inc vartab+1
      0af6 d0 e2        718          bne sct00
                        719   ;
                        720   ;=================================================
                        721   ;gets to here when the code has been loaded to see what to do
                        722   ;=================================================
                        723   ;
      0af8 ad d011      724   gotprg lda 53265
      0afb 09 10        725          ora #16             ;turn the screen back on
      0afd 8d d011      726          sta 53265
      0b00 58           727          cli                 ;let interrupts fly now
      0b01 ad 0401      728          lda $0401           ;was this basic
      0b04 c9 08        729          cmp #8              ;if sa = $0801 then yes
      0b06 d0 0d        730          bne sysprg          ;hi byte not right
      0b08 ad 0400      731          lda $0400
      0b0b c9 01        732          cmp #1
      0b0d d0 06        733          bne sysprg
                        734   ;
      0b0f 20 a659      735          jsr $a659      ;runc, it was basic so run it
      0b12 4c a7ae      736          jmp $a7ae      ;newstt
                        737   ;
      0b15 6c 0400      738   sysprg jmp ($0400)
```

```
error addr  code          seq   source statement

                          739   ;
                          740   ;=======================================================
                          741   ;this routine does a fast byte handshake with the screen off
                          742   ;=======================================================
                          743   ;
      0b18 98             744   fstb64 tya            ;save .y
      0b19 48             745          pha
      0b1a a9 01          746          lda #$01       ;put a flag bit in the data byte
      0b1c 85 fb          747          sta byte64
                          748   ;
      0b1e ad dd00        749   fs6400 lda pb64       ;wait for clkin = 0 (disk sent data)
      0b21 a8             750          tay
      0b22 29 40          751          and #cin64
      0b24 d0 f8          752          bne fs6400     ;ok, bit isn't clear yet
                          753   ;
      0b26 98             754          tya
      0b27 0a             755          asl a          ;ok, we have the data bit
      0b28 26 fb          756          rol byte64     ;move it into the data byte
      0b2a 90 f2          757          bcc fs6400     ;more bits to fetch
                          758   ;
      0b2c 68             759          pla            ;restore .y
      0b2d a8             760          tay
      0b2e a5 fb          761          lda byte64     ;get back the data byte
      0b30 60             762          rts            ;all done
                          763   ;
                          764   ;
                          765   ;=======================================================
                          766   ;this is the frame handshake that starts transmission of 8 bits
                          767   ;in any direction.  it is called by the 64's fast input/output
                          768   ;routines to make sure the code in both the drive and the 64 is
                          769   ;in synch. it has been positioned here so that it is moved to
                          770   ;the screen ram along with the super fast i/o routine.
                          771   ;=======================================================
                          772   ;
      0b31 ad dd00        773   fram64 lda pb64      ;set clkout = 1 to start the handshake
      0b34 09 10          774          ora #cout64
      0b36 8d dd00        775          sta pb64
                          776   ;
      0b39 ad dd00        777   fr0064 lda pb64      ;wait for the drive to set datin = 0
      0b3c a8             778          tay           ;save the current port value for later
      0b3d 29 80          779          and #din64
      0b3f d0 f8          780          bne fr0064      ;drive didn't respond yet
                          781   ;
      0b41 98             782          tya             ;set clkout = 0 to set lines to 0 state
      0b42 29 ef          783          and #$ff-cout64 ;and let the drive know we are ready
      0b44 8d dd00        784          sta pb64
      0b47 ad dd00        785   fr1064 lda pb64        ;finally wait for the drive to set datin=1
      0b4a 29 80          786          and #din64
      0b4c f0 f9          787          beq fr1064      ;drive didn't respond yet
      0b4e 60             788          rts             ;ok, frame handshake is done
                          789   ;
                          790   ;
      0b4f 00             791   scend .byte 0
                          792   ;
                          793   ;=======================================================
                          794   ;this section displays the menu and gets the users selection
                          795   ;from that menu.  Sprites are downloaded to $3000
```

```
error addr  code          seq   source statement

                          796   ;=================================================
                          797   ;
      0b50 18 34 30       798   postbl .byte 24,52,48,52,72,52
      0b53 34 48 34
                          799   ;
      0b56 a9 00          800   menu   lda #0      ;screen background black
      0b58 8d d021        801          sta 53281
      0b5b a9 0d          802          lda #13     ;border light green
      0b5d 8d d020        803          sta 53280
      0b60 a9 93          804          lda #147
      0b62 20 ffd2        805          jsr chrout  ;clear the screen
                          806   ;
      0b65 a2 00          807          ldx #0      ;start at line 0
      0b67 a0 03          808          ldy #3      ;and do 3 reversed lines
      0b69 a9 0d          809          lda #13     ;in light green on the screen
      0b6b 20 0e2d        810          jsr doline
                          811   ;
      0b6e a2 04          812          ldx #4      ;now start at line 4
      0b70 a0 03          813          ldy #3      ;and do 3 reversed lines
      0b72 a9 01          814          lda #1      ;in white
      0b74 20 0e2d        815          jsr doline
                          816   ;
                          817   ;******* this code may be scrapped if program gets too big ******
                          818   ;
      0b77 a2 00          819          ldx #$00    ;index into sprite data
      0b79 bd 0f81        820   menu10 lda titlez,x ;download sprite data to $3000
      0b7c 9d 3000        821          sta $3000,x
      0b7f e8             822          inx
      0b80 e0 c0          823          cpx #192    ;have we downloaded all of the data ?
      0b82 d0 f5          824          bne menu10  ;nope, not yet
                          825   ;
      0b84 a2 03          826   menu20 ldx #$03    ;color all three sprites
      0b86 8a             827   menu25 txa         ;in white,red and cyan
      0b87 9d d026        828          sta $d026,x
      0b8a ca             829          dex
      0b8b d0 f9          830          bne menu25
                          831   ;
      0b8d 8e d025        832          stx $d025   ;mob multicolor 0 = black
      0b90 8e d017        833          stx $d017   ;no y expansion
      0b93 8e d01d        834          stx $d01d   ;no x expansion
                          835   ;
      0b96 a2 c0          836          ldx #192    ;build pointers to each sprite in memory
      0b98 8e 07f8        837          stx 2040
      0b9b e8             838          inx
      0b9c 8e 07f9        839          stx 2041
      0b9f e8             840          inx
      0ba0 8e 07fa        841          stx 2042
                          842   ;
      0ba3 a2 05          843          ldx #$05    ;now give each sprite its x,y position
      0ba5 bd 0b50        844   menu35 lda postbl,x
      0ba8 9d d000        845          sta $d000,x
      0bab ca             846          dex
      0bac 10 f7          847          bpl menu35
                          848   ;
      0bae a2 07          849          ldx #$07    ;turn on the 3 used sprites
      0bb0 8e d01c        850          stx $d01c   ;multicolor
      0bb3 8e d015        851          stx $d015   ;enable
```

```
error addr  code        seq   source statement

                        852   ;
                        853   ;******* this code may be scrapped if program gets too big ******
                        854   ;
      0bb6 a9 00        855         lda #0       ;put message 0 on the screen
      0bb8 20 0e47      856         jsr mesage   ;'1541 fast load by steve beats'
                        857   ;
                        858   ;=============================================================
                        859   ;menu screen is all set up now, so check that there is a disk in the
                        860   ;drive and ask the user for one if there isn't.
                        861   ;=============================================================
                        862   ;
      0bbb a9 01        863   menu40 lda #$01     ;tell the drive to do a seek
      0bbd 20 0aa3      864         jsr obyt64
      0bc0 20 0a77      865         jsr ibyt64   ;get the error return
      0bc3 c9 01        866         cmp #1       ;is there a disk in there
      0bc5 f0 08        867         beq menu45   ;yes
                        868   ;
      0bc7 a9 01        869         lda #1       ;nope, so ask the guy for a disk
      0bc9 20 0e47      870         jsr mesage
      0bcc 4c 0bbb      871         jmp menu40   ;and keep seeking until a disk is there
                        872   ;
                        873   ;
      0bcf a9 01        874   menu45 lda #$01     ;do one more seek because removing a disk...
      0bd1 20 0aa3      875         jsr obyt64   ;can make seek return a 1 when no disk is there
      0bd4 20 0a77      876         jsr ibyt64
      0bd7 c9 01        877         cmp #1
      0bd9 d0 e0        878         bne menu40   ;sorry, we shouldn't have got here yet
      0bdb a9 02        879         lda #2       ;put up the disk name message
      0bdd 20 0e47      880         jsr mesage
                        881   ;
      0be0 a9 02        882         lda #2       ;tell drive to do a directory send
      0be2 20 0aa3      883         jsr obyt64
      0be5 20 0a77      884         jsr ibyt64   ;get the status code for the 18,0 read
      0be8 c9 01        885         cmp #1       ;if 1 then ok to carry on and get the name
      0bea f0 08        886         beq menu50   ;no problems
                        887   ;
      0bec a9 00        888         lda #$00     ;****** error for now *******
      0bee 8d d015      889         sta $d015
      0bf1 4c 0aa3      890         jmp obyt64   ;turn off the drive code
                        891   ;
      0bf4 a2 05        892   menu50 ldx #5       ;build a pointer to the disk name field
      0bf6 20 0de1      893         jsr setptr
      0bf9 a0 16        894         ldy #22      ;where to store the stuff
      0bfb a2 10        895         ldx #16      ;we are going to get 16 bytes now
                        896   ;
      0bfd 20 0a77      897   menu55 jsr ibyt64   ;get a byte from the drive
      0c00 29 3f        898         and #63      ;convert character to reverse screen code
      0c02 09 80        899         ora #128
      0c04 91 6c        900         sta (vptr),y ;store the byte on the screen
      0c06 c8           901         iny
      0c07 ca           902         dex          ;have we fetched all bytes yet ?
      0c08 d0 f3        903         bne menu55   ;nope
                        904   ;
                        905   ;=============================================================
                        906   ;ok, we now have the disk name so fetch track,sector,name for each file
                        907   ;that the drive sends us.  A track number of zero means all files done
                        908   ;=============================================================
```

```
error addr  code        seq   source statement

                        909   ;
            0c0a 20 0dcc 910          jsr setfp    ;build a pointer to directory storage area
            0c0d a9 00   911          lda #0       ;keep a count of programs
            0c0f 8d 02a7 912          sta prgcnt
                        913   ;
            0c12 a0 00   914   menu60 ldy #$00     ;set index into storage
            0c14 20 0a77 915          jsr ibyt64   ;get the track number
            0c17 f0 1e   916          beq gotdir   ;0, so all is done
            0c19 91 fe   917          sta (ptr),y  ;remember the track number
            0c1b c8      918          iny
            0c1c 20 0a77 919          jsr ibyt64   ;fetch the sector number and store it
            0c1f 91 fe   920          sta (ptr),y
            0c21 c8      921          iny          ;now point to the name storage
            0c22 20 0a77 922   menu65 jsr ibyt64   ;fetch a byte of the filename
            0c25 29 3f   923          and #63      ;convert to display code
            0c27 91 fe   924          sta (ptr),y  ;and store in memory
            0c29 c8      925          iny
            0c2a c0 12   926          cpy #18
            0c2c d0 f4   927          bne menu65
            0c2e ee 02a7 928          inc prgcnt   ;update number of programs there
            0c31 20 0dd5 929          jsr nxtfil   ;move pointer on by 18 bytes
            0c34 4c 0c12 930          jmp menu60   ;bra
                        931   ;
                        932   ;=============================================================
                        933   ;all the directory entries are in memory now, so set up the display
                        934   ;and allow the user to cursor around to select a file for loading
                        935   ;=============================================================
                        936   ;
            0c37 ad 02a7 937   gotdir lda prgcnt  ;were there any files at all ?
            0c3a d0 24   938          bne gotd00   ;yes so let user choose
                        939   ;
            0c3c a2 08   940          ldx #8       ;error, no programs on the disk
            0c3e a0 05   941          ldy #5       ;display this in a 5 line reverse box
            0c40 a9 01   942          lda #1
            0c42 20 0e2d 943          jsr doline
                        944   ;
            0c45 a9 04   945          lda #4       ;use messages 4 and 5
            0c47 20 0e47 946          jsr mesage
            0c4a a9 05   947          lda #5
            0c4c 20 0e47 948          jsr mesage
                        949   ;
            0c4f 20 ffe4 950   noprog jsr getin    ;wait for stop or return
            0c52 c9 0d   951          cmp #13      ;was it return to change disks ?
            0c54 d0 03   952          bne nopr00   ;nope
            0c56 4c 0b56 953          jmp menu     ;redraw the menu and start again
                        954   ;
            0c59 c9 03   955   nopr00 cmp #3       ;was it stop
            0c5b d0 f2   956          bne noprog   ;nope
            0c5d 6c fffc 957          jmp ($fffc)  ;reset to clean up
                        958   ;
            0c60 a9 00   959   gotd00 lda #$00     ;set up to display files for the first time
            0c62 8d 02a9 960          sta fprg     ;first program on the screen
            0c65 8d 02a8 961          sta curprg   ;program number the cursor is over
            0c68 8d 02aa 962          sta curlin   ;current line where the cursor is
                        963   ;
            0c6b a2 15   964          ldx #21      ;put a 3 line deep reverse bar at screen bottom
            0c6d a0 03   965          ldy #3
```

```
error addr  code        seq   source statement

      0c6f a9 01        966         lda #1      ;make it white
      0c71 20 0e2d      967         jsr doline
      0c74 a9 03        968         lda #3      ;display the instructions on it
      0c76 20 0e47      969         jsr mesage
                        970   ;
      0c79 20 0d4d      971   dspm00 jsr dspfil  ;display as many files as possible
      0c7c 20 ffe4      972   dspm10 jsr getin   ;go get a key from the user
      0c7f f0 fb        973         beq dspm10
      0c81 c9 03        974         cmp #3      ;was it stop
      0c83 d0 03        975         bne dspm15  ;nope
      0c85 4c 0b56      976         jmp menu    ;start over if stop was pressed
                        977   ;
      0c88 c9 11        978   dspm15 cmp #17    ;was it cursor down ?
      0c8a d0 22        979         bne dspm20  ;nope
      0c8c ae 02a8      980         ldx curprg  ;first check if we would go past the end by moving
      0c8f e8           981         inx
      0c90 e8           982         inx
      0c91 ec 02a7      983         cpx prgcnt
      0c94 b0 e6        984         bcs dspm10  ;it was so don't do anything
      0c96 8e 02a8      985         stx curprg  ;still ok
      0c99 ae 02aa      986   dspm16 ldx curlin  ;move down a line
      0c9c e8           987         inx
      0c9d e0 0c        988         cpx #12     ;unless we were on the last one
      0c9f d0 07        989         bne dspm17  ;we aren't so just increment and carry on
      0ca1 ee 02a9      990         inc fprg    ;time to scroll so first prog = first+2
      0ca4 ee 02a9      991         inc fprg
      0ca7 ca           992         dex         ;correct .x because it went too far
      0ca8 8e 02aa      993   dspm17 stx curlin
      0cab 4c 0c79      994         jmp dspm00  ;redisplay the screen full of files
                        995   ;
      0cae c9 91        996   dspm20 cmp #145   ; was it cursor up ?
      0cb0 d0 20        997         bne dspm30  ;nope
      0cb2 ae 02a8      998         ldx curprg  ;only move up if prog # is 2 or more
      0cb5 e0 02        999         cpx #2
      0cb7 90 c3       1000         bcc dspm10  ;nothing to do we are on the top line
      0cb9 ca          1001         dex         ;move back by 2
      0cba ca          1002         dex
      0cbb 8e 02a8     1003         stx curprg
      0cbe ad 02aa     1004         lda curlin  ;move current line back unless we are on the top
      0cc1 d0 09       1005         bne dspm25  ;its ok, we are not on the top line
      0cc3 ce 02a9     1006         dec fprg    ;first = first -2
      0cc6 ce 02a9     1007         dec fprg
      0cc9 4c 0c79     1008         jmp dspm00  ;display the new files that moved in
      0ccc ce 02aa     1009   dspm25 dec curlin  ;ok to just move the cursor up the screen
      0ccf 4c 0c79     1010         jmp dspm00
                       1011   ;
      0cd2 c9 1d       1012   dspm30 cmp #29     ;was it cursor forward
      0cd4 d0 14       1013         bne dspm40  ;nope
      0cd6 ad 02a8     1014         lda curprg  ;move to the next if curprg<prgcnt and even
      0cd9 aa          1015         tax
      0cda e8          1016         inx
      0cdb ec 02a7     1017         cpx prgcnt
      0cde f0 9c       1018         beq dspm10  ;already at the maximum
      0ce0 29 01       1019         and #$01
      0ce2 d0 98       1020         bne dspm10  ;already in the second column
      0ce4 8e 02a8     1021         stx curprg  ;store the new position and redisplay
      0ce7 4c 0c79     1022         jmp dspm00
```

```
error addr  code        seq   source statement

                        1023   ;
      0cea c9 9d        1024   dspm40 cmp #157    ;was it cursor back ?
      0cec d0 0b        1025          bne dspm50  ;nope
      0cee ad 02a8      1026          lda curprg  ;make number even
      0cf1 29 fe        1027          and #$fe
      0cf3 8d 02a8      1028          sta curprg
      0cf6 4c 0c79      1029          jmp dspm00  ;redisplay the files
                        1030   ;
      0cf9 c9 0d        1031   dspm50 cmp #13     ;was it return to load ?
      0cfb f0 03        1032          beq dspm55  ;yes, so go load the program fast
      0cfd 4c 0c7c      1033          jmp dspm10  ;nope, so just get another key
                        1034   ;
                        1035   ;=====================================================
                        1036   ;gets to here when the user has chosen a program from the menu
                        1037   ;the main loading code gets put into screen ram and executed
                        1038   ;to load in the program.  depending on start address, the program
                        1039   ;is either run or called directly.
                        1040   ;=====================================================
                        1041   ;
      0d00 a9 93        1042   dspm55 lda #147         ;clear the screen
      0d02 20 ffd2      1043          jsr chrout
      0d05 a2 7f        1044          ldx #scend-sctop ;move code into the screen
      0d07 bd 0ad0      1045   dspm56 lda sctop,x
      0d0a 9d 0402      1046          sta $0402,x      ;leave room to store start address
      0d0d ca           1047          dex
      0d0e e0 ff        1048          cpx #$ff
      0d10 d0 f5        1049          bne dspm56
                        1050   ;
      0d12 78           1051          sei              ;no interrupts now please
      0d13 ad d011      1052          lda 53265        ;turn the screen off
      0d16 29 ef        1053          and #255-16
      0d18 8d d011      1054          sta 53265
      0d1b a9 00        1055          lda #$00         ;turn the sprites off
      0d1d 8d d015      1056          sta $d015
      0d20 8d 0801      1057          sta $0801        ;make this program disappear
      0d23 8d 0802      1058          sta $0802
      0d26 a9 03        1059          lda #3           ;tell the drive to go to the burst load
      0d28 20 0aa3      1060          jsr obyt64
      0d2b ad 02a8      1061          lda curprg       ;find out which program to load
      0d2e 20 0dbc      1062          jsr fndprg
      0d31 a0 00        1063          ldy #$00         ;tell burst load where to start from
      0d33 b1 fe        1064          lda (ptr),y      ;track
      0d35 20 0aa3      1065          jsr obyt64
      0d38 c8           1066          iny
      0d39 b1 fe        1067          lda (ptr),y      ;sector
      0d3b 20 0aa3      1068          jsr obyt64
      0d3e 20 0a77      1069          jsr ibyt64       ;get the start address of this program
      0d41 8d 0400      1070          sta $0400        ;save it for the fast loader routine
      0d44 20 0a77      1071          jsr ibyt64
      0d47 8d 0401      1072          sta $0401
      0d4a 4c 0402      1073          jmp $0402        ;call the zap part of zaploader
                        1074   ;
                        1075   ;=====================================================
                        1076   ;routine to display files that are in memory on the screen
                        1077   ;
                        1078   ;prgcnt    actual number of programs in memory
                        1079   ;curprg    program number that the cursor is currently on
```

```
error addr  code        seq   source statement

                        1080  ;curlin    line number that the cursor is on (0-11)
                        1081  ;fprg      first program being displayed on the screen
                        1082  ;=============================================
                        1083  ;
      0d4d a2 08        1084  dspfil ldx #8          ;point to the first line
      0d4f 20 0de1      1085         jsr setptr      ;where the menu will be displayed
      0d52 a9 00        1086         lda #$00        ;we are on line 0 now
      0d54 8d 02ab      1087         sta wrklin
      0d57 ad 02a9      1088         lda fprg        ;which program do we start at
      0d5a 8d 02ac      1089         sta wrkprg
      0d5d 20 0dbc      1090         jsr fndprg      ;build a pointer (in ptr) to the first program
                        1091  ;
      0d60 a2 00        1092  dspf00 ldx #$00        ;assume current prog is not the same as the...
      0d62 ad 02ac      1093         lda wrkprg      ;one the cursor is over
      0d65 cd 02a8      1094         cmp curprg
      0d68 d0 02        1095         bne dspf10
      0d6a a2 80        1096         ldx #$80        ;it is the same so display it reversed
      0d6c 86 65        1097  dspf10 stx temp        ;temp contains the reverse flag
      0d6e a0 02        1098         ldy #$02        ;point to the name and it's screen destination
      0d70 b1 fe        1099  dspf15 lda (ptr),y
      0d72 05 65        1100         ora temp
      0d74 91 6c        1101         sta (vptr),y
      0d76 a9 0f        1102         lda #15         ;display names in light grey
      0d78 91 62        1103         sta (cptr),y
      0d7a c8           1104         iny
      0d7b c0 12        1105         cpy #18
      0d7d d0 f1        1106         bne dspf15
                        1107  ;
      0d7f 18           1108         clc             ;move screen pointers to next column
      0d80 a5 6c        1109         lda vptr
      0d82 69 14        1110         adc #20
      0d84 85 6c        1111         sta vptr
      0d86 85 62        1112         sta cptr
      0d88 90 04        1113         bcc dspf20
      0d8a e6 6d        1114         inc vptr+1
      0d8c e6 63        1115         inc cptr+1
                        1116  ;
      0d8e 20 0dd5      1117  dspf20 jsr nxtfil       ;move filename pointer to the next entry
                        1118  ;
      0d91 ae 02ac      1119  dspf25 ldx wrkprg       ;have we finished ?
      0d94 e8           1120         inx
      0d95 ec 02a7      1121         cpx prgcnt
      0d98 f0 13        1122         beq dspf50      ;yes so see if we have to erase 2nd column
      0d9a 8e 02ac      1123         stx wrkprg
      0d9d 8a           1124         txa             ;if program number is even then move down a line
      0d9e 29 01        1125         and #$01
      0da0 d0 be        1126         bne dspf00      ;nope, we are on the same line
      0da2 ee 02ab      1127         inc wrklin      ;do we have room to display any more
      0da5 ad 02ab      1128         lda wrklin
      0da8 c9 0c        1129         cmp #12
      0daa d0 b4        1130         bne dspf00      ;yes keep displaying the programs
      0dac 60           1131  dspf30 rts             ;out of room, so finish
                        1132  ;
      0dad 8a           1133  dspf50 txa             ;if we finished in column 2 then whats there
      0dae 29 01        1134         and #$01        ;now will have to be erased (no file here now)
      0db0 f0 fa        1135         beq dspf30      ;nope, we finished in column 1
      0db2 a0 12        1136         ldy #18         ;ok, we have to blank out the last line
```

```
error addr  code         seq   source statement

      0db4 a9 20        1137          lda #32
      0db6 91 6c        1138    dspf55 sta (vptr),y
      0db8 88           1139          dey
      0db9 d0 fb        1140          bne dspf55
      0dbb 60           1141          rts
                        1142    ;
                        1143    ;=================================================
                        1144    ;this routine builds a pointer to the program entry number in .a
                        1145    ;the pointer is left in (ptr) for the caller to use as an indirect
                        1146    ;=================================================
                        1147    ;
      0dbc aa           1148    fndprg tax            ;keep a counter
      0dbd 20 0dcc      1149          jsr setfp       ;build pointer to first entry
      0dc0 e0 00        1150          cpx #0          ;is the pointer correct now ?
      0dc2 d0 01        1151          bne fndp00      ;nope
      0dc4 60           1152          rts             ;pointer done
                        1153    ;
      0dc5 20 0dd5      1154    fndp00 jsr nxtfil     ;add 18 to ptr for each filename entry
      0dc8 ca           1155          dex             ;still more to do ?
      0dc9 d0 fa        1156          bne fndp00      ;yes
      0dcb 60           1157          rts
                        1158    ;
                        1159    ;
                        1160    ;build a pointer to the filename area in memory
                        1161    ;
      0dcc a9 41        1162    setfp  lda #<dirstr
      0dce 85 fe        1163           sta ptr
      0dd0 a9 10        1164           lda #>dirstr
      0dd2 85 ff        1165           sta ptr+1
      0dd4 60           1166           rts
                        1167    ;
                        1168    ;
                        1169    ;move ptr to the next file entry (by adding 18)
                        1170    ;
      0dd5 18           1171    nxtfil clc
      0dd6 a5 fe        1172           lda ptr
      0dd8 69 12        1173           adc #18
      0dda 85 fe        1174           sta ptr
      0ddc 90 02        1175           bcc nxtf00
      0dde e6 ff        1176           inc ptr+1
      0de0 60           1177    nxtf00 rts
                        1178    ;
                        1179    ;================================================
                        1180    ;these are general purpose screen and message routines
                        1181    ;================================================
                        1182    ;
                        1183    ;this routine sets a pointer in vptr and cptr to the screen
                        1184    ;line sent in .x
                        1185    ;
      0de1 48           1186    setptr pha
      0de2 8a           1187           txa
      0de3 48           1188           pha
      0de4 0a           1189           asl a
      0de5 aa           1190           tax
      0de6 bd 0dfb      1191           lda scrlin,x
      0de9 85 6c        1192           sta vptr
      0deb 85 62        1193           sta cptr        ;color and screen lo bytes are the same
```

```
error addr  code          seq   source statement

      0ded bd 0dfc       1194          lda scrlin+1,x
      0df0 85 6d         1195          sta vptr+1
      0df2 18            1196          clc
      0df3 69 d4         1197          adc #>54272     ;offset to the color ram
      0df5 85 63         1198          sta cptr+1
      0df7 68            1199          pla
      0df8 aa            1200          tax
      0df9 68            1201          pla
      0dfa 60            1202          rts
                         1203    ;
      0dfb 0400 0428     1204    scrlin .word 1024,1064,1104,1144,1184
      0dff 0450 0478
      0e03 04a0
      0e05 04c8 04f0     1205          .word 1224,1264,1304,1344,1384
      0e09 0518 0540
      0e0d 0568
      0e0f 0590 05b8     1206          .word 1424,1464,1504,1544,1584
      0e13 05e0 0608
      0e17 0630
      0e19 0658 0680     1207          .word 1624,1664,1704,1744,1784
      0e1d 06a8 06d0
      0e21 06f8
      0e23 0720 0748     1208          .word 1824,1864,1904,1944,1984
      0e27 0770 0798
      0e2b 07c0
                         1209    ;
                         1210    ;
                         1211    ;=================================================
                         1212    ;this routine puts reverse spaces on .y lines of the screen
                         1213    ;starting at line .x in color .a
                         1214    ;=================================================
                         1215    ;
      0e2d 85 64         1216    doline sta color     ;save the color to do
      0e2f 84 65         1217           sty temp      ;save the line count
      0e31 20 0de1       1218    doli00 jsr setptr    ;build a pointer to the line in .x
      0e34 a0 27         1219           ldy #39
      0e36 a5 64         1220    doli10 lda color     ;store the color
      0e38 91 62         1221           sta (cptr),y
      0e3a a9 a0         1222           lda #$a0      ;store a reverse space
      0e3c 91 6c         1223           sta (vptr),y
      0e3e 88            1224           dey
      0e3f 10 f5         1225           bpl doli10
      0e41 e8            1226           inx           ;point .x to the next line
      0e42 c6 65         1227           dec temp      ;any more to do ?
      0e44 d0 eb         1228           bne doli00    ;yes
      0e46 60            1229           rts           ;all done
                         1230    ;
                         1231    ;=================================================
                         1232    ;call this routine to display message .a on the screen at the
                         1233    ;co-ordinates that are stored with the message.  The text is
                         1234    ;displayed in reverse video.
                         1235    ;=================================================
                         1236    ;
      0e47 0a            1237    mesage asl a          ;access table of message addresses
      0e48 aa            1238           tax
      0e49 bd 0e82       1239           lda msgtbl,x
      0e4c 85 fe         1240           sta ptr        ;build a pointer to the desired message
```

```
error addr  code         seq   source statement

      0e4e bd 0e83      1241          lda msgtbl+1,x
      0e51 85 ff        1242          sta ptr+1
      0e53 a0 00        1243          ldy #$00       ;find out which line to store the message on
      0e55 b1 fe        1244          lda (ptr),y
      0e57 aa           1245          tax
      0e58 20 0de1      1246          jsr setptr
      0e5b c8           1247          iny            ;make pointer absolute (only screen ptr
      0e5c b1 fe        1248          lda (ptr),y    ;is used because color has been done already)
      0e5e 18           1249          clc
      0e5f 65 6c        1250          adc vptr
      0e61 85 6c        1251          sta vptr
      0e63 90 02        1252          bcc mesa00
      0e65 e6 6d        1253          inc vptr+1
                        1254    ;
      0e67 a5 fe        1255    mesa00 lda ptr       ;move pointer to text so that it points
      0e69 18           1256          clc            ;to the text string with index y = 0
      0e6a 69 02        1257          adc #2
      0e6c 85 fe        1258          sta ptr
      0e6e 90 02        1259          bcc mesa10
      0e70 e6 ff        1260          inc ptr+1
      0e72 a0 00        1261    mesa10 ldy #$00      ;ok, pointers are set up
                        1262    ;
      0e74 b1 fe        1263    mesa15 lda (ptr),y   ;get a byte of the text
      0e76 f0 09        1264          beq mesa20     ;0 byte terminates it
      0e78 29 3f        1265          and #63        ;convert to reverse screen code
      0e7a 09 80        1266          ora #128
      0e7c 91 6c        1267          sta (vptr),y
      0e7e c8           1268          iny
      0e7f d0 f3        1269          bne mesa15
      0e81 60           1270    mesa20 rts           ;all done
                        1271    ;
      0e82 0e8e 0eae    1272    msgtbl .word msg1,msg2,msg3,msg4,msg5,msg6
      0e86 0ed9 0f04
      0e8a 0f2f 0f58
                        1273    ;
      0e8e 01 0a 31     1274    msg1   .byt 1,10,'1541 fast load by steve beats',0
      0e91 35 34 31
      0e94 20 46 41
      0e97 53 54 20
      0e9a 4c 4f 41
      0e9d 44 20 42
      0ea0 59 20 53
      0ea3 54 45 56
      0ea6 45 20 42
      0ea9 45 41 54
      0eac 53 00
      0eae 05 ˇ 50      1275    msg2   .byt 5,0,'please put your program disk in the 1541',0
      0eb1 4c 45 41
      0eb4 53 45 20
      0eb7 50 55 54
      0eba 20 59 4f
      0ebd 55 52 20
      0ec0 50 52 4f
      0ec3 47 52 41
      0ec6 4d 20 44
      0ec9 49 53 4b
      0ecc 20 49 4e
```

```
error addr  code          seq   source statement

      0ecf 20 54 48
      0ed2 45 20 31
      0ed5 35 34 31
      0ed8 00
      0ed9 05 00 50     1276    msg3   .byt 5,0,'programs on the disk "               " ',0
      0edc 52 4f 47
      0edf 52 41 4d
      0ee2 53 20 4f
      0ee5 4e 20 54
      0ee8 48 45 20
      0eeb 44 49 53
      0eee 4b 20 22
      0ef1 20 20 20
      0ef4 20 20 20
      0ef7 20 20 20
      0efa 20 20 20
      0efd 20 20 20
      0f00 20 22 20
      0f03 00
      0f04 16 00 43     1277    msg4   .byt 22,0,'cursor keys to select and return to load',0
      0f07 55 52 53
      0f0a 4f 52 20
      0f0d 4b 45 59
      0f10 53 20 54
      0f13 4f 20 53
      0f16 45 4c 45
      0f19 43 54 20
      0f1c 41 4e 44
      0f1f 20 52 45
      0f22 54 55 52
      0f25 4e 20 54
      0f28 4f 20 4c
      0f2b 4f 41 44
      0f2e 00
      0f2f 09 01 54     1278    msg5   .byt 9,1,'there are no programs on this disk !!! ',0
      0f32 48 45 52
      0f35 45 20 41
      0f38 52 45 20
      0f3b 4e 4f 20
      0f3e 50 52 4f
      0f41 47 52 41
      0f44 4d 53 20
      0f47 4f 4e 20
      0f4a 54 48 49
      0f4d 53 20 44
      0f50 49 53 4b
      0f53 20 21 21
      0f56 21 00
      0f58 0b 01 52     1279    msg6   .byt 11,1,'return to change disks or stop to quit ',0
      0f5b 45 54 55
      0f5e 52 4e 20
      0f61 54 4f 20
      0f64 43 48 41
      0f67 4e 47 45
      0f6a 20 44 49
      0f6d 53 4b 53
      0f70 20 4f 52
```

```
error addr  code         seq   source statement

      0f73 20 53 54
      0f76 4f 50 20
      0f79 54 4f 20
      0f7c 51 55 49
      0f7f 54 00
                        1280   ;
                        1281   ;================================================
                        1282   ;this is the data for the sprites that make up the title
                        1283   ;================================================
                        1284   ;
      0f81 55 55 54     1285   titlez .byt $55,$55,$54,$6a,$aa,$a4,$6a,$aa
      0f84 6a aa a4
      0f87 6a aa
      0f89 a4 6a aa     1286          .byt $a4,$6a,$aa,$a4,$55,$5a,$a4,$00
      0f8c a4 55 5a
      0f8f a4 00
      0f91 1a 90 00     1287          .byt $1a,$90,$00,$1a,$90,$00,$6a,$40
      0f94 1a 90 00
      0f97 6a 40
      0f99 00 6a 40     1288          .byt $00,$6a,$40,$01,$a9,$00,$01,$a9
      0f9c 01 a9 00
      0f9f 01 a9
      0fa1 00 06 a4     1289          .byt $00,$06,$a4,$00,$06,$a4,$00,$1a
      0fa4 00 06 a4
      0fa7 00 1a
      0fa9 90 00 1a     1290          .byt $90,$00,$1a,$90,$00,$6a,$95,$54
      0fac 90 00 6a
      0faf 95 54
      0fb1 6a aa a4     1291          .byt $6a,$aa,$a4,$6a,$aa,$a4,$6a,$aa
      0fb4 6a aa a4
      0fb7 6a aa
      0fb9 a4 55 55     1292          .byt $a4,$55,$55,$54,$00,$00,$00,$24
      0fbc 54 00 00
      0fbf 00 24
                        1293   ;
      0fc1 55 55 54     1294   titlea .byt $55,$55,$54,$6a,$aa,$a4,$6a,$aa
      0fc4 6a aa a4
      0fc7 6a aa
      0fc9 a4 6a aa     1295          .byt $a4,$6a,$aa,$a4,$69,$55,$a4,$69
      0fcc a4 69 55
      0fcf a4 69
      0fd1 01 a4 69     1296          .byt $01,$a4,$69,$01,$a4,$69,$01,$a4
      0fd4 01 a4 69
      0fd7 01 a4
      0fd9 69 55 a4     1297          .byt $69,$55,$a4,$6a,$aa,$a4,$6a,$aa
      0fdc 6a aa a4
      0fdf 6a aa
      0fe1 a4 6a aa     1298          .byt $a4,$6a,$aa,$a4,$69,$55,$a4,$69
      0fe4 a4 69 55
      0fe7 a4 69
      0fe9 01 a4 69     1299          .byt $01,$a4,$69,$01,$a4,$69,$01,$a4
      0fec 01 a4 69
      0fef 01 a4
      0ff1 69 01 a4     1300          .byt $69,$01,$a4,$69,$01,$a4,$69,$01
      0ff4 69 01 a4
      0ff7 69 01
      0ff9 a4 55 01     1301          .byt $a4,$55,$01,$54,$00,$00,$00,$54
```

```
error addr  code         seq   source statement

      0ffc 54 00 00
      0fff 00 54
                         1302  ;
      1001 55 55 54      1303  titlep .byt $55,$55,$54,$6a,$aa,$a4,$6a,$aa
      1004 6a aa a4
      1007 6a aa
      1009 a4 6a aa      1304        .byt $a4,$6a,$aa,$a4,$69,$55,$a4,$69
      100c a4 69 55
      100f a4 69
      1011 01 a4 69      1305        .byt $01,$a4,$69,$01,$a4,$69,$01,$a4
      1014 01 a4 69
      1017 01 a4
      1019 69 55 a4      1306        .byt $69,$55,$a4,$6a,$aa,$a4,$6a,$aa
      101c 6a aa a4
      101f 6a aa
      1021 a4 6a aa      1307        .byt $a4,$6a,$aa,$a4,$69,$55,$54,$69
      1024 a4 69 55
      1027 54 69
      1029 00 00 69      1308        .byt $00,$00,$69,$00,$00,$69,$00,$00
      102c 00 00 69
      102f 00 00
      1031 69 00 00      1309        .byt $69,$00,$00,$69,$00,$00,$69,$00
      1034 69 00 00
      1037 69 00
      1039 00 55 00      1310        .byt $00,$55,$00,$00,$00,$00,$00,$c4
      103c 00 00 00
      103f 00 c4
                         1311  ;
      1041 00            1312  dirstr .byte 0 ;where directory entries ae stored
                         1313        .end
  0 errors detected
```

symbol table
(blank) = label, (=) = symbol, (+)= multibly defined

```
ad2061  080d  atnin  =0080  atnout =0008  bcnt15 =000f  bcnt64 =00fc  bdir   098b  bdir00  0992  bdir05  099f  bdir10  09a1
bdir20  09ab  bdir25  09b6  bdir30  09bb  bdir35  09ce  bdir40  09d8  bdir80  09e9  brstld  08da  buf1   =0300  buf2   =0400
buf3   =0500  buf4   =0600  buf5   =0700  bufptr =000a  bump   =00c0  byte15 =000e  byte64 =00fb  chkin  =ffc6  chrout =ffd2
cin15  =0004  cin64  =0040  ckout  =ffc9  close  =ffc3  clrchn =ffcc  cmdlp  08ac  color  =0064  cout15 =0008  cout64 =0010
cptr   =0062  curlin =02aa  curprg =02a8  dcend  0a76  dclen  =01d2  dcntr  =02ad  dcode  08a4  ddest  08a1  din15  =0001
din64  =0080  dirstr  1041  dodir  08d7  dojmp  08b5  doli00  0e31  doli10  0e36  doline  0e2d  doseek  08c4  dout15 =0002
dout64 =0020  down00  0839  down10  0840  down15  084d  down16  0868  down20  0873  down25  0889  dspf00  0d60  dspf10  0d6c
dspf15  0d70  dspf20  0d8e  dspf25  0d91  dspf30  0dac  dspf50  0dad  dspf55  0db6  dspfil  0d4d  dspm00  0c79  dspm10  0c7c
dspm15  0c88  dspm16  0c99  dspm17  0ca8  dspm20  0cae  dspm25  0ccc  dspm30  0cd2  dspm40  0cea  dspm50  0cf9  dspm55  0d00
dspm56  0d07  exec   =00e0  flo10  0a23  flo15  0a2f  flo20  0a31  flo30  0a35  flo50  0a4b  fload  0a03  fndp00  0dc5
fndprg  0dbc  fprg   =02a9  fr0015  08ec  fr0064  0b39  fr1064  0b47  fram15  08dd  fram64  0b31  fs1500  0a56  fs1510  0a62
fs6400  0b1e  fstb15  0a4f  fstb64  0b18  getin  =ffe4  gotd00  0c60  gotdir  0c37  gotprg  0af8  gs     0978  gt     097d
hdrs   =0006  ib1015  093a  ib1064  0a82  ib2015  0946  ib2064  0a8c  ibyt15  092f  ibyt64  0a77  joboff =000c  jobs   =0000
jumpc  =00d0  l15     08d0  lpb15  0780  lpb64  =00fd  memcmd  089e  menu   0b56  menu10  0b79  menu20  0b84  menu25  0b86
menu35  0ba5  menu40  0bbb  menu45  0bcf  menu50  0bf4  menu55  0bfd  menu60  0c12  menu65  0c22  mesa00  0e67  mesa10  0e72
mesa15  0e74  mesa20  0e81  mesage  0e47  msg1   0e8e  msg2   0eae  msg3   0ed9  msg4   0f04  msg5   0f2f  msg6   0f58
msgtbl  0e82  nopr00  0c59  noprog  0c4f  nxtf00  0de0  nxtfil  0dd5  ob1015  0906  ob1064  0aae  ob2015  0915  ob2064  0aba
ob3015  0918  obyt15  08fb  obyt64  0aa3  open   =ffc0  pb15   =1800  pb64   =dd00  postbl  0b50  prgcnt =02a7  ptr    =00fe
ptr2   =006a  read   =0080  rfblok  0957  rnb100  098a  rnblok  0961  scend   0b4f  scrlin  0dfb  sct00   0ada  sct10   0ae3
sctop   0ad0  secsek =00b8  seek   =00b0  setfp   0dcc  setlfs =ffba  setnam =ffbd  setptr  0de1  snex00  09f9  snext   09ec
sysprg  0b15  temp   =0065  titlea  0fc1  titlep  1001  titlez  0f81  vartab =002d  vptr   =006c  write  =0090  wrklin =02ab
wrkoff =000d  wrkprg =02ac  wverfy =00a0
```

cross reference
(<#> = definition, <$> = write, <blank> = read)

cross reference
(<#> = definition, <$> = write, <blank> = read)

```
doseek  08c4     233    248#
dout15 =0002      56#    212    283    292    320    328    577    585
dout64 =0020       8#    666    676
down00  0839     133#    169    173
down10  0840     137#    141
down15  084d     144#    148
down16  0868     155    158#
down20  0873     162    165#
down25  0889     171    182#
dspf00  0d60    1092#   1126   1130
dspf10  0d6c    1095    1097#
dspf15  0d70    1099#   1106
dspf20  0d8e    1113    1117#
dspf25  0d91    1119#
dspf30  0dac    1131#   1135
dspf50  0dad    1122    1133#
dspf55  0db6    1138#   1140
dspfil  0d4d     971    1084#
dspm00  0c79     971#    994    1008   1010   1022   1029
dspm10  0c7c     972#    973     984   1000   1018   1020   1033
dspm15  0c88     975     978#
dspm16  0c99     986#
dspm17  0ca8     989     993#
dspm20  0cae     979     996#
dspm25  0ccc    1005    1009#
dspm30  0cd2     997    1012#
dspm40  0cea    1013    1024#
dspm50  0cf9    1025    1031#
dspm55  0d00    1032    1042#
dspm56  0d07    1045#   1049
exec   =00e0      71#
flo10   0a23     532#    554
flo15   0a2f     535     539#
flo20   0a31     528     540#
flo30   0a35     543#    547
flo50   0a4b     551     559#
fload   0a03     270     515#
fndp00  0dc5    1151    1154#  1156
fndprg  0dbc    1062    1090   1148#
fprg   =02a9.     42#    960$    990$   991$   1006$  1007$  1088
fr0015  08ec     287#    290
fr0064  0b39     777#    780
fr1064  0b47     785#    787
fram15  08dd     278#    281     308    351
fram64  0b31     619     659     773#
fs1500  0a56     574#    588
fs1510  0a62     576     579#
fs6400  0b1e     749#    752     757
fstb15  0a4f     541     544     568#
fstb64  0b18     700     706     744#
getin  =ffe4      18#    950     972
gotd00  0c60     938     959#
gotdir  0c37     916     937#
gotprg  0af8     702     724#
gs      0978     401$    405#
gt      097d     400$    407#
hdrs   =0006      62#     82      83     84     85     86    249$   251$   406$   408$
```

cross reference
(<#> = definition, <$> = write, <blank> = read)

cross reference
(<#> = definition, <$> = write, <blank> = read)

```
read   =0080      64#   411
rfblok  0957     386#   431    519
rnbl00  098a     409    413#
rnblok  0961     393#   504
scend   0b4f     791#  1044
scrlin  0dfb    1191   1194   1204#
sct00   0ada     700#   716    718
sct10   0ae3     706#   710
sctop   0ad0     695#   700    706   1044   1045
secsek =00b8      68#
seek   =00b0      67#    68    252
setfp   0dcc     910   1149   1162#
setlfs =ffba      20#   111
setnam =ffbd      21#   113
setptr  0de1     893   1085   1186#  1218   1246
snex00  09f9     501#   502
snext   09ec     433    451    494#   520    552
sysprg  0b15     730    733    738#
temp   =0065      35#  1097$  1100   1217$  1227$
titlea  0fc1    1294#
titlep  1001    1303#
titlez  0f81     820   1285#
vartab =002d      36#   696$   698$   707$   714    715$   717$
vptr   =006c      32#   900$  1101$  1109   1111$  1114$  1138$  1192$  1195$  1223$  1250   1251$  1253$  1267$
write  =0090      65#
wrklin =02ab      44#  1087$  1127$  1128
wrkoff =000d      84#   394$   398
wrkprg =02ac      45#  1089$  1093   1119   1123$
wverfy =00a0      66#
```

CHAPTER 8

64 Fast Load #3

Hardware: C64 with a 1541, 1571 or a 1581

1) 2-3x speed up
2) User interface supports all units
3) Loads to any address between $0100 - $feff

Normal LOAD time for 112 blocks	Fast Load #3 LOAD time for 112 blocks
1 min 17 sec	1541: 26 sec 1571: 26 sec 1581: 39 sec

Known bugs: If a 1581 is unit 8 and a 1541 or 1571 is unit 9, an attempt to load from unit 9 will cause the program to crash.

Fast load #3 includes 2 fast load routines. One that works with the screen on (fast) and one that works with the screen off (extra-fast). Some code gets downloaded to the drive. This allows us to work with the drive's job queue directly and to use our own serial handshake.

Once all the routines are in place, the fast routine is used to send the list of program files from the drive to the c64. The user then selects the file he wants to load. On the 1541 and 1571, the screen will blank as the extra-fast routine is used to fetch the user's file.

On the 1581, the screen doesn't blank. Instead the fast routine is used to fetch the file. The extra-fast routine won't work. The 1581 runs at 2 mHz instead of 1 mHz like the 1541 and 1571. Because of this, the extra-fast routine does not work with the 1581.

The 1581 has other differences which make patches to the code necessary:

1) Header, directory and BAM on track 40, not 18
2) Job queue and job parameters in new location
3) Job return codes slightly different
4) Serial port at $4001 instead of $1c00
5) Zero page usage different
6) Only fast routine can be used, extra-fast won't work

The program determines which drive is in use. If it is the 1581, the program modifies itself to account for the differences shown above.

The small section of code that does the actual loading is moved up to $ff45 and run. It is put here so it is not overwritten by the load. BASIC programs finish with a bit of code in the cassette buffer. None-BASIC programs get finished with a bit of code in zero-page.

```
error addr  code          seq   source statement

                           1    ;=========================
                           2    ;these are declarations for
                           3    ;constants and variables that
                           4    ;will be used within the
                           5    ;c64 during program runs
                           6    ;=========================
                           7    ;
          =dd00            8    pb64   =$dd00     ;port containing serial i/o bits
          =0080            9    din64  =$80       ;data in
          =0020           10    dout64 =$20       ;data out
          =0040           11    cin64  =$40       ;clock in
          =0010           12    cout64 =$10       ;clock out
          =0008           13    atnout =$08       ;atn output (not used)
                          14    ;
                          15    ;kernal routines
                          16    ;
          =ffc6           17    chkin  =$ffc6     ;make file input
          =ffc9           18    ckout  =$ffc9     ;make file output
          =ffcc           19    clrchn =$ffcc     ;clear channels
          =ffe4           20    getin  =$ffe4     ;get a character from channel
          =ffd2           21    chrout =$ffd2     ;print a character to channel
          =ffba           22    setlfs =$ffba     ;set logical file number and device
          =ffbd           23    setnam =$ffbd     ;set file name for open
          =ffc0           24    open   =$ffc0     ;open a file
          =ffc3           25    close  =$ffc3     ;close a file
                          26    ;
                          27    ;zero page variables
                          28    ;
          =00fb           29    byte64 =$fb       ;incoming or outgoing byte during fast i/o
          =00fc           30    bcnt64 =$fc       ;counter for bytes to send or receive
          =00fd           31    lpb64  =$fd       ;copy of last state of the port for burst load
          =00fe           32    ptr    =$fe       ;pointer for data fetches or stores
          =006a           33    ptr2   =$6a       ;work pointer (in fac #2)
          =006c           34    vptr   =$6c       ;pointer to screen
                          35    ;
          =0062           36    cptr   =$62       ;pointer to color (in fac #1)
          =0064           37    color  =$64       ;current color
          =0065           38    temp   =$65       ;temporary variable
          =002d           39    vartab =$2d       ;end of program pointer
                          40    ;
                          41    ;non zero page variables
                          42    ;
          =02a7           43    prgcnt =$02a7     ;number of pgms on disk
          =02a8           44    curprg =$02a8     ;current program cursor is on
          =02a9           45    fprg   =$02a9     ;1st program on screen
          =02aa           46    curlin =$02aa     ;current menu line cursor is on
          =02ab           47    wrklin =$02ab     ;used during
          =02ac           48    wrkprg =$02ac     ; filename display
          =02ad           49    dcntr  =$02ad     ;counter for bytes downloaded to disk
          =02af           50    drtype =$02af     ;drive type
          =02b0           51    unitno =$02b0
          =ff45           52    bzcode =$ff45     ;memory where loader code will go
                          53    ;
                          54    ;=======================
                          55    ;these are the declarations
                          56    ;for constants and varbs
                          57    ;used within the 1541 drive
```

```
error addr  code          seq   source statement

                           58   ;during program runs
                           59   ;====================
                           60   ;
                           61   ;
          =1800            62   pb15   =$1800     ;port containing serial i/o bits
          =0001            63   din15  =$01       ;data in line
          =0002            64   dout15 =$02       ;data out line
          =0004            65   cin15  =$04       ;clock in line
          =0008            66   cout15 =$08       ;clock out line
          =0080            67   atnin  =$80       ;atn input (not used)
                           68   ;
          =0000            69   jobs   =$00       ;job queue for the controller
          =0006            70   hdrs   =$06       ;arguments for each job (only first 4 are used)
                           71   ;
          =0080            72   read   =$80       ;these are the codes for each job to be performed
          =00b0            73   seek   =$b0
                           74   ;
          =0300            75   buf1   =$0300     ;addresses of each buffer in disk ram
          =0400            76   buf2   =$0400     ;first 2 buffers are used for data
          =0500            77   buf3   =$0500     ;this is where the static code lives
          =0600            78   buf4   =$0600     ; - it is downloaded to here
          =0700            79   buf5   =$0700     ;this buffer is for variables
                           80   ;
                           81   ;zero page storage used by the disk routines,
                           82   ;uses hdrs associated with buffers 3 through 5
                           83   ;because these are never used by the controller
                           84   ;
          =0010            85   bufptr =$10  ;pointer to current active buffer
          =000f            86   joboff =$0f  ;index to current active job
          =000a            87   wrkoff =$0a  ;previous active job          ($12 on 1581)
          =000b            88   byte15 =$0b  ;current input/output byte    ($13 on 1581)
          =000c            89   bcnt15 =$0c  ;counter for multiple byte i/o ($14 on 1581)
                           90   ;
                           91   ;non zero-page storage
                           92   ;for the disk routines
                           93   ;
          =0780            94   *=buf5+$80        ;just use the upper half of buffer 5
                           95   ;
     0780 =0781            96   lpb15 *=*+1       ;copy of last port value during burst load
                           97   ;
                           98   ;
                           99   ;================================================
                          100   ; This section of code downloads fast i/o routines to
                          101   ;buffer 5 in the drive.  The fast i/o routines are then
                          102   ;used for all other code transfers between the drive and
                          103   ;c64.
                          104   ;================================================
                          105   ;
                          106   ;
          =0801           107   *=$0801  (c64 BASIC start address)
                          108   ;
                          109   ;this data forms a basic line 10 that says sys2061
                          110   ;
     0801 0b 08 0a        111         .byte $0b,$08,$0a,0,$9e, 2061 ,0,0,0
     0804 00 9e 32
     0807 30 36 31
     080a 00 00 00
```

```
error addr  code        seq   source statement

                        112   ;
            080d        113   ad2061
            080d a9 08  114         lda #8
            080f 8d 02b0 115        sta unitno  ;set up for open
            0812 d0 15  116         bne opener  ;and bra
            0814        117   reopen
            0814 a9 0f  118         lda #15
            0816 20 ffc3 119        jsr close
            0819 20 ffcc 120        jsr clrchn
            081c ee 02b0 121        inc unitno
            081f ad 02b0 122        lda unitno
            0822 c9 0c  123         cmp #12
            0824 d0 03  124         bne opener
            0826 6c fffc 125        jmp ($fffc)
            0829        126   opener
            0829 a9 0f  127         lda #15     ;open 15,8,15
            082b a8     128         tay         ;
            082c ae 02b0 129        ldx unitno  ;
            082f 20 ffba 130        jsr setlfs
            0832 a9 00  131         lda #$00
            0834 20 ffbd 132        jsr setnam
            0837 20 ffc0 133        jsr open
            083a b0 d8  134         bcs reopen ;on error, try again
                        135   ;
            083c a9 95  136         lda #<dcode   ;build a pointer to the disk code
            083e 85 fe  137         sta ptr
            0840 a9 09  138         lda #>dcode
            0842 85 ff  139         sta ptr+1
                        140   ;
            0844 a9 00  141         lda #<buf3    ;build address of dest into
            0846 8d 08e3 142        sta ddest     ;mem-write command.
            0849 a9 05  143         lda #>buf3    ;
            084b 8d 08e4 144        sta ddest+1   ;
                        145   ;
            084e a9 d2  146         lda #<dclen   ;counter for length of disk code
            0850 8d 02ad 147        sta dcntr
            0853 a9 01  148         lda #>dclen
            0855 8d 02ae 149        sta dcntr+1
            0858 a2 0f  150         ldx #15
            085a 20 ffc9 151        jsr ckout     ;get drive type by using
            085d b0 b5  152         bcs reopen    ;mem-read at $e5c6 in drive
            085f a2 06  153         ldx #$06
            0861 20 08f2 154        jsr domemc    ;This is part of
            0864 b0 ae  155         bcs reopen    ;power on msg in 1541 and 1571
            0866 20 ffcc 156        jsr clrchn
            0869 a2 0f  157         ldx #15       ;1541=4
            086b 20 ffc6 158        jsr chkin     ;1571=7
            086e 20 ffe4 159        jsr getin     ;1581=$ff
            0871 48     160         pha
            0872 20 ffcc 161        jsr clrchn
            0875 68     162         pla
            0876 8d 02af 163        sta drtype
            0879 c9 ff  164         cmp #$ff
            087b d0 05  165         bne no1581
                        166   ;
                        167   ;============================
                        168   ;NOTE: The fix19 routine makes
```

```
error addr  code        seq   source statement

                        169   ;the neceesary patches to the
                        170   ;drive code so it will work
                        171   ;with the 1581.
                        172   ;
      087d 20 0902      173         jsr fix19
      0880 a9 38        174         lda #$38
      0882 8d 100a      175   no1581 sta drmssg
      0885 8d 1041      176         sta drmsg2
                        177   ;
                        178   ;====================
                        179   ;
                        180   ;this is the main loop to download
                        181   ;code in 32 byte sections
                        182   ;
      0888 a2 0f        183   down00 ldx #15       ;make channel 15 output
      088a 20 ffc9      184          jsr ckout
      088d a2 00        185          ldx #$00      ;send command to disk
      088f 20 08f2      186          jsr domemc
      0892 a0 00        187          ldy #$00      ;now send 32 bytes of disk code
      0894 b1 fe        188   down15 lda (ptr),y
      0896 20 ffd2      189          jsr chrout
      0899 c8           190          iny
      089a c0 20        191          cpy #32
      089c d0 f6        192          bne down15
                        193   ;
      089e 20 ffcc      194          jsr clrchn
      08a1 18           195          clc
      08a2 ad 08e3      196          lda ddest     ;move the destination pointer up 32 bytes
      08a5 69 20        197          adc #32
      08a7 8d 08e3      198          sta ddest
      08aa 90 03        199          bcc down16
      08ac ee 08e4      200          inc ddest+1
                        201   ;
      08af 18           202   down16 clc           ;move pointer to the source data up 32 bytes
      08b0 a5 fe        203          lda ptr
      08b2 69 20        204          adc #32
      08b4 85 fe        205          sta ptr
      08b6 90 02        206          bcc down20
      08b8 e6 ff        207          inc ptr+1
                        208   ;
      08ba ad 02ad      209   down20 lda dcntr     ;check if we have downloaded all data
      08bd 38           210          sec
      08be e9 20        211          sbc #32
      08c0 8d 02ad      212          sta dcntr
      08c3 b0 c3        213          bcs down00    ;still more data to send
      08c5 ad 02ae      214          lda dcntr+1   ;check the hi byte
      08c8 f0 06        215          beq down25    ;all data has been sent
      08ca ce 02ae      216          dec dcntr+1
      08cd 4c 0888      217          jmp down00
                        218   ;
                        219   ;
                        220   ;=======================================
                        221   ;gets to here when all code has been downloaded,
                        222   ;start the disk code running with a m-e command
                        223   ;give it time to install itself before using.
                        224   ;=======================================
                        225   ;
```

```
error addr  code         seq   source statement

      08d0 a2 0f        226   down25 ldx #15      ;make channel 15 output
      08d2 20 ffc9      227          jsr ckout
      08d5 a2 0c        228          ldx #$0c
      08d7 20 08f2      229          jsr domemc
      08da 20 ffcc      230          jsr clrchn    ;this will leave dout64 and cout64 = 0
      08dd 4c 0c79      231          jmp menu      ;all ready to go so print the menu
                        232   ;
                        233   ;
      08e0 4d 2d 57     234   memcmd .byte 'm-w'   ;command to do a memory write
      08e3 0500         235   ddest  .word buf3    ;variable, where to write to
      08e5 20           236          .byte 32      ;constant, number of bytes to be written
                        237   ;
      08e6 4d 2d 52     238   memred .byte 'm-r'   ;command to do a memory write  (x=6)
      08e9 c6 e5        239          .byte 198,229 ;($e5c6)
      08eb 01           240          .byte 1       ;constant
                        241   ;
      08ec 4d 2d 45     242   memexe .byte 'm-e'   ;command to do a memory execute   (x=c)
      08ef 00 05 00     243          .byte 0,5,0   ;at $0500
                        244   ;
      08f2 a0 06        245   domemc ldy #$06
      08f4 bd 08e0      246   nxmemc lda memcmd,x
      08f7 20 ffd2      247          jsr chrout
      08fa b0 05        248          bcs donemc
      08fc e8           249          inx
      08fd 88           250          dey
      08fe d0 f4        251          bne nxmemc    ;nope, not yet
      0900 18           252          clc
      0901 60           253   donemc rts
                        254   ;
                        255   ;
                        256   ;
                        257   ;=======================
                        258   ; patches for 1581 version
                        259   ;=======================
                        260   ;
      0902 a0 0b        261   fix19  ldy #$0b ;adjust headers
      0904 8c 09b8      262          sty h1+1 ;address for 1581,
      0907 8c 0a72      263          sty h4+1
      090a c8           264          iny
      090b 8c 09bc      265          sty h2+1
      090e 8c 0a6d      266          sty h3+1
                        267   ;
      0911 a0 02        268          ldy #$02 ;adjust job
      0913 8c 09c0      269          sty j1+1 ;queue
      0916 8c 09c2      270          sty j2+1 ;location,
      0919 8c 0a7a      271          sty j3+1 ;for 1581
      091c 8c 0aeb      272          sty j4+1
                        273   ;
      091f a0 12        274          ldy #$12 ;adjust
      0921 8c 0a55      275          sty w1+1 ;zero-page
      0924 8c 0a5c      276          sty w2+1 ;locations,
                        277   ;
      0927 a0 13        278          ldy #$13 ;adjust
      0929 8c 09ed      279          sty b1+1 ;more z-page
      092c 8c 0a01      280          sty b2+1 ;locations,
      092f 8c 0a23      281          sty b3+1
      0932 8c 0a36      282          sty b4+1
```

```
error addr  code        seq  source statement

      0935 8c 0a46      283          sty b5+1
                        284  ;
      0938 a0 14        285          ldy #$14 ;and more,
      093a 8c 0abe      286          sty c1+1
      093d 8c 0ac6      287          sty c2+1
                        288  ;
      0940 a9 4c        289          lda #$4c     ;substitute
      0942 8d 0b40      290          sta fstb15   ; obyt15 for
      0945 20 0b68      291          jsr fixd81   ; fstb15,
                        292  ;
      0948 a2 2b        293          ldx #$2b     ;replace
      094a bd 0b7f      294  nxmcod lda ibyt64,x ; fstb64 code
      094d 9d 0c28      295          sta fstb64,x ; with
      0950 ca           296          dex          ; ibyt64 code,
      0951 10 f7        297          bpl nxmcod
                        298  ;
      0953 a9 28        299          lda #40      ;fix track
      0955 8d 09b6      300          sta doseek+1 ;for
      0958 8d 0a7d      301          sta bdir+1   ;dir/bam
      095b a9 04        302          lda #4       ;and
      095d 8d 0a91      303          sta bdir05+1 ;location
      0960 a9 14        304          lda #20      ;of
      0962 8d 0a99      305          sta bdir15+1 ;header
                        306  ;
      0965 a9 01        307          lda #$01     ;fix cia addr
      0967 a2 01        308          ldx #$01     ;lo
      0969 20 0970      309          jsr fixcia
      096c a9 40        310          lda #$40     ;fix
      096e a2 02        311          ldx #$02     ;hi and done!
                        312  ;
      0970 9d 0995      313  fixcia sta p1,x
      0973 9d 099a      314         sta p2,x
      0976 9d 09ce      315         sta p3,x
      0979 9d 09d9      316         sta p4,x
      097c 9d 09dd      317         sta p5,x
      097f 9d 09e8      318         sta p6,x
      0982 9d 09f7      319         sta p7,x
      0985 9d 0a06      320         sta p8,x
      0988 9d 0a09      321         sta p9,x
      098b 9d 0a14      322         sta p10,x
      098e 9d 0a2b      323         sta p11,x
      0991 9d 0a37      324         sta p12,x
      0994 60           325         rts
                        326  ;
                        327  ;===============================================
                        328  ;  This is the code that gets downloaded to the drive.
                        329  ;it contains two fast transfer routines for the 15x1.
                        330  ;
                        331  ;  I/O routine #1 works with the screen and interrupts
                        332  ;turned on.  This is used for sending commands to the
                        333  ;drive and returning errors to the 64.  I/O routine #2
                        334  ;is a faster handshake that works with the screen
                        335  ;turned off.  If the drive is a 1581, I/O routine #2
                        336  ;is not used!  Instead messy code patches are made so
                        337  ;that only the #1 style handshake is used.
                        338  ;
                        339  ;  Bug: if unit 8 is 1581 and unit 9 is a 1541,
```

```
error addr  code          seq   source statement

                          340   ;  a load from unit 9 will cause patched i/o
                          341   ;  routines to be used and the program will crash!
                          342   ;
                          343   ;  The jsr commands use a computed address so that this
                          344   ;code can be assembled with the main 64 code but will have
                          345   ;the correct address when the code is in the disk buffers.
                          346   ;=================================================
                          347   ;
      0995                348   p1
      0995 ad 1800        349   dcode  lda pb15
      0998 29 f5          350          and #$ff-dout15-cout15  ;clear clock and data lines
      099a 8d 1800        351   p2     sta pb15
                          352   ;
                          353   ;
                          354   ;====================================
                          355   ;this is the main loop for the drive code.
                          356   ;it waits for the 64 to send a command
                          357   ;byte and then calls the correct routine.
                          358   ;
                          359   ;command bytes are:
                          360   ;
                          361   ;0     quit and return control to the dos
                          362   ;1     do a seek and return the status to the 64
                          363   ;2     do the directory search and send routine
                          364   ;3     do the burst load from a given start
                          365   ;       track and sector
                          366   ;====================================
                          367   ;
      099d 20 058b        368   cmdlp  jsr ibyt15-dcode+buf3    ;get a command from the 64
      09a0 20 0511        369          jsr dojmp-dcode+buf3     ;call the correct routine
      09a3 4c 0508        370          jmp cmdlp-dcode+buf3     ;and get another command.
                          371   ;
      09a6 c9 01          372   dojmp  cmp #1           ;does the 64 want to do a seek on the disk ?
      09a8 f0 0b          373          beq doseek       ;yes
      09aa c9 02          374          cmp #2           ;is this a directory command ?
      09ac f0 1a          375          beq dodir        ;yes
      09ae c9 03          376          cmp #3           ;is it a burst load command ?
      09b0 f0 19          377          beq brstld       ;yes
      09b2 68             378          pla              ;no, any other value returns to dos
      09b3 68             379          pla              ;by scrapping return address to cmdlp
      09b4 60             380          rts              ;and returning to caller of this code
                          381   ;
                          382   ;====================================
                          383   ;this code just does a seek on directory
                          384   ;and returns the error code to the 64.
                          385   ;used to check for a disk is in the drive
                          386   ;====================================
                          387   ;
      09b5 a9 12          388   doseek lda #18      ;set up track and sector for the seek
      09b7 85 06          389   h1     sta hdrs
      09b9 a9 00          390          lda #0
      09bb 85 07          391   h2     sta hdrs+1
      09bd a9 b0          392          lda #seek
      09bf 85 00          393   j1     sta jobs
                          394   ;
      09c1                395   j2
      09c1 a5 00          396   l15    lda jobs
```

```
error addr  code          seq   source statement

      09c3 30 fc          397         bmi 115             ;wait for the seek to finish
      09c5 4c 0557        398         jmp obyt15-dcode+buf3 ;and send status to the 64
                          399   ;
                          400   ;
                          401   ;==================================
                          402   ;call the directory routine with a jump
                          403   ;==================================
                          404   ;
      09c8 4c 05e7        405   dodir  jmp bdir-dcode+buf3
                          406   ;
                          407   ;==================================
                          408   ;call the burst load routine with a jump
                          409   ;==================================
                          410   ;
      09cb 4c 065f        411   brstld jmp fload-dcode+buf3
                          412   ;
                          413   ;=========================================
                          414   ; This is the frame handshake that starts transfer
                          415   ;of 8 bits in any direction.  It is called by the
                          416   ;input and output routines to put the 64 and 1541
                          417   ;code in synch for a faster handshake on bits
                          418   ;=========================================
                          419   ;
      09ce                420   p3
      09ce ad 1800        421   fram15 lda pb15         ;wait for clkin = 1 (64 is ready for the byte)
      09d1 a8             422         tay             ;save the port value for later
      09d2 29 04          423         and #cin15
      09d4 f0 f8          424         beq fram15      ;64 isn't ready yet
      09d6 98             425         tya             ;acknowledge 64's ready signal with datout=1
      09d7 09 02          426         ora #dout15     ;64 will see this as datin = 0
      09d9 8d 1800        427   p4    sta pb15
      09dc 78             428         sei             ;disk can't have any irq's now
                          429   ;
      09dd                430   p5
      09dd ad 1800        431   fr0015 lda pb15         ;now wait for the 64 to acknowledge again
      09e0 a8             432         tay             ;by resetting clkin to 0
      09e1 29 04          433         and #cin15
      09e3 d0 f8          434         bne fr0015      ;hasn't answered us yet
      09e5 98             435         tya             ;finalise frame handshake by setting datout=0
      09e6 29 fd          436         and #$ff-dout15 ;64 will see this as datin=1
      09e8 8d 1800        437   p6    sta pb15
      09eb 60             438         rts             ;now go and do the bit handshake
                          439   ;
                          440   ;
                          441   ;==============================
                          442   ;this routine sends a byte of data
                          443   ;to the 64 using fast handshake #1
                          444   ;enter with .a=byte to send.
                          445   ;==============================
                          446   ;
      09ec                447   b1
      09ec 85 0b          448   obyt15 sta byte15       ;store the byte being sent
      09ee 8a             449         txa             ;save the .x register
      09ef 48             450         pha
      09f0 98             451         tya
      09f1 48             452         pha                 ;save the .y register
      09f2 a2 08          453         ldx #$08            ;keep a bit counter
```

```
error addr  code        seq   source statement

      09f4 20 0539      454         jsr fram15-dcode+buf3  ;go do a frame handshake
                        455   ;
                        456   ;the following handshake is performed for each bit that is sent to the 64
                        457   ;
      09f7              458   p7
      09f7 ad 1800      459   ob1015 lda pb15         ;wait for clkin=1 (64 is ready for the bit)
      09fa a8           460          tay              ;save port value for later
      09fb 29 04        461          and #cin15
      09fd f0 f8        462          beq ob1015       ;64 isn't ready yet
                        463   ;
      09ff 98           464          tya              ;64 expects the bit to be valid very soon!
      0a00 46 0b        465   b2     lsr byte15       ;datout is 0 at the moment so see what to send
      0a02 b0 02        466          bcs ob2015       ;if a 1 is needed, send 0 to complement data
      0a04 09 02        467          ora #dout15      ;a 0 is needed so send a 1 to complement the data
      0a06              468   p8
      0a06 8d 1800      469   ob2015 sta pb15         ;present the 64 with it's data
                        470   ;
      0a09              471   p9
      0a09 ad 1800      472   ob3015 lda pb15         ;wait for the 64 to say it has the data
      0a0c a8           473          tay              ;it will set clkin = 0 when it has
      0a0d 29 04        474          and #cin15
      0a0f d0 f8        475          bne ob3015       ;64 didn't pick it up yet
      0a11 98           476          tya              ;set datout to a known state again (0)
      0a12 29 fd        477          and #$ff-dout15
      0a14 8d 1800      478   p10    sta pb15
      0a17 ca           479          dex              ;are there any more bits to send ?
      0a18 d0 dd        480          bne ob1015       ;yes, so start the bit handshake again
      0a1a 58           481          cli              ;ok, all bits sent so allow irq's again
      0a1b 68           482          pla              ;restore .y register
      0a1c a8           483          tay
      0a1d 68           484          pla              ;restore .x register
      0a1e aa           485          tax
      0a1f 60           486          rts              ;bye bye
                        487   ;
                        488   ;
                        489   ;==============================
                        490   ; this routine gets a byte of data
                        491   ;from the 64 using fast handshake #1
                        492   ;enter with .a=byte to send.
                        493   ;==============================
                        494   ;
      0a20 a9 01        495   ibyt15 lda #$01             ;put a flag bit into the data byte
      0a22 85 0b        496   b3     sta byte15           ;so we know when 8 bits have been sent
      0a24 8a           497          txa                  ;save the .x register
      0a25 48           498          pha
      0a26 98           499          tya                  ;save the .y register
      0a27 48           500          pha
      0a28 20 0539      501          jsr fram15-dcode+buf3 ;go do a frame handshake
                        502   ;
                        503   ;the following handshake is performed for each bit that is sent by the 64
                        504   ;
      0a2b              505   p11
      0a2b ad 1800      506   ib1015 lda pb15    ;wait for clkin=1 (64 has sent the data bit)
      0a2e a8           507          tay         ;save port value for later
      0a2f 29 04        508          and #cin15
      0a31 f0 f8        509          beq ib1015  ;64 hasn't set it yet
      0a33 98           510          tya         ;ok, get the bit back
```

```
error addr  code        seq   source statement

      0a34 4a           511         lsr a       ;move it to the carry
      0a35 26 0b        512   b4    rol byte15  ;and then into the data byte
                        513   ;
      0a37              514   p12
      0a37 ad 1800      515   ib2015 lda pb15   ;wait for the 64 to set clkin to 0 again
      0a3a 29 04        516         and #cin15  ;note, any flag bit in the carry is preserved
      0a3c d0 f9        517         bne ib2015  ;not done it yet
      0a3e 90 eb        518         bcc ib1015  ;flag bit didn't drop off yet so get another bit
      0a40 58           519         cli         ;ok, all bits fetched so allow irq's again
      0a41 68           520         pla         ;restore .y
      0a42 a8           521         tay
      0a43 68           522         pla         ;restore .x
      0a44 aa           523         tax
      0a45 a5 0b        524   b5    lda byte15  ;and return received byte in .a
      0a47 60           525         rts
                        526   ;
                        527   ;
                        528   ;=============================================================
                        529   ;  These are subroutines for starting reads on chained blocks.
                        530   ;call rfblok to read the first block into buf1 and call rnblok to
                        531   ;read the next block into the buffer that is not being used.  rnblok
                        532   ;takes its arguments from the current active buffer, if the first
                        533   ;byte is 0 then there is no block to chain to and nothing is done
                        534   ;=============================================================
                        535   ;
                        536   ;call rfblok with .x=track and .y=sector to be read
                        537   ;
      0a48 8e 0400      538   rfblok stx buf2     ;store the first track we want to read
      0a4b 8c 0401      539         sty buf2+1    ;and the first sector
      0a4e a9 01        540         lda #1        ;fool rnblok into thinking buf2 is active
      0a50 85 0f        541         sta joboff    ;drop through to the read next block routine
                        542   ;
                        543   ;this routine starts a block read into buf1 or buf2 depending on joboff
                        544   ;
      0a52 a5 0f        545   rnblok lda joboff   ;joboff tells us where the track and sector are
      0a54 85 0a        546   w1    sta wrkoff    ;save current value as a work pointer
      0a56 49 01        547         eor #1        ;make the other buffer the active one
      0a58 85 0f        548         sta joboff    ;joboff now points to buffer where block will go
      0a5a 18           549         clc           ;use old value of joboff as place to get t/s
      0a5b a5 0a        550   w2    lda wrkoff
      0a5d 69 03        551         adc #>buf1    ;use the result to modify some code
      0a5f 8d 05db      552         sta gt-dcode+buf3+2
      0a62 8d 05d6      553         sta gs-dcode+buf3+2
      0a65 a5 0f        554         lda joboff    ;compute index to the correct header
      0a67 0a           555         asl a
      0a68 aa           556         tax           ;to set up track and sector for reading
      0a69 ad 0301      557   gs    lda buf1+1    ;this gets modified - fetch the sector
      0a6c 95 07        558   h3    sta hdrs+1,x
      0a6e ad 0300      559   gt    lda buf1      ;this gets modified too - fetch the track
      0a71 95 06        560   h4    sta hdrs,x
      0a73 f0 06        561         beq rnb100    ;track was 0, so nothing else to read
      0a75 a6 0f        562         ldx joboff    ;now start a read into the correct buffer
      0a77 a9 80        563         lda #read
      0a79 95 00        564   j3    sta jobs,x
      0a7b 60           565   rnb100 rts          ;all done
                        566   ;
                        567   ;
```

```
error addr  code        seq   source statement

                        568   ;=================================================
                        569   ;this routine reads the name of the disk and sends all
                        570   ;program names to the 64.  The send format is as follows:
                        571   ;
                        572   ;1   the error code when reading 18,0 - exit if <> 1
                        573   ;2   16 bytes that make up the name of the disk
                        574   ;
                        575   ;3   track and sector where file starts on the disk,
                        576   ;      0 for track means directory read is complete, end.
                        577   ;4   send the name of the program (16 bytes)
                        578   ;5   go back to step 3
                        579   ;=================================================
                        580   ;
     0a7c a2 12         581   bdir   ldx #18              ;read 18,0 to find the disk name
     0a7e a0 00         582          ldy #0
     0a80 20 05b3       583          jsr rfblok-dcode+buf3
                        584   ;
     0a83 20 0648       585   bdir00 jsr snext-dcode+buf3  ;build pointer and start next read
     0a86 48            586          pha                  ;save the controller return code
     0a87 20 0557       587          jsr obyt15-dcode+buf3 ;send the code to the 64
     0a8a 68            588          pla                  ;get the code back
     0a8b c9 02         589          cmp #2               ;was our active block read ok ?
     0a8d 30 01         590          bmi bdir05           ;yes so proceed to read the disk name
     0a8f 60            591          rts                  ;error, just return to main command loop
                        592   ;
     0a90 a0 90         593   bdir05 ldy #144             ;index to the disk name
     0a92 b1 10         594   bdir10 lda (bufptr),y       ;get next character of the disk name
     0a94 20 0557       595          jsr obyt15-dcode+buf3 ;send it to the 64
     0a97 c8            596          iny
     0a98 c0 a0         597   bdir15 cpy #160             ;have we sent the whole name ?
     0a9a d0 f6         598          bne bdir10           ;nope, not yet
                        599   ;
                        600   ;ok, the disk name has been sent so find all program types and send
                        601   ;the names and start t/s to the 64 for user selection
                        602   ;
     0a9c 20 0648       603   bdir20 jsr snext-dcode+buf3  ;start the next block reading
     0a9f c9 02         604          cmp #2               ;did the current block read ok ?
     0aa1 30 04         605          bmi bdir25           ;yes
     0aa3 a9 00         606          lda #$00             ;error!, send 64 a zero byte and return
     0aa5 f0 33         607          beq bdir80           ;bra
                        608   ;
     0aa7 a9 02         609   bdir25 lda #$02         ;point to the first directory entry
     0aa9 8d 0780       610          sta lpb15        ;keep the index
     0aac ac 0780       611   bdir30 ldy lpb15        ;get current index to the directory entries
     0aaf c8            612          iny              ;point to its track/sector field
     0ab0 b1 10         613          lda (bufptr),y   ;if track = 0 then directory done
     0ab2 f0 26         614          beq bdir80       ;yep, send a zero byte and exit
     0ab4 88            615          dey              ;ok, a file is there so check if it's a prog
     0ab5 b1 10         616          lda (bufptr),y   ;get the type byte
     0ab7 c9 82         617          cmp #$82         ;is it a closed program file ?
     0ab9 d0 0e         618          bne bdir40       ;nope, go to the next entry
     0abb a9 12         619          lda #18          ;keep a byte counter
     0abd 85 0c         620   c1     sta bcnt15
                        621   ;
     0abf c8            622   bdir35 iny              ;ok, send the next 18 bytes (t/s,name)
     0ac0 b1 10         623          lda (bufptr),y
     0ac2 20 0557       624          jsr obyt15-dcode+buf3
```

```
error addr  code          seq   source statement

      0ac5 c6 0c          625   c2     dec bcnt15       ;have we sent it all ?
      0ac7 d0 f6          626          bne bdir35       ;nope
                          627   ;
      0ac9 ad 0780        628   bdir40 lda lpb15        ;point to the next directory entry
      0acc 18             629          clc
      0acd 69 20          630          adc #32
      0acf 8d 0780        631          sta lpb15
      0ad2 90 d8          632          bcc bdir30       ;haven't finished yet
                          633   ;
      0ad4 a0 00          634          ldy #$00         ;ok, finished this block, see if there's another
      0ad6 b1 10          635          lda (bufptr),y   ;if track link <> 0 then there is
      0ad8 d0 c2          636          bne bdir20       ;start another read and send next buffer
                          637   ;
      0ada 4c 0557        638   bdir80 jmp obyt15-dcode+buf3 ;send a zero byte to finish
                          639   ;
                          640   ;
                          641   ;=============================================
                          642   ; this routine waits for the job to finish on the
                          643   ; buffer we are going to read, and then starts
                          644   ; another read on the other buffer (buf1 or 2)
                          645   ;=============================================
                          646   ;
      0add a9 00          647   snext  lda #$00               ;build pointer to the current buffer
      0adf 85 10          648          sta bufptr
      0ae1 a5 0f          649          lda joboff
      0ae3 18             650          clc
      0ae4 69 03          651          adc #>buf1
      0ae6 85 11          652          sta bufptr+1
      0ae8 a6 0f          653          ldx joboff           ;wait for the current job to be finished
      0aea                654   j4
      0aea b5 00          655   snex00 lda jobs,x
      0aec 30 fc          656          bmi snex00
      0aee 48             657          pha                  ;save the returned error code
      0aef 20 05bd        658          jsr rnblok-dcode+buf3 ;start a read on the next block
      0af2 68             659          pla                  ;get back the error code
      0af3 60             660          rts                  ;all done
                          661   ;
                          662   ;=============================================
                          663   ; This routine does the burst load of a program file.
                          664   ;The start track and sector and start address of the
                          665   ;program are xfered using handshake #1.  The file
                          666   ;itself is xfered using the faster handshke #2
                          667   ;(except on the 1581!)
                          668   ;=============================================
                          669   ;
      0af4 20 058b        670   fload  jsr ibyt15-dcode+buf3 ;get the start track
      0af7 aa             671          tax
      0af8 20 058b        672          jsr ibyt15-dcode+buf3 ;get the start sector
      0afb a8             673          tay
      0afc 20 05b3        674          jsr rfblok-dcode+buf3  ;read the first block into buf1
      0aff 20 0648        675          jsr snext-dcode+buf3   ;and start the second block reading
      0b02 ad 0302        676          lda buf1+2             ;send the start address of the code
      0b05 20 0557        677          jsr obyt15-dcode+buf3
      0b08 ad 0303        678          lda buf1+3
      0b0b 20 0557        679          jsr obyt15-dcode+buf3
                          680   ;
      0b0e a0 04          681          ldy #4                 ;where to start the send from
```

```
error addr  code          seq   source statement

      0b10 a2 fc          682          ldx #252               ;number of bytes being sent
      0b12 d0 0e          683          bne flo20              ;send the block starting at .y index
                          684   ;
                          685   ;this is the main loop that sends a block of code to the 64
                          686   ;
      0b14 a0 00          687   flo10  ldy #$00               ;assume its a full block
      0b16 a2 fe          688          ldx #254
      0b18 b1 10          689          lda (bufptr),y         ;if track = 0 then it isnt
      0b1a d0 04          690          bne flo15              ;everything is ok
      0b1c c8             691          iny                    ;point to sector field for nmber of bytes to send
      0b1d b1 10          692          lda (bufptr),y
      0b1f aa             693          tax
      0b20 a0 02          694   flo15  ldy #$02
      0b22 8a             695   flo20  txa                    ;tell 64 how many bytes
      0b23 20 06ab        696   f1     jsr fstb15-dcode+buf3
                          697   ;
      0b26 b1 10          698   flo30  lda (bufptr),y         ;now send all the bytes of data
      0b28 20 06ab        699   f2     jsr fstb15-dcode+buf3
      0b2b c8             700          iny
      0b2c ca             701          dex
      0b2d d0 f7          702          bne flo30              ;more to send
                          703   ;
      0b2f a0 00          704          ldy #$00               ;see if there is another block to do
      0b31 b1 10          705          lda (bufptr),y
      0b33 f0 07          706          beq flo50              ;nope, all done so finish by sending 0
      0b35 20 0648        707          jsr snext-dcode+buf3 ;start the next block reading
      0b38 c9 02          708          cmp #2                 ;did the new block read ok ?
      0b3a 30 d8          709          bmi flo10              ;yes so get another block
                          710   ;
                          711   ;gets to here when the whole file has been sent to the 64 or an error
                          712   ;was encountered while reading program blocks
                          713   ;
      0b3c 68             714   flo50  pla                    ;scrap return address to main command loop
      0b3d 68             715          pla
      0b3e a9 00          716          lda #$00               ;send 64 a zero byte to terminate
                          717   ;
                          718   ;
                          719   ;================================
                          720   ; This is the fast i/o routine #2
                          721   ;that handshakes with screen off.
                          722   ;Not used by the 1581!
                          723   ;================================
                          724   ;
      0b40 85 0b          725   fstb15 sta byte15      ;save the byte to be sent
      0b42 8a             726          txa             ;save .x
      0b43 48             727          pha
      0b44 a2 08          728          ldx #$08        ;ke[ a count of bits to be sent
      0b46 78             729          sei             ;and don't allow interrupts now
                          730   ;
                          731
      0b47 ad 1800        732   fs1500 lda pb15        ;get current value of the port
      0b4a 06 0b          733          asl byte15      ;move the next data bit into the carry
      0b4c b0 05          734          bcs fs1510      ;nothing to do if bit is set (bus complements)
      0b4e 09 02          735          ora #dout15     ;send a 1 bit if 0 was required
      0b50 8d 1800        736          sta pb15
      0b53 09 08          737   fs1510 ora #cout15     ;set datout = 1 to say data available
      0b55 8d 1800        738          sta pb15
```

```
error addr  code          seq   source statement

      0b58 ea             739         nop             ;give the 64 time to find the data
      0b59 ea             740         nop             ;  this runs close....
      0b5a ea             741         nop             ;  1 more nop doesn't slow it down much
                          742   ;
      0b5b 29 f5          743         and #$ff-dout15-cout15   ;return port to known state
      0b5d 8d 1800        744         sta pb15
      0b60 ca             745         dex             ;any more bits ?
      0b61 d0 e4          746         bne fs1500      ;yes
      0b63 58             747         cli             ;irqs are ok now
      0b64 68             748         pla             ;restore .x
      0b65 aa             749         tax
      0b66 60             750         rts             ;all done
                          751   ;
                          752   ;
      0b67 00             753   dcend .byte 0 ;end of disk drive code
           =01d2          754   dclen = dcend-dcode
                          755   ;
           =0557          756   xoby15= obyt15-dcode+buf3
                          757   ;
                          758   ;
      0b68 a9 57          759   fixd81 lda #<xoby15  ;This is
      0b6a 8d 0b24        760          sta f1+1      ;part of the
      0b6d 8d 0b29        761          sta f2+1      ;messy 1581
      0b70 8d 0b41        762          sta fstb15+1  ;patching code!
      0b73 a9 05          763          lda #>xoby15
      0b75 8d 0b25        764          sta f1+2
      0b78 8d 0b2a        765          sta f2+2
      0b7b 8d 0b42        766          sta fstb15+2
      0b7e 60             767          rts
                          768   ;
                          769   ;=========================================
                          770   ;  This is the input routine for the 64 that
                          771   ;works with the screen and interrupts turned on.
                          772   ;=========================================
                          773   ;
                          774   ;
      0b7f a9 80          775   ibyt64 lda #$80         ;put a flag bit into the data byte we will fetch
      0b81 85 fb          776          sta byte64
      0b83 8a             777          txa             ;save .x
      0b84 48             778          pha
      0b85 98             779          tya             ;save .y
      0b86 48             780          pha
      0b87 20 0c54        781          jsr fram64      ;go do the byte handshake
                          782   ;
                          783   ;this section loops around a quick handshake to fetch 8 bits of data
                          784   ;
      0b8a ad dd00        785   ib1064 lda pb64        ;set clkout = 1 to say we want a bit now
      0b8d 09 10          786          ora #cout64
      0b8f 8d dd00        787          sta pb64
                          788   ;
      0b92 a2 07          789          ldx #$07        ;give the disk time to present the data
      0b94 ca             790   ib2064 dex             ;with this small delay loop
      0b95 d0 fd          791          bne ib2064
                          792   ;
      0b97 ad dd00        793          lda pb64        ;ok disk' data should be valid by now
      0b9a 29 ef          794          and #$ff-cout64 ;tell the disk we fetched it by setting clkout=0
      0b9c 8d dd00        795          sta pb64
```

```
error addr  code         seq   source statement

      0b9f 0a            796          asl a           ;move the data bit into the data byte
      0ba0 66 fb         797          ror byte64
      0ba2 90 e6         798          bcc ib1064      ;flag bit didn't drop out yet so get another bit
                         799    ;
      0ba4 68            800          pla          ;restore .y
      0ba5 a8            801          tay
      0ba6 68            802          pla          ;restore .x
      0ba7 aa            803          tax
      0ba8 a5 fb         804          lda byte64   ;fetch the assembled byte of data
      0baa 60            805          rts ;bye bye
                         806    ;
                         807    ;
                         808    ;=======================================================
                         809    ; This is the output routine that sends a byte of data to disk
                         810    ;using handshake #1 (screen and interrupts ON).  The 64 can
                         811    ;call the shots and depend on the 1541 to be waiting for
                         812    ;data at any given time.
                         813    ;=======================================================
                         814    ;
      0bab 85 fb         815    obyt64 sta byte64   ;save the data byte to be sent
      0bad 8a            816           txa          ;save .x
      0bae 48            817           pha
      0baf 98            818           tya          ;save .y
      0bb0 48            819           pha
      0bb1 a2 08         820           ldx #$08     ;keep a count of the bits to send
      0bb3 20 0c54       821           jsr fram64   ;do the frame handshake for this byte
                         822    ;
                         823    ;this is the loop to handshake each bit over to the 1541
                         824    ;
      0bb6 ad dd00       825    ob1064 lda pb64     ;get the current value of the port
      0bb9 06 fb         826           asl byte64   ;move the next data bit into the carry
      0bbb 90 05         827           bcc ob2064   ;nothing to do, the bit is clear
      0bbd 09 20         828           ora #dout64  ;it's a 1 bit that needs to be sent
      0bbf 8d dd00       829           sta pb64     ;this is a fix (1526 drops bits if 2 are changed)
      0bc2 09 10         830    ob2064 ora #cout64  ;set clkout to 1 to say data is there
      0bc4 8d dd00       831           sta pb64
                         832    ;
      0bc7 ea            833           nop          ;give the drive time to find the bit
      0bc8 ea            834           nop
      0bc9 ea            835           nop
      0bca ea            836           nop
                         837    ;
      0bcb 29 cf         838           and #$ff-dout64-cout64
      0bcd 8d dd00       839           sta pb64     ;set the port back to a known state
      0bd0 ca            840           dex          ;are there any more bits to send
      0bd1 d0 e3         841           bne ob1064   ;yes
      0bd3 68            842           pla          ;restore .y
      0bd4 a8            843           tay
      0bd5 68            844           pla          ;restore .x
      0bd6 aa            845           tax
      0bd7 60            846           rts          ;bye bye
                         847    ;===============================================
                         848    ; This is the real fast load routine that fetches
                         849    ;a file from the drive with handshake #2.  The 64 has
                         850    ;to watch the bus at all times, therefore it must have
                         851    ;the video chip turned off and interrupts disabled.
                         852    ;
```

```
error addr  code          seq   source statement

                          853   ; This code is moved into the memory at -bzcode- so
                          854   ;that it cannot be killed by programs loading over it
                          855   ;(most cases)
                          856   ;
                          857   ; NOTE: On the 1581, messy patches are made to this code
                          858   ;so that only handshake #1 is used.
                          859   ;
                          860   ;=============================================
                          861   ;
      0bd8 a5 41          862   sctop  lda $41     ;build a pointer to the code
      0bda 85 2d          863          sta vartab  ;use vartab so basic progs know where they finish
      0bdc a5 42          864          lda $42
      0bde 85 2e          865          sta vartab+1
                          866   ;
      0be0 20 ff95        867   sct00  jsr fstb64-sctop+bzcode    ;go get a super fast byte
      0be3 aa             868          tax         ;this is the number of bytes being sent
      0be4 f0 18          869          beq gotprg  ;0 means we are all done
      0be6 48             870          pha         ;save for later
      0be7 a0 00          871          ldy #$00
                          872   ;
      0be9 20 ff95        873   sct10  jsr fstb64-sctop+bzcode  ;get a byte of the code
      0bec 91 2d          874          sta (vartab),y          ;store it in memory
      0bee c8             875          iny
      0bef ca             876          dex                     ;any more bytes in this block ?
      0bf0 d0 f7          877          bne sct10               ;yes
                          878   ;
      0bf2 18             879          clc                     ;update the pointer
      0bf3 68             880          pla
      0bf4 65 2d          881          adc vartab
      0bf6 85 2d          882          sta vartab
      0bf8 90 e6          883          bcc sct00
      0bfa e6 2e          884          inc vartab+1
      0bfc d0 e2          885          bne sct00
                          886   ;
                          887   ;============================
                          888   ;gets to here when the code has
                          889   ;been loaded to see what to do
                          890   ;============================
                          891   ;
      0bfe c6 2d          892   gotprg dec vartab   ;fix top
      0c00 a5 2d          893          lda vartab   ;of text
      0c02 c9 ff          894          cmp #$ff     ;pointer
      0c04 d0 02          895          bne okvart
      0c06 c6 2e          896          dec vartab+1
      0c08 a9 00          897   okvart lda #$00     ;turn the sprites off
      0c0a 8d d015        898          sta $d015
      0c0d ad d011        899          lda 53265
      0c10 09 10          900          ora #16      ;turn the screen back on
      0c12 8d d011        901          sta 53265
                          902   ;
      0c15 a5 01          903          lda $01      ;get ready to bank
      0c17 09 03          904          ora #$03     ;ROMs back in
      0c19 a8             905          tay          ;and clr screen
      0c1a a9 93          906          lda #147
                          907   ;
      0c1c a6 41          908          ldx $41      ;if lo byte of load address=01
      0c1e e0 01          909          cpx #1       ;then assume it is BASIC,
```

```
error addr  code         seq   source statement

      0c20 d0 03         910          bne sysprg   ;so final bit of code
      0c22 4c 033c       911          jmp $033c    ;is in cassette buffer...
                         912   ;
      0c25 4c 003d       913   sysprg jmp $003d    ;else final bit is in zero page...
                         914   ;
                         915   ;=====================================
                         916   ; This routine does fast i/o handshake #2 with
                         917   ;the screen off.  If the drive is a 1581 this
                         918   ;routine is wiped out, and obyt64 (handshake #1)
                         919   ;is copied here instead.  What a mess!
                         920   ;=====================================
                         921   ;
      0c28 98            922   fstb64 tya           ;save .y
      0c29 48            923          pha
      0c2a a9 01         924          lda #$01      ;put a flag bit in the data byte
      0c2c 85 fb         925          sta byte64
                         926   ;
      0c2e ad dd00       927   fs6400 lda pb64      ;wait for clkin = 0 (disk sent data)
      0c31 a8            928          tay
      0c32 29 40         929          and #cin64
      0c34 d0 f8         930          bne fs6400    ;ok, bit isn't clear yet
                         931   ;
      0c36 98            932          tya
      0c37 0a            933          asl a         ;ok, we have the data bit
      0c38 26 fb         934          rol byte64    ;move it into the data byte
      0c3a 90 f2         935          bcc fs6400    ;more bits to fetch
                         936   ;
      0c3c 68            937          pla           ;restore .y
      0c3d a8            938          tay
      0c3e a5 fb         939          lda byte64    ;get back the data byte
      0c40 60            940          rts           ;all done
                         941   ;
      0c41 30 30 30      942          .byt '0000000000000000000'    ;filler for obyt15
      0c44 30 30 30
      0c47 30 30 30
      0c4a 30 30 30
      0c4d 30 30 30
      0c50 30 30 30
      0c53 30
                         943   ;
                         944   ;================================================
                         945   ; This is the frame handshake that starts xfer of 8 bits.
                         946   ;It is called by the 64's fast i/o routines to make sure
                         947   ;the code in the drive and the 64 is in sync. It has been
                         948   ;positioned here so that it is moved to the bzcode area
                         949   ;along with the rest of the fast loader.
                         950   ;================================================
                         951   ;
      0c54 ad dd00       952   fram64 lda pb64      ;set clkout = 1 to start the handshake
      0c57 09 10         953          ora #cout64
      0c59 8d dd00       954          sta pb64
                         955   ;
      0c5c ad dd00       956   fr0064 lda pb64      ;wait for the drive to set datin = 0
      0c5f a8            957          tay           ;save the current port value for later
      0c60 29 80         958          and #din64
      0c62 d0 f8         959          bne fr0064    ;drive didn't respond yet
                         960   ;
```

```
error addr  code          seq   source statement

      0c64 98            961          tya             ;set clkout = 0 to set lines to 0 state
      0c65 29 ef         962          and #$ff-cout64 ;and let the drive know we are ready
      0c67 8d dd00       963          sta pb64
      0c6a ad dd00       964   fr1064 lda pb64        ;finally wait for the drive to set datin=1
      0c6d 29 80         965          and #din64
      0c6f f0 f9         966          beq fr1064      ;drive didn't respond yet
      0c71 60            967          rts             ;ok, frame handshake is done
                         968   ;
      0c72 00            969   scend .byte 0
                         970   ;
                         971   ;=======================================
                         972   ;this section displays the menu and gets the
                         973   ;users selection from that menu.  Sprites are
                         974   ;downloaded to $3000
                         975   ;=======================================
                         976   ;
      0c73 18 34 30      977   postbl .byte 24,52,48,52,72,52
      0c76 34 48 34
                         978   ;
      0c79 a9 00         979   menu   lda #0      ;screen background black
      0c7b 8d d021       980          sta 53281
      0c7e a9 0d         981          lda #13     ;border light green
      0c80 8d d020       982          sta 53280
      0c83 a9 93         983          lda #147
      0c85 20 ffd2       984          jsr chrout  ;clear the screen
                         985   ;
      0c88 a2 00         986          ldx #0      ;start at line 0
      0c8a a0 03         987          ldy #3      ;and do 3 reversed lines
      0c8c a9 0d         988          lda #13     ;in light green on the screen
      0c8e 20 0fa5       989          jsr doline
                         990   ;
      0c91 a2 04         991          ldx #4      ;now start at line 4
      0c93 a0 03         992          ldy #3      ;and do 3 reversed lines
      0c95 a9 01         993          lda #1      ;in white
      0c97 20 0fa5       994          jsr doline
                         995   ;
      0c9a a2 00         996          ldx #$00    ;index into sprite data
      0c9c bd 10e5       997   menu10 lda titlez,x ;download sprite data to $3000
      0c9f 9d 3000       998          sta $3000,x
      0ca2 e8            999          inx
      0ca3 e0 c0        1000          cpx #192    ;have we downloaded all of the data ?
      0ca5 d0 f5        1001          bne menu10  ;nope, not yet
                        1002   ;
      0ca7 a2 03        1003   menu20 ldx #$03    ;color all three sprites
      0ca9 8a           1004   menu25 txa         ;in white,red and cyan
      0caa 9d d026      1005          sta $d026,x
      0cad ca           1006          dex
      0cae d0 f9        1007          bne menu25
                        1008   ;
      0cb0 8e d025      1009          stx $d025   ;mob multicolor 0 = black
      0cb3 8e d017      1010          stx $d017   ;no y expansion
      0cb6 8e d01d      1011          stx $d01d   ;no x expansion
                        1012   ;
      0cb9 a2 c0        1013          ldx #192    ;build pointers to each sprite in memory
      0cbb 8e 07f8      1014          stx 2040
      0cbe e8           1015          inx
      0cbf 8e 07f9      1016          stx 2041
```

```
error addr  code         seq   source statement

      0cc2 e8          1017          inx
      0cc3 8e 07fa     1018          stx 2042
                       1019    ;
      0cc6 a2 05       1020          ldx #$05     ;now give each sprite its x,y position
      0cc8 bd 0c73     1021    menu35 lda postbl,x
      0ccb 9d d000     1022          sta $d000,x
      0cce ca          1023          dex
      0ccf 10 f7       1024          bpl menu35
                       1025    ;
      0cd1 a2 07       1026          ldx #$07     ;turn on the 3 used sprites
      0cd3 8e d01c     1027          stx $d01c    ;multicolor
      0cd6 8e d015     1028          stx $d015    ;enable
                       1029    ;
      0cd9 a9 00       1030          lda #0       ;put message 0 on the screen
      0cdb 20 0fbf     1031          jsr mesage   ;'15x1 fast load'
                       1032    ;
                       1033    ;====================================
                       1034    ;menu screen is all set up now, so check
                       1035    ;that there is a disk in the drive and
                       1036    ;ask the user for one if there isn't.
                       1037    ;====================================
                       1038    ;
      0cde a9 01       1039    menu40 lda #$01     ;tell the drive to do a seek
      0ce0 20 0bab     1040           jsr obyt64
      0ce3 20 0b7f     1041           jsr ibyt64   ;get the error return
      0ce6 c9 02       1042           cmp #2       ;is there a disk in there
      0ce8 30 0d       1043           bmi menu45   ;yes
                       1044    ;
      0cea a9 01       1045          lda #1       ;nope, so ask for a disk
      0cec 20 0fbf     1046          jsr mesage
      0cef 20 ffe4     1047    waitdk jsr getin
      0cf2 f0 fb       1048          beq waitdk
      0cf4 4c 0cde     1049          jmp menu40   ;and keep seeking until a disk is there
                       1050    ;
      0cf7 a9 01       1051    menu45 lda #$01     ;do one more seek because removing a disk...
      0cf9 20 0bab     1052          jsr obyt64   ;can make seek return a 1 when no disk is there
      0cfc 20 0b7f     1053          jsr ibyt64
      0cff c9 02       1054          cmp #2
      0d01 10 db       1055          bpl menu40   ;loop until disk in
      0d03 a9 02       1056          lda #2       ;put up the disk name message
      0d05 20 0fbf     1057          jsr mesage
                       1058    ;
      0d08 a9 02       1059          lda #2       ;tell drive to do a directory send
      0d0a 20 0bab     1060          jsr obyt64
      0d0d 20 0b7f     1061          jsr ibyt64   ;get the status code for the 18,0 read
      0d10 c9 02       1062          cmp #2       ;if 1 then ok to carry on and get the name
      0d12 30 08       1063          bmi menu50   ;no problems
                       1064    ;
      0d14 a9 00       1065          lda #$00     ;error for now
      0d16 8d d015     1066          sta $d015
      0d19 4c 0bab     1067          jmp obyt64   ;turn off the drive code
                       1068    ;
      0d1c a2 05       1069    menu50 ldx #5       ;build a pointer to the disk name field
      0d1e 20 0f59     1070           jsr setptr
      0d21 a0 16       1071           ldy #22      ;where to store the stuff
      0d23 a2 10       1072           ldx #16      ;we are going to get 16 bytes now
                       1073    ;
```

```
error addr  code          seq   source statement

      0d25 20 0b7f      1074   menu55 jsr ibyt64   ;get a byte from the drive
      0d28 29 3f        1075          and #63      ;convert character to reverse screen code
      0d2a 09 80        1076          ora #128
      0d2c 91 6c        1077          sta (vptr),y ;store the byte on the screen
      0d2e c8           1078          iny
      0d2f ca           1079          dex          ;have we fetched all bytes yet ?
      0d30 d0 f3        1080          bne menu55   ;nope
                        1081   ;
                        1082   ;=========================================
                        1083   ;ok, we now have the disk name so fetch track,
                        1084   ;sector,name for each file that the drive sends
                        1085   ;us.  A track number of zero means all files done
                        1086   ;=========================================
                        1087   ;
      0d32 20 0f44      1088          jsr setfp    ;build a pointer to dir storage area
      0d35 a9 00        1089          lda #0       ;keep a count of programs
      0d37 8d 02a7      1090          sta prgcnt
                        1091   ;
      0d3a a0 00        1092   menu60 ldy #$00     ;set index into storage
      0d3c 20 0b7f      1093          jsr ibyt64   ;get the track number
      0d3f f0 1e        1094          beq gotdir   ;0, so all is done
      0d41 91 fe        1095          sta (ptr),y  ;remember the track number
      0d43 c8           1096          iny
      0d44 20 0b7f      1097          jsr ibyt64   ;fetch the sector number and store it
      0d47 91 fe        1098          sta (ptr),y
      0d49 c8           1099          iny          ;now point to the name storage
      0d4a 20 0b7f      1100   menu65 jsr ibyt64   ;fetch a byte of the filename
      0d4d 29 3f        1101          and #63      ;convert to display code
      0d4f 91 fe        1102          sta (ptr),y  ;and store in memory
      0d51 c8           1103          iny
      0d52 c0 12        1104          cpy #18
      0d54 d0 f4        1105          bne menu65
      0d56 ee 02a7      1106          inc prgcnt   ;update number of programs there
      0d59 20 0f4d      1107          jsr nxtfil   ;move pointer on by 18 bytes
      0d5c 4c 0d3a      1108          jmp menu60   ;bra
                        1109   ;
                        1110   ;=========================================
                        1111   ;all the directory entries are in memory now,
                        1112   ;so set up the display and allow the user to
                        1113   ;cursor around to select a file for loading
                        1114   ;=========================================
                        1115   ;
      0d5f ad 02a7      1116   gotdir lda prgcnt  ;were there any files at all ?
      0d62 d0 18        1117          bne gotd00  ;yes so let user choose
                        1118   ;
      0d64 a2 08        1119          ldx #8      ;error, no programs on the disk
      0d66 a0 05        1120          ldy #5      ;display this in a 5 line reverse box
      0d68 a9 01        1121          lda #1
      0d6a 20 0fa5      1122          jsr doline
                        1123   ;
      0d6d a9 04        1124          lda #4      ;use messages 4 and 5
      0d6f 20 0fbf      1125          jsr mesage
                        1126   ;
      0d72 20 ffe4      1127   noprog jsr getin   ;wait for stop or return
      0d75 c9 0d        1128          cmp #13     ;was it return to change disks ?
      0d77 d0 f9        1129          bne noprog  ;nope
      0d79 4c 0c79      1130          jmp menu    ;redraw the menu and start again
```

```
error addr  code          seq   source statement

                          1131  ;
      0d7c a9 00          1132  gotd00 lda #$00    ;set up to display files for the first time
      0d7e 8d 02a9        1133         sta fprg    ;first program on the screen
      0d81 8d 02a8        1134         sta curprg  ;program number the cursor is over
      0d84 8d 02aa        1135         sta curlin  ;current line where the cursor is
                          1136  ;
      0d87 a2 15          1137         ldx #21     ;put a 3 line deep reverse bar at screen bottom
      0d89 a0 03          1138         ldy #3
      0d8b a9 01          1139         lda #1      ;make it white
      0d8d 20 0fa5        1140         jsr doline
      0d90 a9 03          1141         lda #3      ;display the instructions on it
      0d92 20 0fbf        1142         jsr mesage
      0d95 a9 05          1143         lda #5
      0d97 20 0fbf        1144         jsr mesage
                          1145  ;
      0d9a 20 0ec5        1146  dspm00 jsr dspfil  ;display as many files as possible
      0d9d 20 ffe4        1147  dspm10 jsr getin   ;go get a key from the user
      0da0 f0 fb          1148         beq dspm10
      0da2 c9 03          1149         cmp #3      ;was it stop
      0da4 d0 03          1150         bne dspm15  ;nope
      0da6 4c 0c79        1151         jmp menu    ;start over if stop was pressed
                          1152  ;
      0da9 c9 11          1153  dspm15 cmp #17     ;was it cursor down ?
      0dab d0 22          1154         bne dspm20  ;nope
      0dad ae 02a8        1155         ldx curprg  ;first check if we would go past the end by moving
      0db0 e8             1156         inx
      0db1 e8             1157         inx
      0db2 ec 02a7        1158         cpx prgcnt
      0db5 b0 e6          1159         bcs dspm10  ;it was so don't do anything
      0db7 8e 02a8        1160         stx curprg  ;still ok
      0dba ae 02aa        1161  dspm16 ldx curlin  ;move down a line
      0dbd e8             1162         inx
      0dbe e0 0c          1163         cpx #12     ;unless we were on the last one
      0dc0 d0 07          1164         bne dspm17  ;we aren't so just increment and carry on
      0dc2 ee 02a9        1165         inc fprg    ;time to scroll so first prog = first+2
      0dc5 ee 02a9        1166         inc fprg
      0dc8 ca             1167         dex         ;correct .x because it went too far
      0dc9 8e 02aa        1168  dspm17 stx curlin
      0dcc 4c 0d9a        1169         jmp dspm00  ;redisplay the screen full of files
                          1170  ;
      0dcf c9 91          1171  dspm20 cmp #145    ; was it cursor up ?
      0dd1 d0 20          1172         bne dspm30  ;nope
      0dd3 ae 02a8        1173         ldx curprg  ;only move up if prog # is 2 or more
      0dd6 e0 02          1174         cpx #2
      0dd8 90 c3          1175         bcc dspm10  ;nothing to do we are on the top line
      0dda ca             1176         dex         ;move back by 2
      0ddb ca             1177         dex
      0ddc 8e 02a8        1178         stx curprg
      0ddf ad 02aa        1179         lda curlin  ;move current line back unless we are on the top
      0de2 d0 09          1180         bne dspm25  ;its ok, we are not on the top line
      0de4 ce 02a9        1181         dec fprg    ;first = first -2
      0de7 ce 02a9        1182         dec fprg
      0dea 4c 0d9a        1183         jmp dspm00  ;display the new files that moved in
      0ded ce 02aa        1184  dspm25 dec curlin  ;ok to just move the cursor up the screen
      0df0 4c 0d9a        1185         jmp dspm00
                          1186  ;
      0df3 c9 1d          1187  dspm30 cmp #29     ;was it cursor forward
```

```
error addr  code          seq   source statement

      0df5 d0 14        1188          bne dspm40  ;nope
      0df7 ad 02a8      1189          lda curprg  ;move to the next if curprg<prgcnt and even
      0dfa aa           1190          tax
      0dfb e8           1191          inx
      0dfc ec 02a7      1192          cpx prgcnt
      0dff f0 9c        1193          beq dspm10  ;already at the maximum
      0e01 29 01        1194          and #$01
      0e03 d0 98        1195          bne dspm10  ;already in the second column
      0e05 8e 02a8      1196          stx curprg  ;store the new position and redisplay
      0e08 4c 0d9a      1197          jmp dspm00
                        1198    ;
      0e0b c9 9d        1199    dspm40 cmp #157    ;was it cursor back ?
      0e0d d0 0b        1200          bne dspm52  ;nope
      0e0f ad 02a8      1201          lda curprg  ;make number even
      0e12 29 fe        1202          and #$fe
      0e14 8d 02a8      1203          sta curprg
      0e17 4c 0d9a      1204          jmp dspm00  ;redisplay the files
                        1205    ;
      0e1a c9 44        1206    dspm52 cmp #68     ;was it 'd'
      0e1c d0 03        1207          bne dspm53  ;nope
      0e1e 4c 0c79      1208          jmp menu    ;else new disk
                        1209    ;
      0e21 c9 55        1210    dspm53 cmp #85    ;was it 'u'
      0e23 d0 08        1211          bne dspm54 ;nope
      0e25 a9 00        1212          lda #$00   ;
      0e27 20 0bab      1213          jsr obyt64 ;else restore DOS and
      0e2a 4c 0814      1214          jmp reopen ;open next unit no.
                        1215    ;
      0e2d c9 0d        1216    dspm54 cmp #13     ;was it return to load ?
      0e2f f0 1b        1217          beq dspm55  ;yes, so !do it!
      0e31 4c 0d9d      1218          jmp dspm10  ;nope, so just get another key
                        1219    ;
                        1220    ;
                        1221    ;=========================================
                        1222    ;Gets to here when the user has chosen a program.
                        1223    ;The loading code gets moved to -bzcode- and
                        1224    ;the final bits get moved to the cassette buff
                        1225    ;and z-page.
                        1226    ;=========================================
                        1227    ;
          =ffc1         1228    xfram  =fram64-sctop+bzcode
                        1229    ;
      0e34              1230    bascod
      0e34 84 01        1231          sty $01    ; This is the final bit
      0e36 58           1232          cli        ;for BASIC programs.
      0e37 20 ffd2      1233          jsr chrout ; It gets moved to the
      0e3a 20 a659      1234          jsr $a659  ;cassette buffer...
      0e3d 20 a533      1235          jsr $a533
      0e40 4c a7ae      1236          jmp $a7ae
      0e43              1237    endbas
                        1238    ;
      0e43              1239    execut
      0e43 84 01        1240          sty $01       ;3a   This is the final bit
      0e45 58           1241          cli           ;3c   for ML programs.  It
      0e46 20 ffd2      1242          jsr chrout    ;3d   gets moved to zero page...
      0e49 4c 0000      1243    notbas jmp $0000    ;40 lo-41 hi-42
      0e4c              1244    endxeq
```

```
error addr  code          seq   source statement

                          1245   ;
                          1246   ;
      0e4c 78             1247   dspm55 sei            ;move #1 / move loader to -bzcode-
      0e4d a5 01          1248          lda $01        ;
      0e4f 29 fd          1249          and #$fd       ; Kill kernal,basic,i/o
      0e51 85 01          1250          sta $01
      0e53 a2 9a          1251          ldx #scend-sctop
      0e55 bd 0bd8        1252   dspm56 lda sctop,x
      0e58 9d ff45        1253          sta bzcode,x
      0e5b ca             1254          dex
      0e5c e0 ff          1255          cpx #$ff
      0e5e d0 f5          1256          bne dspm56
                          1257   ;
      0e60 a2 09          1258          ldx #endxeq-execut ;move #2  /move ML execute
      0e62 bd 0e43        1259   dspm57 lda execut,x       ;code into the z-page area
      0e65 95 3a          1260          sta $003a,x
      0e67 ca             1261          dex
      0e68 10 f8          1262          bpl dspm57
                          1263   ;
      0e6a a2 0f          1264          ldx #endbas-bascod ;move #3  /move basic run
      0e6c bd 0e34        1265   dspm58 lda bascod,x       ;code into the cassette buff
      0e6f 9d 033c        1266          sta $033c,x
      0e72 ca             1267          dex
      0e73 10 f7          1268          bpl dspm58
                          1269   ;
      0e75 ad 02af        1270          lda drtype   ;Which drive?
      0e78 c9 ff          1271          cmp #$ff     ;ff=1581
      0e7a d0 0d          1272          bne okblnk
                          1273   ;
      0e7c a9 c1          1274          lda #<xfram
      0e7e 8d ff9e        1275          sta fstb64-sctop+bzcode+9  ;This is part
      0e81 a9 ff          1276          lda #>xfram                ;of the messy
      0e83 8d ff9f        1277          sta fstb64-sctop+bzcode+10 ;1581 patch job!
      0e86 4c 0e96        1278          jmp noblnk
                          1279   ;
      0e89                1280   okblnk
      0e89 ad d011        1281          lda 53265    ;turn the screen off
      0e8c 29 ef          1282          and #255-16
      0e8e 8d d011        1283          sta 53265
      0e91 a9 00          1284          lda #$00      ;turn the sprites off
      0e93 8d d015        1285          sta $d015
      0e96                1286   noblnk
      0e96 a9 03          1287          lda #3        ;tell the drive to burst load
      0e98 20 0bab        1288          jsr obyt64
      0e9b ad 02a8        1289          lda curprg    ;find out which program to load
      0e9e 20 0f34        1290          jsr fndprg
      0ea1 a0 00          1291          ldy #$00      ;tell burst load where to start from
      0ea3 b1 fe          1292          lda (ptr),y   ;track
      0ea5 20 0bab        1293          jsr obyt64
      0ea8 c8             1294          iny
      0ea9 b1 fe          1295          lda (ptr),y   ;sector
      0eab 20 0bab        1296          jsr obyt64
      0eae 20 0b7f        1297          jsr ibyt64    ;get the start address of this program
      0eb1 85 41          1298          sta $41       ;save it for the fast loader routine
      0eb3 20 0b7f        1299          jsr ibyt64
      0eb6 85 42          1300          sta $42
      0eb8 a5 41          1301          lda $41
```

```
error addr  code        seq   source statement

      0eba c9 01       1302         cmp #$01     ;if lo byte of load address=01
      0ebc d0 04       1303         bne mlcode   ;then assume it is CBM basic
      0ebe a9 08       1304   cbmbas lda #$08     ;and load into $0801
      0ec0 85 42       1305         sta $42
      0ec2 4c ff45     1306   mlcode jmp bzcode   ;call the zap part of zaploader
                       1307   ;
                       1308   ;=======================================
                       1309   ;routine to display files on the screen
                       1310   ;
                       1311   ;prgcnt    number of programs in memory
                       1312   ;curprg    program number cursor is currently on
                       1313   ;curlin    line number that the cursor is on (0-11)
                       1314   ;fprg      first program displayed on the screen
                       1315   ;=======================================
                       1316   ;
      0ec5 a2 08       1317   dspfil ldx #8          ;point to the first line
      0ec7 20 0f59     1318          jsr setptr      ;where the menu will be displayed
      0eca a9 00       1319          lda #$00        ;we are on line 0 now
      0ecc 8d 02ab     1320          sta wrklin
      0ecf ad 02a9     1321          lda fprg        ;which program do we start at
      0ed2 8d 02ac     1322          sta wrkprg
      0ed5 20 0f34     1323          jsr fndprg      ;build a pointer (in ptr) to the first program
                       1324   ;
      0ed8 a2 00       1325   dspf00 ldx #$00        ;assume current prog is not the same as the...
      0eda ad 02ac     1326          lda wrkprg      ;one the cursor is over
      0edd cd 02a8     1327          cmp curprg
      0ee0 d0 02       1328          bne dspf10
      0ee2 a2 80       1329          ldx #$80        ;it is the same so display it reversed
      0ee4 86 65       1330   dspf10 stx temp        ;temp contains the reverse flag
      0ee6 a0 02       1331          ldy #$02        ;point to the name and it's screen destination
      0ee8 b1 fe       1332   dspf15 lda (ptr),y
      0eea 05 65       1333          ora temp
      0eec 91 6c       1334          sta (vptr),y
      0eee a9 0f       1335          lda #15         ;display names in light grey
      0ef0 91 62       1336          sta (cptr),y
      0ef2 c8          1337          iny
      0ef3 c0 12       1338          cpy #18
      0ef5 d0 f1       1339          bne dspf15
                       1340   ;
      0ef7 18          1341          clc             ;move screen pointers to next column
      0ef8 a5 6c       1342          lda vptr
      0efa 69 14       1343          adc #20
      0efc 85 6c       1344          sta vptr
      0efe 85 62       1345          sta cptr
      0f00 90 04       1346          bcc dspf20
      0f02 e6 6d       1347          inc vptr+1
      0f04 e6 63       1348          inc cptr+1
                       1349   ;
      0f06 20 0f4d     1350   dspf20 jsr nxtfil      ;move filename pointer to the next entry
                       1351   ;
      0f09 ae 02ac     1352   dspf25 ldx wrkprg      ;have we finished ?
      0f0c e8          1353          inx
      0f0d ec 02a7     1354          cpx prgcnt
      0f10 f0 13       1355          beq dspf50      ;yes so see if we have to erase 2nd column
      0f12 8e 02ac     1356          stx wrkprg
      0f15 8a          1357          txa             ;if program number is even then move down a line
      0f1c 29 01       1358          and #$01
```

```
error addr  code         seq   source statement

      0f18 d0 be        1359          bne dspf00      ;nope, we are on the same line
      0f1a ee 02ab      1360          inc wrklin      ;do we have room to display any more
      0f1d ad 02ab      1361          lda wrklin
      0f20 c9 0c        1362          cmp #12
      0f22 d0 b4        1363          bne dspf00      ;yes keep displaying the programs
      0f24 60           1364    dspf30 rts            ;out of room, so finish
                        1365    ;
      0f25 8a           1366    dspf50 txa            ;if we finished in column 2 then whats there
      0f26 29 01        1367          and #$01        ;now will have to be erased (no file here now)
      0f28 f0 fa        1368          beq dspf30      ;nope, we finished in column 1
      0f2a a0 12        1369          ldy #18         ;ok, we have to blank out the last line
      0f2c a9 20        1370          lda #32
      0f2e 91 6c        1371    dspf55 sta (vptr),y
      0f30 88           1372          dey
      0f31 d0 fb        1373          bne dspf55
      0f33 60           1374          rts
                        1375    ;
                        1376    ;==============================
                        1377    ;this routine builds a pointer to
                        1378    ;the program entry number in .a
                        1379    ;the pointer is left in (ptr) for
                        1380    ;the caller to use as an indirect
                        1381    ;==============================
                        1382    ;
      0f34 aa           1383    fndprg tax            ;keep a counter
      0f35 20 0f44      1384          jsr setfp       ;build pointer to first entry
      0f38 e0 00        1385          cpx #0          ;is the pointer correct now ?
      0f3a d0 01        1386          bne fndp00      ;nope
      0f3c 60           1387          rts             ;pointer done
                        1388    ;
      0f3d 20 0f4d      1389    fndp00 jsr nxtfil     ;add 18 to ptr for each filename entry
      0f40 ca           1390          dex             ;still more to do ?
      0f41 d0 fa        1391          bne fndp00      ;yes
      0f43 60           1392          rts
                        1393    ;
                        1394    ;
                        1395    ;build a pointer to the filename area in memory
                        1396    ;
      0f44 a9 a5        1397    setfp  lda #<dirstr
      0f46 85 fe        1398          sta ptr
      0f48 a9 11        1399          lda #>dirstr
      0f4a 85 ff        1400          sta ptr+1
      0f4c 60           1401          rts
                        1402    ;
                        1403    ;
                        1404    ;move ptr to the next file entry (by adding 18)
                        1405    ;
      0f4d 18           1406    nxtfil clc
      0f4e a5 fe        1407          lda ptr
      0f50 69 12        1408          adc #18
      0f52 85 fe        1409          sta ptr
      0f54 90 02        1410          bcc nxtf00
      0f56 e6 ff        1411          inc ptr+1
      0f58 60           1412    nxtf00 rts
                        1413    ;
                        1414    ;============================
                        1415    ;these are general purpose
```

```
error addr  code          seq   source statement

                          1416  ;screen and message routines
                          1417  ;=========================
                          1418  ;
                          1419  ;this routine sets a pointer in vptr and cptr to the screen
                          1420  ;line sent in .x
                          1421  ;
      0f59 48             1422  setptr pha
      0f5a 8a             1423         txa
      0f5b 48             1424         pha
      0f5c 0a             1425         asl a
      0f5d aa             1426         tax
      0f5e bd 0f73        1427         lda scrlin,x
      0f61 85 6c          1428         sta vptr
      0f63 85 62          1429         sta cptr       ;color and screen lo bytes are the same
      0f65 bd 0f74        1430         lda scrlin+1,x
      0f68 85 6d          1431         sta vptr+1
      0f6a 18             1432         clc
      0f6b 69 d4          1433         adc #>54272    ;offset to the color ram
      0f6d 85 63          1434         sta cptr+1
      0f6f 68             1435         pla
      0f70 aa             1436         tax
      0f71 68             1437         pla
      0f72 60             1438         rts
                          1439  ;
      0f73 0400 0428      1440  scrlin .word 1024,1064,1104,1144,1184
      0f77 0450 0478
      0f7b 04a0
      0f7d 04c8 04f0      1441         .word 1224,1264,1304,1344,1384
      0f81 0518 0540
      0f85 0568
      0f87 0590 05b8      1442         .word 1424,1464,1504,1544,1584
      0f8b 05e0 0608
      0f8f 0630
      0f91 0658 0680      1443         .word 1624,1664,1704,1744,1784
      0f95 06a8 06d0
      0f99 06f8
      0f9b 0720 0748      1444         .word 1824,1864,1904,1944,1984
      0f9f 0770 0798
      0fa3 07c0
                          1445  ;
                          1446  ;
                          1447  ;================================
                          1448  ;this routine puts reverse spaces on
                          1449  ;.y lines of the screen
                          1450  ;starting at line .x in color .a
                          1451  ;================================
                          1452  ;
      0fa5 85 64          1453  doline sta color    ;save the color to do
      0fa7 84 65          1454         sty temp     ;save the line count
      0fa9 20 0f59        1455  doli00 jsr setptr   ;build a pointer to the line in .x
      0fac a0 27          1456         ldy #39
      0fae a5 64          1457  doli10 lda color    ;store the color
      0fb0 91 62          1458         sta (cptr),y
      0fb2 a9 a0          1459         lda #$a0     ;store a reverse space
      0fb4 91 6c          1460         sta (vptr),y
      0fb6 88             1461         dey
      0fb7 10 f5          1462         bpl doli10
```

```
error addr  code         seq   source statement

      0fb9 e8          1463          inx           ;point .x to the next line
      0fba c6 65       1464          dec temp      ;any more to do ?
      0fbc d0 eb       1465          bne doli00    ;yes
      0fbe 60          1466          rts           ;all done
                       1467    ;
                       1468    ;=============================================
                       1469    ;call this routine to display message .a on the
                       1470    ;screen at the co-ordinates that are stored with
                       1471    ;the message.  The text is displayed in reverse video.
                       1472    ;=============================================
                       1473    ;
      0fbf 0a          1474    mesage asl a          ;access table of message addresses
      0fc0 aa          1475           tax
      0fc1 bd 0ffa     1476           lda msgtbl,x
      0fc4 85 fe       1477           sta ptr        ;build a pointer to the desired message
      0fc6 bd 0ffb     1478           lda msgtbl+1,x
      0fc9 85 ff       1479           sta ptr+1
      0fcb a0 00       1480           ldy #$00       ;find out which line to store the message on
      0fcd b1 fe       1481           lda (ptr),y
      0fcf aa          1482           tax
      0fd0 20 0f59     1483           jsr setptr
      0fd3 c8          1484           iny            ;make pointer absolute (only screen ptr
      0fd4 b1 fe       1485           lda (ptr),y    ;is used because color has been done already)
      0fd6 18          1486           clc
      0fd7 65 6c       1487           adc vptr
      0fd9 85 6c       1488           sta vptr
      0fdb 90 02       1489           bcc mesa00
      0fdd e6 6d       1490           inc vptr+1
                       1491    ;
      0fdf a5 fe       1492    mesa00 lda ptr      ;move pointer to text so that it points
      0fe1 18          1493           clc          ;to the text string with index y = 0
      0fe2 69 02       1494           adc #2
      0fe4 85 fe       1495           sta ptr
      0fe6 90 02       1496           bcc mesa10
      0fe8 e6 ff       1497           inc ptr+1
      0fea a0 00       1498    mesa10 ldy #$00     ;ok, pointers are set up
                       1499    ;
      0fec b1 fe       1500    mesa15 lda (ptr),y  ;get a byte of the text
      0fee f0 09       1501           beq mesa20   ;0 byte terminates it
      0ff0 29 3f       1502           and #63      ;convert to reverse screen code
      0ff2 09 80       1503           ora #128
      0ff4 91 6c       1504           sta (vptr),y
      0ff6 c8          1505           iny
      0ff7 d0 f3       1506           bne mesa15
      0ff9 60          1507    mesa20 rts          ;all done
                       1508    ;
      0ffa 1006 1019   1509    msgtbl .word msg1,msg2,msg3,msg4,msg5,msg6
      0ffe 1044 106f
      1002 10c2 1099
                       1510    ;
      1006 01 0c 31    1511    msg1   .byt 1,12,'15
      1009 35
      100a 34 31 20    1512    drmssg .byt '41 fast loader ',0
      100d 46 41 53
      1010 54 20 4c
      1013 4f 41 44
      1016 45 52 00
```

```
error addr  code          seq   source statement

      1019 05 00 50     1513   msg2   .byt 5,0,'please put your program disk in the 15'
      101c 4c 45 41
      101f 53 45 20
      1022 50 55 54
      1025 20 59 4f
      1028 55 52 20
      102b 50 52 4f
      102e 47 52 41
      1031 4d 20 44
      1034 49 53 4b
      1037 20 49 4e
      103a 20 54 48
      103d 45 20 31
      1040 35
      1041 34 31 00     1514   drmsg2 .byt '41',0
      1044 05 00 50     1515   msg3   .byt 5,0,'programs on the disk "               " ',0
      1047 52 4f 47
      104a 52 41 4d
      104d 53 20 4f
      1050 4e 20 54
      1053 48 45 20
      1056 44 49 53
      1059 4b 20 22
      105c 20 20 20
      105f 20 20 20
      1062 20 20 20
      1065 20 20 20
      1068 20 20 20
      106b 20 22 20
      106e 00
      106f 16 00 55     1516   msg4   .byt 22,0,'use cursor to select and return to load ',0
      1072 53 45 20
      1075 43 55 52
      1078 53 4f 52
      107b 20 54 4f
      107e 20 53 45
      1081 4c 45 43
      1084 54 20 41
      1087 4e 44 20
      108a 52 45 54
      108d 55 52 4e
      1090 20 54 4f
      1093 20 4c 4f
      1096 41 44 00
      1099 17 01 55     1517   msg6   .byt 23,1,'u - change unit/quit   d - change disk',0
      109c 20 2d 20
      109f 43 48 41
      10a2 4e 47 45
      10a5 20 55 4e
      10a8 49 54 2f
      10ab 51 55 49
      10ae 54 20 20
      10b1 20 44 20
      10b4 2d 20 43
      10b7 48 41 4e
      10ba 47 45 20
      10bd 44 49 53
```

```
error addr  code         seq   source statement

      10c0 4b 00
      10c2 09 01 54     1518   msg5   .byt 9,1,'there are no files on this disk!',0
      10c5 48 45 52
      10c8 45 20 41
      10cb 52 45 20
      10ce 4e 4f 20
      10d1 46 49 4c
      10d4 45 53 20
      10d7 4f 4e 20
      10da 54 48 49
      10dd 53 20 44
      10e0 49 53 4b
      10e3 21 00
                        1519   ;
                        1520   ;============================
                        1521   ;this is the data for the sprites
                        1522   ;that make up the title
                        1523   ;============================
                        1524   ;
      10e5 55 55 54     1525   titlez .byt $55,$55,$54,$6a,$aa,$a4,$6a,$aa
      10e8 6a aa a4
      10eb 6a aa
      10ed a4 6a aa     1526          .byt $a4,$6a,$aa,$a4,$55,$5a,$a4,$00
      10f0 a4 55 5a
      10f3 a4 00
      10f5 1a 90 00     1527          .byt $1a,$90,$00,$1a,$90,$00,$6a,$40
      10f8 1a 90 00
      10fb 6a 40
      10fd 00 6a 40     1528          .byt $00,$6a,$40,$01,$a9,$00,$01,$a9
      1100 01 a9 00
      1103 01 a9
      1105 00 06 a4     1529          .byt $00,$06,$a4,$00,$06,$a4,$00,$1a
      1108 00 06 a4
      110b 00 1a
      110d 90 00 1a     1530          .byt $90,$00,$1a,$90,$00,$6a,$95,$54
      1110 90 00 6a
      1113 95 54
      1115 6a aa a4     1531          .byt $6a,$aa,$a4,$6a,$aa,$a4,$6a,$aa
      1118 6a aa a4
      111b 6a aa
      111d a4 55 55     1532          .byt $a4,$55,$55,$54,$00,$00,$00,$24
      1120 54 00 00
      1123 00 24
                        1533   ;
      1125 55 55 54     1534   titlea .byt $55,$55,$54,$6a,$aa,$a4,$6a,$aa
      1128 6a aa a4
      112b 6a aa
      112d a4 6a aa     1535          .byt $a4,$6a,$aa,$a4,$69,$55,$a4,$69
      1130 a4 69 55
      1133 a4 69
      1135 01 a4 69     1536          .byt $01,$a4,$69,$01,$a4,$69,$01,$a4
      1138 01 a4 69
      113b 01 a4
      113d 69 55 a4     1537          .byt $69,$55,$a4,$6a,$aa,$a4,$6a,$aa
      1140 6a aa a4
      1143 6a aa
      1145 a4 6a aa     1538          .byt $a4,$6a,$aa,$a4,$69,$55,$a4,$69
```

```
error addr  code          seq   source statement

      1148 a4 69 55
      114b a4 69
      114d 01 a4 69      1539          .byt $01,$a4,$69,$01,$a4,$69,$01,$a4
      1150 01 a4 69
      1153 01 a4
      1155 69 01 a4      1540          .byt $69,$01,$a4,$69,$01,$a4,$69,$01
      1158 69 01 a4
      115b 69 01
      115d a4 55 01      1541          .byt $a4,$55,$01,$54,$00,$00,$00,$54
      1160 54 00 00
      1163 00 54
                         1542    ;
      1165 55 55 54      1543    titlep .byt $55,$55,$54,$6a,$aa,$a4,$6a,$aa
      1168 6a aa a4
      116b 6a aa
      116d a4 6a aa      1544          .byt $a4,$6a,$aa,$a4,$69,$55,$a4,$69
      1170 a4 69 55
      1173 a4 69
      1175 01 a4 69      1545          .byt $01,$a4,$69,$01,$a4,$69,$01,$a4
      1178 01 a4 69
      117b 01 a4
      117d 69 55 a4      1546          .byt $69,$55,$a4,$6a,$aa,$a4,$6a,$aa
      1180 6a aa a4
      1183 6a aa
      1185 a4 6a aa      1547          .byt $a4,$6a,$aa,$a4,$69,$55,$54,$69
      1188 a4 69 55
      118b 54 69
      118d 00 00 69      1548          .byt $00,$00,$69,$00,$00,$69,$00,$00
      1190 00 00 69
      1193 00 00
      1195 69 00 00      1549          .byt $69,$00,$00,$69,$00,$00,$69,$00
      1198 69 00 00
      119b 69 00
      119d 00 55 00      1550          .byt $00,$55,$00,$00,$00,$00,$00,$c4
      11a0 00 00 00
      11a3 00 c4
                         1551    ;
      11a5 00            1552    dirstr .byte 0 ;where directory entries ae stored
                         1553           .end
   0 errors detected
```

symbol table
(blank) = label, (=) = symbol, (+)= multibly defined

ad2061	080d	atnin	=0080	atnout	=0008	b1	09ec	b2	0a00	b3	0a22	b4	0a35	b5	0a45	bascod	0e34
bcnt15	=000c	bcnt64	=00fc	bdir	0a7c	bdir00	0a83	bdir05	0a90	bdir10	0a92	bdir15	0a98	bdir20	0a9c	bdir25	0aa7
bdir30	0aac	bdir35	0abf	bdir40	0ac9	bdir80	0ada	brstld	09cb	buf1	=0300	buf2	=0400	buf3	=0500	buf4	=0600
buf5	=0700	bufptr	=0010	byte15	=000b	byte64	=00fb	bzcode	=ff45	c1	0abd	c2	0ac5	cbmbas	0ebe	chkin	=ffc6
chrout	=ffd2	cin15	=0004	cin64	=0040	ckout	=ffc9	close	=ffc3	clrchn	=ffcc	cmdlp	099d	color	=0064	cout15	=0008
cout64	=0010	cptr	=0062	curlin	=02aa	curprg	=02a8	dcend	0b67	dclen	=01d2	dcntr	=02ad	dcode	0995	ddest	08e3
din15	=0001	din64	=0080	dirstr	11a5	dodir	09c8	dojmp	09a6	doli00	0fa9	doli10	0fae	doline	0fa5	domemc	08f2
donemc	0901	doseek	09b5	dout15	=0002	dout64	=0020	down00	0888	down15	0894	down16	08af	down20	08ba	down25	08d0
drmsg2	1041	drmssg	100a	drtype	=02af	dspf00	0ed8	dspf10	0ee4	dspf15	0ee8	dspf20	0f06	dspf25	0f09	dspf30	0f24
dspf50	0f25	dspf55	0f2e	dspfil	0ec5	dspm00	0d9a	dspm10	0d9d	dspm15	0da9	dspm16	0dba	dspm17	0dc9	dspm20	0dcf
dspm25	0ded	dspm30	0df3	dspm40	0e0b	dspm52	0e1a	dspm53	0e21	dspm54	0e2d	dspm55	0e4c	dspm56	0e55	dspm57	0e62
dspm58	0e6c	endbas	0e43	endxeq	0e4c	execut	0e43	f1	0b23	f2	0b28	fix19	0902	fixcia	0970	fixd81	0b68
flo10	0b14	flo15	0b20	flo20	0b22	flo30	0b26	flo50	0b3c	fload	0af4	fndp00	0f3d	fndprg	0f34	fprg	=02a9
fr0015	09dd	fr0064	0c5c	fr1064	0c6a	fram15	09ce	fram64	0c54	fs1500	0b47	fs1510	0b53	fs6400	0c2e	fstb15	0b40
fstb64	0c28	getin	=ffe4	gotd00	0d7c	gotdir	0d5f	gotprg	0bfe	gs	0a69	gt	0a6e	h1	09b7	h2	09bb
h3	0a6c	h4	0a71	hdrs	=0006	ib1015	0a2b	ib1064	0b8a	ib2015	0a37	ib2064	0b94	ibyt15	0a20	ibyt64	0a7f
j1	09bf	j2	09c1	j3	0a79	j4	0aea	joboff	=000f	jobs	=0000	l15	09c1	lpb15	0780	lpb64	=00fd
memcmd	08e0	memexe	08ec	memred	08e6	menu	0c79	menu10	0c9c	menu20	0ca7	menu25	0ca9	menu35	0cc8	menu40	0cde
menu45	0cf7	menu50	0d1c	menu55	0d25	menu60	0d3a	menu65	0d4a	mesa00	0fdf	mesa10	0fea	mesa15	0fec	mesa20	0ff9
mesage	0fbf	mlcode	0ec2	msg1	1006	msg2	1019	msg3	1044	msg4	106f	msg5	10c2	msg6	1099	msgtbl	0ffa
no1581	0882	noblnk	0e96	noprog	0d72	notbas	0e49	nxmcod	094a	nxmemc	08f4	nxtf00	0f58	nxtfil	0f4d	ob1015	09f7
ob1064	0bb6	ob2015	0a06	ob2064	0bc2	ob3015	0a09	obyt15	09ec	obyt64	0bab	okblnk	0e89	okvart	0c08	open	=ffc0
opener	0829	p1	0995	p10	0a14	p11	0a2b	p12	0a37	p2	099a	p3	09ce	p4	09d9	p5	09ad
p6	09e8	p7	09f7	p8	0a06	p9	0a09	pb15	=1800	pb64	=dd00	postbl	0c73	prgcnt	=02a7	ptr	=00fe
ptr2	=006a	read	=0080	reopen	0814	rfblok	0a48	rnb100	0a7b	rnblok	0a52	scend	0c72	scrlin	0f73	sct00	0be6
sct10	0be9	sctop	0bd8	seek	=00b0	setfp	0f44	setlfs	=ffba	setnam	=ffbd	setptr	0f59	snex00	0aea	snext	0aad
sysprg	0c25	temp	=0065	titlea	1125	titlep	1165	titlez	10e5	unitno	=02b0	vartab	=002d	vptr	=006c	w1	0a54
w2	0a5b	waitdk	0cef	wrklin	=02ab	wrkoff	=000a	wrkprg	=02ac	xfram	=ffc1	xoay15	=0557				

cross reference
(<#> = definition, <$> = write, <blank> = read)

cross reference
(<#> = definition, <$> = write, <blank> = read)

		699	707	754	756				
ddest	08e3	142$	144$	196	198$	200$	235#		
din15	=0001	63#							
din64	=0080	9#	958	965					
dirstr	11a5	1397	1399	1552#					
dodir	09c8	375	405#						
dojmp	09a6	369	372#						
doli00	0fa9	1455#	1465						
doli10	0fae	1457#	1462						
doline	0fa5	989	994	1122	1140	1453#			
domemc	08f2	154	186	229	245#				
donemc	0901	248	253#						
doseek	09b5	300$	373	388#					
dout15	=0002	64#	350	426	436	467	477	735	743
dout64	=0020	10#	828	838					
down00	0888	183#	213	217					
down15	0894	188#	192						
down16	08af	199	202#						
down20	08ba	206	209#						
down25	08d0	215	226#						
drmsg2	1041	176$	1514#						
drmssg	100a	175$	1512#						
drtype	=02af	50#	163$	1270					
dspf00	0ed8	1325#	1359	1363					
dspf10	0ee4	1328	1330#						
dspf15	0ee8	1332#	1339						
dspf20	0f06	1346	1350#						
dspf25	0f09	1352#							
dspf30	0f24	1364#	1368						
dspf50	0f25	1355	1366#						
dspf55	0f2e	1371#	1373						
dspfil	0ec5	1146	1317#						
dspm00	0d9a	1146#	1169	1183	1185	1197	1204		
dspm10	0d9d	1147#	1148	1159	1175	1193	1195	1218	
dspm15	0da9	1150	1153#						
dspm16	0dba	1161#							
dspm17	0dc9	1164	1168#						
dspm20	0dcf	1154	1171#						
dspm25	0ded	1180	1184#						
dspm30	0df3	1172	1187#						
dspm40	0e0b	1188	1199#						
dspm52	0e1a	1200	1206#						
dspm53	0e21	1207	1210#						
dspm54	0e2d	1211	1216#						
dspm55	0e4c	1217	1247#						
dspm56	0e55	1252#	1256						
dspm57	0e62	1259#	1262						
dspm58	0e6c	1265#	1268						
endbas	0e43	1237#	1264						
endxeq	0e4c	1244#	1258						
execut	0e43	1239#	1258	1259					
f1	0b23	696#	760$	764$					
f2	0b28	699#	761$	765$					
fix19	0902	173	261#						
fixc1a	0970	309	313#						
fixd61	0b68	291	759#						
flo10	0b14	687#	709						

cross reference
(<#> = definition, <$> = write, <blank> = read)

cross reference
(<#> = definition, <$> = write, <blank> = read)

cross reference
(<#> = definition, <$> = write, <blank> = read)

```
rnbl00  0a7b     561    565#
rnblok  0a52     545#   658
scend   0c72     969#  1251
scrlin  0f73    1427   1430   1440#
sct00   0be0     867#   883    885
sct10   0be9     873#   877
sctop   0bd8     862#   867    873   1228   1251   1252   1275$  1277$
seek   =00b0      73#   392
setfp   0f44    1088   1384   1397#
setlfs =ffba      22#   130
setnam =ffbd      23#   132
setptr  0f59    1070   1318   1422#  1455   1483
snex00  0aea     655#   656
snext   0add     585    603    647#   675    707
sysprg  0c25     910    913#
temp   =0065      38#  1330$  1333   1454$  1464$
titlea  1125    1534#
titlep  1165    1543#
titlez  10e5     997   1525#
unitno =02b0      51#   115$   121$   122    129
vartab =002d      39#   863$   865$   874$   881    882$   884$   892$   893    896$
vptr   =006c      34#  1077$  1334$  1342   1344$  1347$  1371$  1428$  1431$  1460$  1487   1488$  1490$  1504$
w1      0a54     275$   546#
w2      0a5b     276$   550#
waitdk  0cef    1047#  1048
wrklin =02ab      47#  1320$  1360$  1361
wrkoff =000a      87#   546$   550
wrkprg =02ac      48#  1322$  1326   1352   1356$
xfram  =ffc1    1228#  1274   1276
xoby15 =0557     756#   759    763
```

CHAPTER 9

RAM Expansion

Hardware: C64 or C128 with the 1764, 1700 or 1750 RAM Expansion Cards

1) Finds size of any RAM card
2) General purpose stash and fetch routines

Known bugs: None

The RAM expansion code consists of 3 routines: howbig, stash and fetch.

Howbig is a routine which will find the size of the RAM card installed. It works with all 3 RAM card models on both the 64 and 128. Call the routine. Results are returned in the accumulator:

.A = 8 for the 1750 512K RAM expander
.A = 2 for the 1700 128K RAM expander
.A = 4 for the 1764 256K RAM expander
.A = 1 if there is no RAM expander

Stash and Fetch are general pupose routines that allow you to move memory between the CPU and expansion card. Stash and Fetch work with all 3 RAM card models on both the C64 and C128. The routines also handle ROM banking on the 128 and 64 so that you can stash and fetch to any memory location.

```
error addr  code          seq   source statement

                           1    ;
                           2    ;
          =1800            3            *=$1800
                           4    ;
                           5    ;
                           6    ;=======================================
                           7    ;                 equates
                           8    ;=======================================
                           9    ;
                          10    ;
          =1700           11    buffer  = $1700
          =00fb           12    numbank = $fb
          =df00           13    ramexp  = $df00
          =d506           14    rcr     = $d506
                          15    ;
                          16    ;
                          17    ;=======================================
                          18    ;                 jump table
                          19    ;=======================================
                          20    ;
                          21    ;
     1800 4c 1813         22    start   jmp howbig
     1803 4c 187e         23            jmp stash
     1806 4c 187b         24            jmp fetch
                          25    ;
                          26    ;
                          27    ;=========================================
                          28    ;              DMA parameters
                          29    ;=========================================
                          30    ;
                          31    ;
     1809 0000            32    params  .word $0000     ; Host address, lo, hi
     180b 0000            33            .word $0000     ; Exp  address, lo, hi
     180d 00              34    expbank .byte $00       ; Expansion  bank no.
     180e 0100            35            .word $0100     ; # bytes to move, lo, hi
     1810 00              36            .byte $00       ; Interrupt mask reg.
     1811 00              37            .byte $00       ; Adress control reg.
                          38    ;
     1812 00              39    bnk128  .byte $00       ; Bank of 128 to work with
     1813                 40    pend
                          41    ;
                          42    ;
                          43    ;=========================================
                          44    ;    Test ram expander to determine size
                          45    ;=========================================
                          46    ;
                          47    ;  Number of banks is returned in .A
                          48    ;  and in numbank.
                          49    ;
                          50    ;   .A =  8 for the 1750 512K expander
                          51    ;   .A =  2 for the 1700 128K expander
                          52    ;   .A =  4 for the 1764 256K expander
                          53    ;   .A =  1 for no RAM expander
                          54    ;
                          55    ;=========================================
                          56    ;
                          57    ;
```

```
error addr  code          seq   source statement

                          58    ;
                          59                          ;  Here are the 8 parameters we
      1813 a2 17          60    howbig  ldx #>buffer  ; must set for stash and fetch:
      1815 8e 180a        61            stx params+1  ; First, set up the hi bytes of
      1818 8e 180c        62            stx params+3  ; the cpu and expansion address.
      181b a2 01          63            ldx #$01
      181d 8e 180f        64            stx params+6  ;  Set up the byte count hi
      1820 ca             65            dex           ; and the byte count lo.
      1821 8e 180e        66            stx params+5
      1824 8e 180d        67            stx expbank   ;  Set up the expansion bank to
      1827 8e 1809        68            stx params+0  ; use and the lo bytes of the
      182a 8e 180b        69            stx params+2  ; cpu and expansion address.
      182d 8e 1812        70            stx bnk128    ; Set the 128 bank to work with.
                          71    ;
      1830 8a             72    20$     txa           ;  Generate a 1 block
      1831 49 5a          73            eor #$5a      ; test pattern in
      1833 9d 1700        74            sta buffer,x  ; buffer.
      1836 ca             75            dex
      1837 d0 f7          76            bne 20$
                          77    ;
      1839 20 187e        78    30$     jsr stash     ;  Now write the test
      183c ee 180d        79            inc expbank   ; pattern in buffer
      183f ad 180d        80            lda expbank   ; to each of the 8
      1842 c9 08          81            cmp #8        ; possible exp. banks.
      1844 d0 f3          82            bne 30$
                          83    ;
      1846 a2 00          84            ldx #0
      1848 8e 180d        85            stx expbank
                          86    ;
      184b a2 00          87    40$     ldx #0        ;  Ok, now change
      184d 8a             88    50$     txa           ; the 1 block test
      184e 49 3c          89            eor #$3c      ; pattern in the buffer
      1850 9d 1700        90            sta buffer,x  ; to a new pattern.
      1853 ca             91            dex
      1854 d0 f7          92            bne 50$
                          93    ;
      1856 20 187e        94            jsr stash     ;  Now write the new
      1859 ee 180d        95            inc expbank   ; pattern to bank (x)...
                          96
      185c ad 180d        97            lda expbank   ;   (check to see
      185f c9 08          98            cmp #8        ;   if we are done)
      1861 f0 12          99            beq 90$
                         100
      1863 20 187b       101            jsr fetch     ; ...and read the pattern
      1866 a2 00         102            ldx #0        ;    from bank (x+1).
      1868 8a            103    60$     txa           ;
      1869 49 5a         104            eor #$5a      ;  We should see the old pattern
      186b dd 1700       105            cmp buffer,x  ; here.  If we don t then the data
      186e d0 05         106            bne 90$       ; changed and we have found the end.
      1870 ca            107            dex
      1871 d0 f5         108            bne 60$       ; Bytes match so all is well.
      1873 f0 d6         109            beq 40$       ; Loop back for next bank.
                         110    ;       .
      1875 ad 180d       111    90$     lda expbank   ;  Number of banks is returned in
      1878 85 fb         112            sta numbank   ; the accumulator and in numbank.
      187a 60            113            rts
                         114    ;
```

```
error addr  code          seq   source statement

                          115   ;
                          116   ;===============================================
                          117   ;          stash  &  fetch  subroutines
                          118   ;===============================================
                          119   ;
                          120   ;  These routines will transfer RAM between the
                          121   ; cpu and expansion unit on the c64 and c128.
                          122   ; Before calling, you must set up 8 parameters
                          123   ; for the DMA as follows:
                          124   ;
                          125   ;       source address      (lo, hi)
                          126   ;       destination address (lo, hi)
                          127   ;       expansion bank
                          128   ;       number of bytes     (lo, hi)
                          129   ;       128 bank to use
                          130   ;
                          131   ;  (parameters are located at "params")
                          132   ;
                          133   ;  You may stash or fetch at any address.
                          134   ; These routines will bank out ROMs and I/O
                          135   ; before starting the DMA.
                          136   ;
                          137   ;  If you want to fetch or stash RAM bank 1
                          138   ; on the C128 be sure to make a copy of this
                          139   ; code in bank 1 too.
                          140   ;
                          141   ;===============================================
                          142   ;
                          143   ;
     187b a0 ed           144   fetch   ldy #$ed        ;  Command to read from expander
     187d 2c              145           .byte $2c       ; with FF00 option enabled.
                          146                           ; (skip 2 bytes)
                          147
     187e a0 ec           148   stash   ldy #$ec        ;  Command to write to expander
                          149                           ; with FF00 option enabled.
     1880 a2 08           150           ldx #pend-params-2
     1882 bd 1809         151   10$     lda params,x    ;  Initialize the DMA
     1885 9d df02         152           sta $df02,x     ; contoller with our
     1888 ca              153           dex             ; parameters,
     1889 10 f7           154           bpl 10$         ;
     188b 8c df01         155           sty $df01       ; and issue command.
                          156
     188e ac 1812         157           ldy bnk128      ;  Set the .y register to the
                          158                           ; 128 bank we want (0 or 1).
                          159   ;
                          160   ;
                          161   ;================================
                          162   ;  turn off ROMS and start DMA
                          163   ;================================
                          164   ;
                          165   ;
     1891                 166   dmarom
     1891 ad ffff         167           lda $ffff   :  The high byte of the
     1894 c9 fc           168           cmp #$fc    : reset vector on all C64s
     1896 f0 2a           169           beq xfer64  : is equal to $fc.
                          170   ;
                          171   ;
```

```
error addr  code          seq   source statement

                          172   ;============================
                          173   ;  c128: turn off all ROMs
                          174   ;============================
                          175   ;
      1898 78             176         sei
      1899 ad d506        177         lda rcr       ;   Save the old value of
      189c 48             178         pha           ; the 128 rcr.  Now convert
      189d 98             179         tya           ; the 128 bank number to a
      189e f0 07          180         beq bk0128    ; mask for the VIC/DMA
      18a0 a9 40          181         lda #$40      ; pointer in the rcr.
      18a2 0d d506        182         ora rcr       ;   This allows a stash or
      18a5 d0 05          183         bne bangit    ; fetch to bank 0 or bank 1
      18a7 a9 3f          184   bk0128 lda #$3f     ; of the 128.  When using
      18a9 2d d506        185         and rcr       ; bank 1, be sure to make a copy
      18ac 8d d506        186   bangit sta rcr      ; of this code in both banks.
                          187   ;
      18af ad ff00        188         lda $ff00     ;  Save the 128 configuration
      18b2 48             189         pha           ; now kill ROMs and I/O.
      18b3 09 3f          190         ora #$3f      ; When we write to FF00
      18b5 8d ff00        191         sta $ff00     ; DMA execution begins.
      18b8 68             192         pla
      18b9 8d ff00        193         sta $ff00     ;  Restore the old
      18bc 68             194         pla           ; configuration and
      18bd 8d d506        195         sta rcr       ; restore the old VIC
      18c0 58             196         cli           ; pointer in the rcr.
      18c1 60             197         rts
                          198   ;
                          199   ;============================
                          200   ;  c64: turn off all ROMs
                          201   ;============================
                          202   ;
      18c2 78             203   xfer64  sei         ;  Save the value of the
      18c3 a5 01          204           lda $01     ; the c64 control port
      18c5 48             205           pha         ; and turn on lower 3 bits
      18c6 09 03          206           ora #$03    ; to bank out ROMs, I/O.
      18c8 85 01          207           sta $01
      18ca 8d ff00        208           sta $ff00   ;  Now start transfer...
                          209   ;
      18cd 68             210           pla         ;  Restore the old
      18ce 85 01          211           sta $01     ; configuration
      18d0 58             212           cli         ; and return.
      18d1 60             213           rts
                          214   ;
      18d2                215   end
                          216   ;
                          217           .end
  0 errors detected
```

symbol table
(blank) = label, (=) = symbol, (+)= multibly defined

banqit	18ac	bk0128	18a7	bnk128	1812	buffer	=1700	dmarom	1891	end	18d2	expbank	180d	fetch	187b
howbig	1813	numbank	=00fb	params	1809	pend	1813	ramexp	=df00	rcr	=d506	start	1800	stash	187e
xfer64	18c2														

cross reference
(<#> = definition, <$> = write, <blank> = read)

```
bangit   18ac    183    186#
bk0128   18a7    180    184#
bnk128   1812     39#    70$   157
buffer  =1700     11#    60     74$    90$   105
dmarom   1891    166#
end      18d2    215#
expbank  180d     34#    67$    79$    80     85$    95$    97    111
fetch    187b     24    101    144#
howbig   1813     22     60#
numbank =00fb     12#   112$
params   1809     32#    61$    62$    64$    66$    68$    69$   150   151
pend     1813     40#   150
ramexp  =df00     13#
rcr     =d506     14#   177    182    185    186$   195$
start    1800     22#
stash    187e     23     78     94    148#
xfer64   18c2    169    203#
```

CHAPTER 10

1351 Mouse Driver #1

Hardware: C64 or C128 with a 1351 mouse

1) BASIC-compatible mouse drivers for the C-64 and C-128.

Known bugs: None

Here are two driver routines for the 1351 mouse. There is one for the C64 and one for the C128. These routines originally appeared in the 1351 Mouse User Guide.

Mouse data appears in the SID chip registers and is read by a short wedge program running off of the IRQ. Shown below are two sample BASIC programs which call the 1351 mouse driver routines:

C64

```
10 IF Z=0 THEN Z=Z+1 :LOAD"MOUSE.POINTER",8,1
20 IF Z=1 THEN Z=Z+1 :LOAD"MOUSE64.BIN",8,1
30 V=13*4096:POKEV+21,1:POKEV+39,1:POKEV+0,100:POKEV+1,100:POKEV+16,0
40 POKE2040,56:SYS12*4096+256
```

C128

```
10 BLOAD"MOUSE.POINTER":KEY8,"."
20 BLOAD"MOUSE128.BIN" :SYSDEC("1800")
30 BA=DEC("0A04") :POKE BA,1 OR PEEK(BA)
40 SPRITE 1,1,2 :MOVSPR 1,100,100
50 GRAPHIC1,1 :CHAR 1,8,1,"BASIC CHEAPO PAINT (TM)"
100 DO:GETKEY A$:IF A$=" "THEN GRAPHIC1,1:CHAR 1,8,1,"BASIC CHEAPO PAINT (TM)"
130 IF JOY(1)<>0 THEN GOSUB1000
140 LOOP
1000 X=RSPPOS(1,0)-25:Y=RSPPOS(1,1)-51:X=-X*(X>0):Y=-Y*(Y>0)
1010 LOCATE X,Y:C=1-RDOT(2):DRAW C,X,Y
1020 DO: X=RSPPOS(1,0)-25:Y=RSPPOS(1,1)-51:X=-X*(X>0):Y=-Y*(Y>0)
1030 DRAW C TO X,Y:LOOP WHILE JOY(1)<>0 : RETURN
```

```
error addr  code          seq   source statement

                           1    ;
                           2    ;
                           3    ;
                           4    ;=========================================
                           5    ;      1351 mouse basic-compatible driver
                           6    ;              for the c128
                           7    ;=========================================
                           8    ;
                           9    ;
                          10    ;
          =0314           11    iirq    = $0314
          =d000           12    vic     = $d000
          =d400           13    sid     = $d400
          =d419           14    potx    = sid+$19
          =d41a           15    poty    = sid+$1a
                          16    ;
                          17    ;
          =117e           18    active  = $117e         ; if zero, then move sprite
                          19    ;
          =11d6           20    vicdata = $11d6         ; basics copy of vic register image
          =11d6           21    xpos    = vicdata+$00   ; low order xposition
          =11d7           22    ypos    = vicdata+$01   ; y position
          =11e6           23    xposmso = vicdata+$10   ; bit 0 is high order x position
                          24    ;
                          25    ;
          =18f0           26          *=$18f0
                          27    ;
                          28    ;
    18f0 =18f2            29    iirq2   *=*+2
    18f2 =18f3            30    opotx   *=*+1
    18f3 =18f4            31    opoty   *=*+1
    18f4 =18f5            32    newvalue        *=*+1
    18f5 =18f6            33    oldvalue        *=*+1
                          34    ;
                          35    ;
          =1800           36          * = $1800
                          37    ;
                          38    ;
    1800 ad 0315          39    install lda iirq+1
    1803 c9 18            40          cmp #>mirq
    1805 f0 19            41          beq 90$
    1807 08               42          php
    1808 78               43          sei
    1809 ad 0314          44          lda iirq
    180c 8d 18f0          45          sta iirq2
    180f ad 0315          46          lda iirq+1
    1812 8d 18f1          47          sta iirq2+1
                          48
    1815 a9 21            49          lda #<mirq
    1817 8d 0314          50          sta iirq
    181a a9 18            51          lda #>mirq
    181c 8d 0315          52          sta iirq+1
                          53    ;
    181f 28               54          plp
    1820 60               55    90$   rts
                          56    ;
                          57
```

```
error addr  code          seq   source statement

                          58
                          59
                          60
                          61
                          62
                          63
                          64
      1821 d8             65    mirq    cld             ; just in case.....
      1822 ad 117e        66            lda active      ; if    basic is moveing sprite
      1825 d0 33          67            bne 90$         ;       let basic have it ( why not ? )
      1827 ad d419        68            lda potx        ; get delta values for x
      182a ac 18f2        69            ldy opotx
      182d 20 185d        70            jsr movchk
      1830 8c 18f2        71            sty opotx
                          72    ;
      1833 18             73            clc             ; modify low order xposition
      1834 6d 11d6        74            adc xpos
      1837 8d 11d6        75            sta xpos
      183a 8a             76            txa
      183b 69 00          77            adc #$00
      183d 29 01          78            and #%00000001
      183f 4d 11e6        79            eor xposmsb
      1842 8d 11e6        80            sta xposmsb
                          81    ;
      1845 ad d41a        82            lda poty        ; get delta value for y
      1848 ac 18f3        83            ldy opoty
      184b 20 185d        84            jsr movchk
      184e 8c 18f3        85            sty opoty
                          86    ;
      1851 38             87            sec             ; modify y position ( decrease y for increase in pot )
      1852 49 ff          88            eor #$ff
      1854 6d 11d7        89            adc ypos
      1857 8d 11d7        90            sta ypos
                          91    ;
      185a 6c 18f0        92    90$     jmp (iirq2)     ; continue w/ irq operation
                          93    ;
                          94    ;================================================
                          95    ; movchk
                          96    ;       entry   y = old value of pot register
                          97    ;               a = currrent value of pot register
                          98    ;       exit    y = value to use for old value
                          99    ;               x,a = delta value for position
                         100    ;================================================
                         101    ;
      185d 8c 18f5       102    movchk  stv oldvalue    ;save old & new values
      1860 8d 18f4       103            sta newvalue
      1863 a2 00         104            ldx #0          ;preload x w/ 0
                         105    ;
      1865 38            106            sec             ;a <=  mod64( new-old )
      1866 ed 18f5       107            sbc oldvalue
      1869 29 7f         108            and #%01111111
      186b c9 40         109            cmp #%01000000  ;if      > 0
      186d b0 07         110            bcs 50$
      186f 4a            111            lsr a           ;       a <= a/2
      1870 f0 12         112            beq 80$         ;       if      <> 0
      1872 ac 18f4       113            ldy newvalue    ;               y <= newvalue
      1875 60            114            rts             ;               return
```

```
error addr  code          seq   source statement

                          115   ;
      1876 09 c0          116   50$     ora #%11000000  ;else   or in high order bits
      1878 c9 ff          117           cmp #$ff        ;       if      <> -1
      187a f0 08          118           beq 80$
      187c 38             119           sec             ;               a <= a/2
      187d 6a             120           ror a
      187e a2 ff          121           ldx #$ff        ;               x <= -1
      1880 ac 18f4        122           ldy newvalue    ;               y <= newvalue
      1883 60             123           rts             ;               return
                          124   ;
      1884 a9 00          125   80$     lda #0          ;a <= 0
      1886 60             126           rts             ;return w/ y = old value
                          127   ;
                          128           .end
   0 errors detected
```

symbol table
(blank) = label, (=) = symbol, (+)= multibly defined

active	=117e	iirq	=0314	iirq2	18f0	install	1800	mirq	1821	movchk	185d	newvalue	18f4	oldvalue	18f5
opotx	18f2	opoty	18f3	potx	=d419	poty	=d41a	sid	=d400	vic	=d000	vicdata	=11d6	xpos	=11d6
xposmsb	=11e6	ypos	=11d7												

cross reference
((#) = definition, (\$) = write, (blank) = read)

active	=117e	18#	66				
iirq	=0314	11#	39	44	46	50\$	52\$
iirq2	18f0	29#	45\$	47\$	92		
install	1800	39#					
mirq	1821	40	49	51	65#		
movchk	185d	70	84	102#			
newvalue	18f4	32#	103\$	113	122		
oldvalue	18f5	33#	102\$	107			
opotx	18f2	30#	69	71\$			
opoty	18f3	31#	83	85\$			
potx	=d419	14#	68				
poty	=d41a	15#	82				
sid	=d400	13#	14	15			
vic	=d000	12#					
vicdata	=11d6	20#	21	22	23		
xpos	=11d6	21#	74	75\$			
xposmsb	=11e6	23#	79	80\$			
ypos	=11d7	22#	89	90\$			

```
error addr  code          seq   source statement

                           1    ;
                           2    ;
                           3    ;
                           4    ;=======================================
                           5    ;      1351 mouse basic-compatible driver
                           6    ;             for the c64
                           7    ;=======================================
                           8    ;
                           9    ;
                          10    ;
          =0314            11    iirq    = $0314
          =d000            12    vic     = $d000
          =d400            13    sid     = $d400
          =d419            14    potx    = sid+$19
          =d41a            15    poty    = sid+$1a
                           16    ;
                           17    ;
          =d000            18    vicdata = $d000         ; basics copy of vic register image
          =d000            19    xpos    = vicdata+$00   ; low order xposition
          =d001            20    ypos    = vicdata+$01   ; y position
          =d010            21    xposmsb = vicdata+$10   ; bit 0 is high order x position
                           22    ;
                           23    ;
          =c000            24          *=$c000
                           25    ;
                           26    ;
     c000 =c002            27    iirq2   *=*+2
     c002 =c003            28    opotx   *=*+1
     c003 =c004            29    opoty   *=*+1
     c004 =c005            30    newvalue        *=*+1
     c005 =c006            31    oldvalue        *=*+1
                           32    ;
                           33    ;
          =c100            34          * = $c100
                           35    ;
                           36    ;
     c100 ad 0315          37    install lda iirq+1
     c103 c9 c1            38          cmp #>mirq
     c105 f0 19            39          beq 90$
     c107 08               40          php
     c108 78               41          sei
     c109 ad 0314          42          lda iirq
     c10c 8d c000          43          sta iirq2
     c10f ad 0315          44          lda iirq+1
     c112 8d c001          45          sta iirq2+1
                           46    ;
     c115 a9 21            47          lda #<mirq
     c117 8d 0314          48          sta iirq
     c11a a9 c1            49          lda #>mirq
     c11c 8d 0315          50          sta iirq+1
                           51    ;
     c11f 28               52          plp
     c120 60               53    90$   rts
                           54    ;
                           55    ;
                           56
                           57
```

```
error addr  code          seq   source statement

                          58
                          59
                          60
                          61
                          62
                          63
                          64
                          65
                          66
     c121 d8              67    mirq    cld             ; just in case.....
     c122 ad d419         68            lda potx        ; get delta values for x
     c125 ac c002         69            ldy opotx
     c128 20 c158         70            jsr movchk
     c12b 8c c002         71            sty opotx
                          72    ;
     c12e 18              73            clc             ; modify low order xposition
     c12f 6d d000         74            adc xpos
     c132 8d d000         75            sta xpos
     c135 8a              76            txa
     c136 69 00           77            adc #$00
     c138 29 01           78            and #%00000001
     c13a 4d d010         79            eor xposmsb
     c13d 8d d010         80            sta xposmsb
                          81    ;
     c140 ad d41a         82            lda poty        ; get delta value for y
     c143 ac c003         83            ldy opoty
     c146 20 c158         84            jsr movchk
     c149 8c c003         85            sty opoty
                          86    ;
     c14c 38              87            sec             ; modify y position ( decrease y for increse in pot )
     c14d 49 ff           88            eor #$ff
     c14f 6d d001         89            adc ypos
     c152 8d d001         90            sta ypos
                          91    ;
     c155 6c c000         92    90$     jmp (iirq2)     ; continue w/ irq operation
                          93    ;
                          94    ;================================================
                          95    ; movchk
                          96    ;       entry   y = old value of pot register
                          97    ;               a = currrent value of pot register
                          98    ;       exit    y = value to use for old value
                          99    ;               x,a = delta value for position
                         100    ;================================================
                         101    ;
     c158 8c c005        102    movchk  sty oldvalue    ;save old &  new values
     c15b 8d c004        103            sta newvalue
     c15e a2 00          104            ldx #0          ;preload x w/ 0
                         105    ;
     c160 38             106            sec             ;a <=  mod64( new-old )
     c161 ed c005        107            sbc oldvalue
     c164 29 7f          108            and #%01111111
     c166 c9 40          109            cmp #%01000000  ;if   > 0
     c168 b0 07          110            bcs 50$
     c16a 4a             111            lsr a           ;       a <= a/2
     c16b f0 12          112            beq 60$         ;       if      <> 0
     c16d ac c004        113            ldy newvalue    ;               y <= newvalue
     c170 60             114            rts             ;       return
```

```
error addr  code          seq   source statement

                          115   ;
      c171 09 c0          116   50$     ora #%11000000  ;else   or in high order bits
      c173 c9 ff          117           cmp #$ff        ;       if      <> -1
      c175 f0 08          118           beq 80$
      c177 38             119           sec             ;               a <= a/2
      c178 6a             120           ror a
      c179 a2 ff          121           ldx #$ff        ;               x <= -1
      c17b ac c004        122           ldy newvalue    ;               y <= newvalue
      c17e 60             123           rts             ;               return
                          124   ;
      c17f a9 00          125   80$     lda #0          ;a <= 0
      c181 60             126           rts             ;return w/ y = old value
                          127   ;
                          128           .end
   0 errors detected
```

symbol table
(blank) = label, (=) = symbol, (+)= multibly defined

```
iirq     =0314  iirq2     c000  install   c100  mirq      c121  movchk    c158  newvalue  c004  oldvalue  c005  opotx     c002
opoty     c003  potx     =d419  poty     =d41a  sid      =d400  vic      =d000  vicdata  =d000  xpos     =d000  xposmsb  =d010
ypos     =d001
```

cross reference
((#) = definition, ($) = write, (blank) = read)

```
iirq     =0314    11#     37      42      44      48$     50$
iirq2     c000    27#     43$     45$     92
install   c100    37#
mirq      c121    38      47      49      67#
movchk    c158    70      84     102#
newvalue  c004    30#    103$    113     122
oldvalue  c005    31#    102$    107
opotx     c002    28#     69      71$
opoty     c003    29#     83      85$
potx     =d419    14#     68
poty     =d41a    15#     82
sid      =d400    13#     14      15
vic      =d000    12#
vicdata  =d000    18#     19      20      21
xpos     =d000    19#     74      75$
xposmsb  =d010    21#     79      80$
ypos     =d001    20#     89      90$
```

1351 Mouse Driver #2

Hardware: C64 or C128 with a 1351 mouse

1) BASIC-compatible mouse drivers for the C-64 and C-128.

Known bugs: None

Here are two additional 1351 mouse driver routines. There is a driver for the C128 and C64.

These routines work in a manner similar to driver #1, but are more powerful and a little more complicated. There are three entry points to the driver. The first entry point is for users with a mouse connected to Port 1. The second entry point is for users with a mouse on port 2. The third entry point will remove the mouse driver wedge from the system.

C64

```
10 REM
20 IF Z=0 THEN Z=1:LOAD"MOUSE.POINTER",8,1
30 IF Z=1 THEN Z=2:LOAD"M1351.64.BIN",8,1
40 INPUT"MOUSE PORT (1/2)";P$:P=VAL(P$)-1
50 IF P<0 OR P>1 THEN 40
60 V=13*4096:POKEV+21,1:POKEV+39,1:      REM SPRITE #1 ON, COLOR
70 POKEV+0,100:POKEV+1,100:POKEV+16,0:   REM SPRITE POSITION
80 POKE 2040,56:                         REM SPRITE DATA @ $E00
90 SYS12*4096+P*3                        REM INSTALL MOUSE DRIVER
```

C128

```
10 REM
15 PRINT"PORT (1/2) ? ";P$
20 DO:GETP$:P=VAL(P$)-1:LOOP UNTIL P=0 OR P=1:PRINT P+1
30 POKE DEC("FE"),P:                      REM SAVE PORT CHOICE
40 BLOAD"MOUSE.POINTER":                  REM LOAD SPRITE DATA
50 BLOAD"M1351.128.BIN":                  REM LOAD MOUSE DRIVER
60 SPRITE 1,1,2:MOVESPR1,100,100:         REM TURN ON SPRITE #1
70 SYS DEC("1800")+P*3:                   REM INSTALL MOUSE IRQ DRIVER
80 XF=25:YF=51:U=1:P=PEEK(DEC("FE"))+1:TRAP900
90 GRAHIC 1,1:CHAR,8,1,"BASIC CHEAPO PAINT (TM)"
100 DO
110 DO:GETA$:LOOP UNTIL JOY(P) OR A$=" "
115 IF JOY(P)=128 THEN 130:ELSE IF JOY(P)=U THEN RUN 80
120 COLOR1,(RCLR(1)AND15)+1:COLOR4,RCLR(1):LOOP
130 LOCATE RSPPOS(U,.)-XF,RSPPOS(U,U)-YF:C=NOT(RDOT(2))AND1
140 DO:DRAW C TO RSPPOS(U,.)-XF,RSPPOS(U,U)-YF:LOOP WHILE JOY(P):LOOP
150 :
900 IF ER=14 THEN RESUME:                 REM IGNORE NEGATIVE CO-ORD.S
910 SYS DEC("1806"):                      REM REMOVE MOUSE IRQ WEDGE
920 DO:GETA$:LOOP UNTIL A$="":            REM EMPTY KEY BUFFER
930 TRAP:END
```

```
error addr  code          seq   source statement

                           1    ;       1351 proportional mouse driver for the c64
                           2    ;
                           3    ;       commodore business machines, inc.   27oct86
                           4    ;               by hedley davis and fred bowen
                           5
          =0314            6    iirq    = $0314
          =d000            7    vic     = $d000
          =d400            8    sid     = $d400
          =dc00            9    cia     = $dc00
          =dc02           10    cia.ddr = $dc02
          =d419           11    potx    = sid+$19
          =d41a           12    poty    = sid+$1a
                          13
          =d000           14    xpos    = vic+$00       ;x position (lsb)
          =d001           15    ypos    = vic+$01       ;y position
          =d010           16    xposmsb = vic+$10       ;x position (msb)
                          17
          =c0f0           18            *=$c0f0
                          19
     c0f0 =c0f2           20    iirq2           *=*+2
     c0f2 =c0f3           21    opotx           *=*+1
     c0f3 =c0f4           22    opoty           *=*+1
     c0f4 =c0f5           23    newvalue        *=*+1
     c0f5 =c0f6           24    oldvalue        *=*+1
     c0f6 =c0f7           25    ciasave         *=*+1
                          26
                          27
          =c000           28            * = $c000
                          29
     c000 4c c009         30            jmp install.1   ;install mouse in port 1
     c003 4c c00c         31            jmp install.2   ;install mouse in port 2
     c006 4c c035         32            jmp remove      ;remove mouse wedge
                          33
                          34
     c009 a2 00           35    install.1       ldx #0          ;port 1 mouse
     c00b 2c              36                    .byte $2c
                          37
     c00c a2 02           38    install.2       ldx #2          ;port 2 mouse
                          39
     c00e ad 0315         40            lda iirq+1      ;install irq wedge
     c011 c9 c0           41            cmp #>mirq.1
     c013 f0 1b           42            beq 90$         ;...branch if already installed!
     c015 08              43            php
     c016 78              44            sei
                          45
     c017 ad 0314         46            lda iirq        ;save current irq indirect for our exit
     c01a 8d c0f0         47            sta iirq2
     c01d ad 0315         48            lda iirq+1
     c020 8d c0f1         49            sta iirq2+1
                          50
     c023 bd c031         51            lda port,x      ;point irq indirect to mouse driver
     c026 8d 0314         52            sta iirq
     c029 bd c032         53            lda port+1,x
     c02c 8d 0315         54            sta iirq+1
     c02f 28              55            plp
     c030 60              56    90$     rts
                          57
```

```
error addr  code          seq   source statement

      c031 c04f            58   port    .word mirq.1
      c033 c04c            59           .word mirq.2
                           60
                           61
      c035 ad 0315         62   remove  lda iirq+1      ;remove irq wedge
      c038 c9 c0           63           cmp #>mirq.1
      c03a d0 0f           64           bne 90$         ;...branch if already removed!
      c03c 08              65           php
      c03d 78              66           sei
      c03e ad c0f0         67           lda iirq2       ;restore saved indirect
      c041 8d 0314         68           sta iirq
      c044 ad c0f1         69           lda iirq2+1
      c047 8d 0315         70           sta iirq+1
      c04a 28              71           plp
      c04b 60              72   90$     rts
                           73
                           74
                           75
      c04c a9 80           76   mirq.2  lda #$80        ;port2 mouse scan
      c04e 2c              77           .byte $2c
                           78
      c04f a9 40           79   mirq.1  lda #$40        ;port1 mouse scan
                           80
      c051 20 c0ba         81           jsr setpot      ;configure cia per .a
                           82
      c054 ad d419         83           lda potx        ;get delta values for x
      c057 ac c0f2         84           ldy opotx
      c05a 20 c090         85           jsr movchk
      c05d 8c c0f2         86           sty opotx
                           87
      c060 18              88           clc             ;modify low order x position
      c061 6d d000         89           adc xpos
      c064 8d d000         90           sta xpos
      c067 8a              91           txa
      c068 69 00           92           adc #$00
      c06a 29 01           93           and #%00000001
      c06c 4d d010         94           eor xposmsb
      c06f 8d d010         95           sta xposmsb
                           96
      c072 ad d41a         97           lda poty        ;get delta value for y
      c075 ac c0f3         98           ldy opoty
      c078 20 c090         99           jsr movchk
      c07b 8c c0f3        100           sty opoty
                          101
      c07e 38             102           sec             ;modify y position (decrease y for increase in pot)
      c07f 49 ff          103           eor #$ff
      c081 6d d001        104           adc ypos
      c084 8d d001        105           sta ypos
                          106
      c087 ae c0f6        107           ldx ciasave     ;restore keyboard
      c08a 8e dc00        108           stx cia
                          109
      c08d 6c c0f0        110   90$     jmp (iirq2)     ;continue w/ irq operation
                          111
                          112
                          113
                          114   ; movchk
```

```
error addr  code          seq   source statement

                          115   ;       entry   y = old value of pot register
                          116   ;               a = currrent value of pot register
                          117   ;       exit    y = value to use for old value
                          118   ;               x,a = delta value for position
                          119   ;
                          120
      c090 8c c0f5        121   movchk  sty oldvalue    ;save old & new values
      c093 8d c0f4        122           sta newvalue
      c096 a2 00          123           ldx #0          ;preload x w/ 0
                          124
      c098 38             125           sec             ;a = mod64(new-old)
      c099 ed c0f5        126           sbc oldvalue
      c09c 29 7f          127           and #%01111111
      c09e c9 40          128           cmp #%01000000  ;if a > 0
      c0a0 b0 07          129           bcs 50$
      c0a2 4a             130           lsr a           ;   then a = a/2
      c0a3 f0 12          131           beq 80$         ;      if a <> 0
      c0a5 ac c0f4        132           ldy newvalue    ;         then y = newvalue
      c0a8 60             133           rts             ;              return
                          134
      c0a9 09 c0          135   50$     ora #%11000000  ;   else or-in high order bits
      c0ab c9 ff          136           cmp #$ff        ;      if a <> -1
      c0ad f0 08          137           beq 80$
      c0af 38             138           sec             ;         then a = a/2
      c0b0 6a             139           ror a
      c0b1 a2 ff          140           ldx #$ff        ;              x = -1
      c0b3 ac c0f4        141           ldy newvalue    ;              y = newvalue
      c0b6 60             142           rts             ;              return
                          143
      c0b7 a9 00          144   80$     lda #0          ;a = 0
      c0b9 60             145           rts             ;return w/ y = old value
                          146
                          147
                          148
      c0ba ae dc00        149   setpot  ldx cia         ;save keyboard lines
      c0bd 8e c0f6        150           stx ciasave
                          151
      c0c0 8d dc00        152           sta cia         ;connect appropriate port to sid
                          153
      c0c3 a2 04          154           ldx #4
      c0c5 a0 c7          155           ldy #$c7        ;delay 4ms to let lines settle & get sync-ec
      c0c7 88             156   10$     dey
      c0c8 d0 fd          157           bne 10$
      c0ca ca             158           dex
      c0cb d0 fa          159           bne 10$
      c0cd 60             160           rts
                          161
                          162           .end
   0 errors detected
```

symbol table
(blank) = label, (=) = symbol, (+)= multibly defined

cia	=dc00	cia.ddr	=dc02	ciasave	c0f6	iirq	=0314	iirq2	c0f0	install.1	c009	install.2	c00c
mirq.1	c04f	mirq.2	c04c	movchk	c090	newvalue	c0f4	oldvalue	c0f5	opotx	c0f2	opoty	c0f3
port	c031	potx	=d419	poty	=d41a	remove	c035	setpot	c0ba	sid	=d400	vic	=d000
xpos	=d000	xposmsb	=d010	ypos	=d001								

cross reference
(<#> = definition, <$> = write, <blank> = read)

```
cia         =dc00    9#   108$   149    152$
cia.ddr     =dc02   10#
ciasave      c0f6   25#   107    150$
iirq        =0314    6#    40     46     48     52$    54$    62     68$    70$
iirq2        c0f0   20#    47$    49$    67     69    110
install.1    c009   30     35#
install.2    c00c   31     38#
mirq.1       c04f   41     58     63     79#
mirq.2       c04c   59     76#
movchk       c090   85     99    121#
newvalue     c0f4   23#   122$   132    141
oldvalue     c0f5   24#   121$   126
opotx        c0f2   21#    84     86$
opoty        c0f3   22#    98    100$
port         c031   51     53     58#
potx        =d419   11#    83
poty        =d41a   12#    97
remove       c035   32     62#
setpot       c0ba   81    149#
sid         =d400    8#    11     12
vic         =d000    7#    14     15     16
xpos        =d000   14#    89     90$
xposmsb     =d010   16#    94     95$
ypos        =d001   15#   104    105$
```

```
error addr  code          seq   source statement

                           1    ;       1351 proportional mouse driver for the c128
                           2    ;
                           3    ;       commodore business machines, inc.   27oct86
                           4    ;               by hedley davis and fred bowen
                           5
          =0314            6    iirq    = $0314
          =d000            7    vic     = $d000
          =d400            8    sid     = $d400
          =dc00            9    cia     = $dc00
          =dc02           10    cia.ddr = $dc02
          =d419           11    potx    = sid+$19
          =d41a           12    poty    = sid+$1a
                          13
          =117e           14    active  = $117e         ;basic7.0 active sprite flag (0=inactive)
                          15
          =11d6           16    vicdata = $11d6         ;basic7.0 copy of vic register image
          =11d6           17    xpos    = vicdata+$00   ;x position (lsb)
          =11d7           18    ypos    = vicdata+$01   ;y position
          =11e6           19    xposmsb = vicdata+$10   ;x position (msb)
                          20
          =18f0           21            *=$18f0
                          22
     18f0 =18f2           23    iirq2           *=*+2
     18f2 =18f3           24    opotx           *=*+1
     18f3 =18f4           25    opoty           *=*+1
     18f4 =18f5           26    newvalue        *=*+1
     18f5 =18f6           27    oldvalue        *=*+1
     18f6 =18f7           28    ciasave         *=*+1
                          29
                          30
          =1800           31            * = $1800
                          32
     1800 4c 1809         33            jmp install.1   ;install mouse in port 1
     1803 4c 180c         34            jmp install.2   ;install mouse in port 2
     1806 4c 1835         35            jmp remove      ;remove mouse wedge
                          36
                          37
     1809 a2 00           38    install.1       ldx #0          ;port 1 mouse
     180b 2c              39                    .byte $2c
                          40
     180c a2 02           41    install.2       ldx #2          ;port 2 mouse
                          42
     180e ad 0315         43            lda iirq+1      ;install irq wedge
     1811 c9 18           44            cmp #>mirq.1
     1813 f0 1b           45            beq 90$         ;...branch if already installed!
     1815 08              46            php
     1816 78               7            sei
                          48
     1817 ad 0314         49            lda iirq        ;save current irq indirect for our exit
     181a 8d 18f0         50            sta iirq2
     181d ad 0315         51            lda iirq+1
     1820 8d 18f1         52            sta iirq2+1
                          53
     1823 bd 1831         54            lda port,x      ;point irq indirect to mouse driver
     1826 8d 0314         55            sta iirq
     1829 bd 1832         56            lda port+1,x
     182c 8d 0315         57            sta iirq+1
```

```
error addr  code          seq   source statement

      182f 28              58           plp
      1830 60              59   90$     rts
                           60
      1831 184f            61   port    .word mirq.1
      1833 184c            62           .word mirq.2
                           63
                           64
      1835 ad 0315         65   remove  lda iirq+1      ;remove irq wedge
      1838 c9 18           66           cmp #>mirq.1
      183a d0 0f           67           bne 90$         ;...branch if already removed!
      183c 08              68           php
      183d 78              69           sei
      183e ad 18f0         70           lda iirq2       ;restore saved indirect
      1841 8d 0314         71           sta iirq
      1844 ad 18f1         72           lda iirq2+1
      1847 8d 0315         73           sta iirq+1
      184a 28              74           plp
      184b 60              75   90$     rts
                           76
                           77
                           78
      184c a9 80           79   mirq.2  lda #$80        ;port2 mouse scan
      184e 2c              80           .byte $2c
                           81
      184f a9 40           82   mirq.1  lda #$40        ;port1 mouse scan
                           83
      1851 20 18bc         84           jsr setpot      ;configure cia per .a
      1854 d0 39           85           bne 90$         ;...oops- basic in control
                           86
      1856 ad d419         87           lda potx        ;get delta values for x
      1859 ac 18f2         88           ldy opotx
      185c 20 1892         89           jsr movchk
      185f 8c 18f2         90           sty opotx
                           91
      1862 18              92           clc             ;modify low order xposition
      1863 6d 11d6         93           adc xpos
      1866 8d 11d6         94           sta xpos
      1869 8a              95           txa
      186a 69 00           96           adc #$00
      186c 29 01           97           and #%00000001
      186e 4d 11e6         98           eor xposmsb
      1871 8d 11e6         99           sta xposmsb
                          100
      1874 ad d41a        101           lda poty        ;get delta value for y
      1877 ac 18f3        102           ldy opoty
      187a 20 1892        103           jsr movchk
      187d 8c 18f3        104           sty opoty
                          105
      1880 38             106           sec             ;modify y position (decrease y for increase in pot)
      1881 49 ff          107           eor #$ff
      1883 6d 11d7        108           adc ypos
      1886 8d 11d7        109           sta ypos
                          110
      1889 ae 18f6        111           ldx ciasave     ;restore keyboard
      188c 8e dc00        112           stx cia
                          113
      188f 6c 18f0        114   30$     jmp (iirq2)     ;continue w/ irq operation
```

```
error addr  code          seq   source statement

                          115
                          116
                          117
                          118   ; movchk
                          119   ;       entry   y = old value of pot register
                          120   ;               a = currrent value of pot register
                          121   ;       exit    y = value to use for old value
                          122   ;               x,a = delta value for position
                          123   ;
                          124
     1892 8c 18f5         125   movchk  sty oldvalue    ;save old & new values
     1895 8d 18f4         126           sta newvalue
     1898 a2 00           127           ldx #0          ;preload x w/ 0
                          128
     189a 38              129           sec             ;a = mod64(new-old)
     189b ed 18f5         130           sbc oldvalue
     189e 29 7f           131           and #%01111111
     18a0 c9 40           132           cmp #%01000000  ;if a > 0
     18a2 b0 07           133           bcs 50$
     18a4 4a              134           lsr a           ;   then a = a/2
     18a5 f0 12           135           beq 80$         ;      if a <> 0
     18a7 ac 18f4         136           ldy newvalue    ;         then y = newvalue
     18aa 60              137           rts             ;              return
                          138
     18ab 09 c0           139   50$     ora #%11000000  ;   else or-in high order bits
     18ad c9 ff           140           cmp #$ff        ;      if a <> -1
     18af f0 08           141           beq 80$
     18b1 38              142           sec             ;         then a = a/2
     18b2 6a              143           ror a
     18b3 a2 ff           144           ldx #$ff        ;              x = -1
     18b5 ac 18f4         145           ldy newvalue    ;              y = newvalue
     18b8 60              146           rts             ;              return
                          147
     18b9 a9 00           148   80$     lda #0          ;a = 0
     18bb 60              149           rts             ;return w/ y = old value
                          150
                          151
                          152
     18bc ae 117e         153   setpot  ldx active      ;is basic moving sprite 1?
     18bf d0 13           154           bne 20$         ;...yes, we'll leave it alone (why not?)
                          155
     18c1 ae dc00         156           ldx cia         ;save keyboard lines
     18c4 8e 18f6         157           stx ciasave
                          158
     18c7 8d dc00         159           sta cia         ;connect appropriate port to sid
                          160
     18ca a2 04           161           ldx #4
     18cc a0 c7           162           ldy #$c7        ;delay 4ms to let lines settle & get sync-ed
     18ce 88              163   10$     dey
     18cf d0 fd           164           bne 10$
     18d1 ca              165           dex
     18d2 d0 fa           166           bne 10$
     18d4 60              167   20$     rts
                          168
                          169           .end

  0 errors detected
```

symbol table
(blank) = label, (=) = symbol, (+)= multibly defined

active	=117e	cia	=dc00	cia.ddr	=dc02	ciasave	18f6	iirq	=0314	iirq2	18f0	install.1	1809
install.2	180c	mirq.1	184f	mirq.2	184c	movchk	1892	newvalue	18f4	oldvalue	18f5	opotx	18f2
opoty	18f3	port	1831	potx	=d419	poty	=d41a	remove	1835	setpot	18bc	sid	=d400
vic	=d000	vicdata	=11d6	xpos	=11d6	xposmsb	=11e6	ypos	=11d7				

cross reference
(<#> = definition, <$> = write, <blank> = read)

```
active    =117e   14#   153
cia       =dc00    9#   112$   156    159$
cia.ddr   =dc02   10#
ciasave    18f6   28#   111    157$
iirq      =0314    6#    43     49     51     55$    57$    65     71$    73$
iirq2      18f0   23#    50$    52$    70     72    114
install.1  1809   33     38#
install.2  180c   34     41#
mirq.1     184f   44     61     66     82#
mirq.2     184c   62     79#
movchk     1892   89    103    125#
newvalue   18f4   26#   126$   136    145
oldvalue   18f5   27#   125$   130
opotx      18f2   24#    88     90$
opoty      18f3   25#   102    104$
port       1831   54     56     61#
potx      =d419   11#    87
poty      =d41a   12#   101
remove     1835   35     65#
setpot     18bc   84    153#
sid       =d400    8#    11     12
vic       =d000    7#
vicdata   =11d6   16#    17     18     19
xpos      =11d6   17#    93     94$
xposmsb   =11e6   19#    98     99$
ypos      =11d7   18#   108    109$
```

CHAPTER 11

1571 Burst Subroutines

Hardware: C128 with a 1571 disk drive

1) A set of subroutines for use with the 1571 Burst mode commands

Known bugs: None

This code includes a set of subroutines with jump table that make it easier to use the Burst mode commands of the 1571. Four commands are supported: read, write, query disk and inquire format.

In addition, there are 3 support routines: open channel, memory compare and send command. See the source listing for details on how to call these routines.

For an example of a BASIC program that calls these routines, see the file named "1571 BURST.BAS" on the release disks. This is a 1571 2-drive backup program.

```
error addr  code        seq  source statement

                          1  ;1571 burst.src
                          2  ;*****************************************************************************
                          3  ;*                                                                           *
                          4  ;*                      ------------------------                             *
                          5  ;*  ------------------ 1571  BURST  SUBROUTINES  ---------------------       *
                          6  ;*                      ------------------------                             *
                          7  ;*                                                                           *
                          8  ;*    The following burst subroutines have been designed for use in          *
                          9  ;* BASIC and machine language programs.                                      *
                         10  ;*                                                                           *
                         11  ;*   If you are programming in assembly language, you may use the           *
                         12  ;* routines as is, or you may modify them to suit your own purpose.          *
                         13  ;*                                                                           *
                         14  ;*    If you are programming in C128 BASIC, you should follow the           *
                         15  ;* procedure shown below for calling the burst subroutines and passing       *
                         16  ;* the appropriate parameters.  (Note: you cannot use burst commands         *
                         17  ;* with the C64.)                                                            *
                         18  ;*                                                                           *
                         19  ;*    BLOAD the binary files containing the routines.                        *
                         20  ;*                                                                           *
                         21  ;*    Assign a logical file number to the command channel to the 1571.       *
                         22  ;*                                                                           *
                         23  ;*    Open the command channel within your BASIC program.                    *
                         24  ;*                                                                           *
                         25  ;*    Execute the BANK 0 command.  This will tell BASIC to PEEK and         *
                         26  ;*    POKE to the RAM under the BASIC ROM.                                   *
                         27  ;*                                                                           *
                         28  ;*    POKE the logical file number to LF.                                    *
                         29  ;*                                                                           *
                         30  ;*    POKE the appropriate parameters into the proper variable               *
                         31  ;*    locations, and SYS to the desired routine.                             *
                         32  ;*                                                                           *
                         33  ;* All of the BURST protocol and handshaking will be done for you.  The      *
                         34  ;* BASIC program can then PEEK any values returned.                          *
                         35  ;*                                                                           *
                         36  ;* If you are using RAM, you should keep in mind that you may only use      *
                         37  ;* RAM between your BASIC text program and $C000.  The KERNEL and I/O        *
                         38  ;* will need the space after $C000.  BASIC programs normally start at       *
                         39  ;* $1C00. If you enable bit-map graphics, then your program will start      *
                         40  ;* at $4000.  The binary files containing these BURST routines load at      *
                         41  ;* $1300, so that they are in a safe place below the BASIC text area.       *
                         42  ;*                                                                           *
                         43  ;* If you want to make your program intelligent, the pointers to the        *
                         44  ;* exact beginning and end of the BASIC program are in locations $002D      *
                         45  ;* and $1210 respectively. For the most part, however, there is no need     *
                         46  ;* to look at those values.  As a general rule, if you use memory          *
                         47  ;* by working backwards from $C000, you'll be OK.  When you PEEK and       *
                         48  ;* POKE this memory from BASIC, be sure to execute the BANK 0 command.     *
                         49  ;* This tells BASIC to PEEK and POKE to the RAM under the BASIC ROM.        *
                         50  ;*                                                                           *
                         51  ;* NOTE:  There is no BURST FORMAT routine provided here.  This is          *
                         52  ;*        because you can easily accomplish BURST formatting from          *
                         53  ;*        BASIC.  For example, the following BASIC commands will format    *
                         54  ;*        physical tracks 10 through 20 of the disk with 5 1024 byte       *
                         55  ;*        MFM sectors per side (sectors 1-5).                              *
                         56  ;*                                                                           *
                         57  ;*        OPEN 1,8,15                                                      *
```

```
error addr  code          seq   source statement

                          38    ;*       PRINT#1,"U0"CHR$(CB)CHR$(129)CHR$(0)CHR$(3)CHR$(10)CHR$(5)    *
                          39    ;*               CHR$(10)CHR$(10);                                      *
                          60    ;*                                                                      *
                          61    ;* Note the use of the semicolon (;) at the end of the statement.  This *
                          62    ;* is very important!  If there was no semicolon, the CBM would        *
                          63    ;* send a carriage return after the last parameter.  Since the 1571    *
                          64    ;* counts the number of bytes sent to determine the number of optional  *
                          65    ;* parameters that are being sent, it would misinterpret the carriage   *
                          66    ;* return as the next optional parameter.  In this case, it would be    *
                          67    ;* fill byte.                                                           *
                          68    ;*                                                                      *
                          69    ;************************************************************************
                          70
                          71    ;************************************************************************
                          72    ;*                                                                      *
                          73    ;*                    VARIABLE DECLARATIONS                             *
                          74    ;*                                                                      *
                          75    ;* These variables are parameters passed between a BURST routine and    *
                          76    ;* its calling program.                                                 *
                          77    ;*                                                                      *
                          78    ;************************************************************************
                          79
      =1300               80            *=$1300
                          81
1300 00                   82    STATUS  .byte 0               ; status byte
1301 08                   83    DEV     .byte 8               ; device number
1302 08                   84    LF      .byte 8               ; logical file number
1303 =1304                85    TRACK   *=*+1                 ; track
1304 =1305                86    SECTOR  *=*+1                 ; sector
1305 =1306                87    NUMSEC  *=*+1                 ; Number of sectors.
1306 =1308                88    BUFLOC  *=*+2                 ; Page # of buffer to get/put data.
1308 =1309                89    SECSIZE *=*+1                 ; Sector size (1=128, 2=512, 4=1024)
1309 =130a                90    SIDE    *=*+1                 ; Physical side of the disk (0 or 1).
130a =130b                91    MINSEC  *=*+1                 ; Minimum logical sector found in QUERY.
130b =130c                92    MAXSEC  *=*+1                 ; Maximum logical sector found in QUERY.
130c =130d                93    INTLV   *=*+1                 ; Physical interleave found in QUERY.
130d =130e                94    FLAG    *=*+1                 ; Empty track flag.
                          95                                  ;  This flag is used to indicate that the
                          96                                  ;  track or data just read contains all 0's.
                          97                                  ;  This is handy in some cases, such as
                          98                                  ;  during a disk copy program.  When a disk
                          99                                  ;  is formatted, the sectors are filled with
                         100                                  ;  0's.  If a sector to be copied contains
                         101                                  ;  all 0's, then we don't bother to
                         102                                  ;  write it to the destination disk (which
                         103                                  ;  can end up saving a great deal of time!).
                         104
                         105   ;************************************************************************
                         106   ;*                                                                      *
                         107   ;* The following are the remaining variables that are used internally   *
                         108   ;* by the BURST routines.                                               *
                         109   ;*                                                                      *
                         110   ;************************************************************************
                         111
130e                     112   cmdline
130e 55 30               113           .byte "U0"              ; Burst prefix.
     =131a               114           *=*+10                  ; Parameter space for burst command.
```

```
error addr  code         seq   source statement

      131a =131b               cmdlen  *=*+1              ; Length of the command string (# of bytes).
      131b =131c               oldclk  *=*+1              ; Status of clock line.
      131c =131d               temp    *=*+1

           =1330                       *=$1330
      1330 31 35 37                    .byte "1571 greg berlin" ; string used to check if assembly
      1333 31 20 47
      1336 52 45 47
      1339 20 42 45
      133c 52 4c 49
      133f 4e
                                                          ; language is in memory.

                               ;********************************************************************
                               ;*                                                                  *
                               ;*                          JUMP TABLE                              *
                               ;*                                                                  *
                               ;* This jump table gives the locations of each of the BURST routines.  *
                               ;* The jump table positions will never change, even if the routines  *
                               ;* below are modified.  Always SYS to this table from BASIC, or JSR or *
                               ;* JMP to this table from assembly language.                         *
                               ;*                                                                  *
                               ;********************************************************************

           =1340                       *=$1340

      1340                     J←INQUIRE←FORMAT
      1340 4c 146a                     jmp INQUIRE←FORMAT
      1343                     J←PHYSICAL←READ
      1343 4c 1355                     jmp PREAD
      1346                     J←PHYSICAL←WRITE
      1346 4c 13c6                     jmp PWRITE
      1349                     J←QUERY←FORMAT
      1349 4c 1493                     jmp QUERY←FORMAT
      134c                     J←COMPARE←MEMORY
      134c 4c 151b                     jmp COMPARE←MEMORY
      134f                     J←SENDCMD
      134f 4c 1555                     jmp SENDCMD
      1352                     J←OPEN←CHANNEL
      1352 4c 156e                     jmp OPEN←CHANNEL

                               ;********************************************************************
                               ;*                                                                  *
                               ;*                       MEMORY LOCATIONS                           *
                               ;*                                                                  *
                               ;* These are the memory locations of some of the routines used by the  *
                               ;* BURST subroutines.                                               *
                               ;*                                                                  *
                               ;********************************************************************

           =ffc9               chkout=$ffc9               ; kernel channel output routine
           =ffcc               clrchn=$ffcc               ; kernel clear channel routine
           =ffba               setlfs=$ffba               ; kernel set logical file number routine
           =ffbd               setnam=$ffbd               ; kernel set filename routine
           =ffd2               bsout =$ffd2               ; kernel basic input/output routine
           =ffc0               open =$ffc0                ; kernel open logical file for I/O routine
           =ff47               spin←out=$ff47             ; Set up fast serial for input or output
```

```
error addr  code          seq   source statement

                          167                              ;  SEC for output, CLC for input.
                          168
                          169   ;*************************************************************************
                          170   ;*                                                                       *
                          171   ;* These addresses are memory locations used by the BURST subroutines.   *
                          172   ;*                                                                       *
                          173   ;*************************************************************************
                          174
          =dd00           175   d2ora =$dd00                   ; C128 serial port location
          =0010           176   clkout=$10                     ; slow serial clock output bit mask
          =0040           177   clkin =$40                     ; slow serial clock input bit mask
          =dc0d           178   d1icr =$dc0d                   ; 6526 CIA interrupt control register
          =dc0c           179   d1sdr =$dc0c                   ; 6526 CIA serial data register
          =00fa           180   buffer=$fa                     ; zero page pointer variable
          =00fc           181   buffer2=$fc                    ; zero page pointer variable
                          182
                          183   ;*************************************************************************
                          184   ;*                                                                       *
                          185   ;*                      BURST COMMAND PRIMITIVES                         *
                          186   ;*                                                                       *
                          187   ;* These are the BURST commands as the 1571 sees them.                   *
                          188   ;*                                                                       *
                          189   ;*************************************************************************
                          190
          =0000           191   FBURSTRD =$00                  ; Burst read.
          =0002           192   FBURSTWR =$02                  ; Burst write.
          =0004           193   BURST=INQUIRE  =$04            ; Burst inquire.
          =0086           194   BURST=QUERY =$86               ; Query disk format.
                          195
                          196   ;*************************************************************************
                          197   ;*                                                                       *
                          198   ;*                         THE BURST ROUTINES                            *
                          199   ;*                                                                       *
                          200   ;*************************************************************************
                          201
      1355                202   FREAD          ;This BURST routine reads sectors from device LF.
                          203                  ;
                          204                  ; Required Parameters:
                          205                  ;       LF      Logical file number to read from.
                          206                  ;       TRACK   Track to be read from.
                          207                  ;       SECTOR  Sector to be read from.
                          208                  ;       BUFLOC  Pointer to the starting location of the
                          209                  ;               buffer in RAM bank 0.  As the characters
                          210                  ;               are read, they are put in this buffer.
                          211                  ;       SECSIZE Physical sector size (1=256, 2=512, 4=1024)
                          212                  ;       NUMSEC  The number of sectors to be read.
                          213                  ;       SIDE    Physical side of the disk to read from,
                          214                  ;               0 or 1.  SIDE is ignored if the disk is GCR.
                          215                  ;
                          216                  ; Returned Parameters:
                          217                  ;       STATUS  Status byte returned after read is attempted.
                          218
      1355 a9 00          219           lda #FBURSTRD          ;Physical burst read command.
      1357 ae 1309        220           ldx SIDE               ; Check which side to read from.
      135a f0 02          221           beq 1$
      135c 09 10          222           ora #$10               ; If side 1, then set bit in the command byte.
      135e 8d 1310        223   1$      sta  cmdline+2
```

```
error addr  code          seq   source statement

                          224
                          225
      1361                226   READ
                          227
      1361 ad ff00        228         lda $ff00           ;Save old MMU setup.
      1364 48             229         pha
                          230
      1365 a9 0e          231         lda #$0e            ;Set MMU for RAM0,KERNEL,I/O.
      1367 8d ff00        232         sta $ff00
                          233
      136a 20 157a        234         jsr SETU0           ;Put "U0" at start of command string.
                          235
      136d ad 1303        236         lda TRACK
      1370 8d 1311        237         sta CMDLINE+3       ; track
      1373 ad 1304        238         lda SECTOR
      1376 8d 1312        239         sta CMDLINE+4       ; sector
      1379 ad 1305        240         lda NUMSEC
      137c 8d 1313        241         sta CMDLINE+5       ; Number of sectors to read.
                          242
      137f a9 06          243         lda #$06            ; Length of command string.
      1381 8d 131a        244         sta CMDLEN
      1384 20 1555        245         jsr  sendcmd        ; send cmd string
                          246
      1387 ad 1306        247         lda  BUFLOC         ; Set up zero page indirect pointer.
      138a 85 fa          248         sta  BUFFER
      138c ad 1307        249         lda  BUFLOC+1
      138f 85 fb          250         sta  BUFFER+1
                          251
      1391 a0 00          252         ldy  #0             ; clear the  empty sectors!  flag.
      1393 8c 130d        253         sty  flag
                          254
      1396 78             255         sei                 ; No irq s allowed during handshake.
                          256
      1397 2c dc0d        257         bit  d1icr          ; clear pending
                          258
      139a 20 1599        259         jsr CLK+CHNG        ;Change state of clock.
                          260
      139d ae 1308        261   1$    ldx  SECSIZE        ; Sector size gives # of pages per sector.
                          262
      13a0 20 15a2        263         jsr WAIT            ;Wait for fast byte (1st is status).
      13a3 ad dc0c        264         lda  d1sdr          ;Get status byte.
      13a6 8d 1300        265         sta STATUS
                          266
      13a9 29 0f          267         and  #15            ;Was there an error?
      13ab c9 02          268         cmp  #2             ;
      13ad b0 21          269         bcs  5$             ; branch if error occured.
                          270
      13af 20 1599        271         jsr CLK+CHNG        ;Change clock so next byte is sent.
                          272
      13b2 20 15a2        273   3$    jsr WAIT            ;Wait for the next byte.
                          274
      13b5 20 1599        275         jsr CLK+CHNG        ;Change state of clock so next byte is sent.
      13b8 ad dc0c        276         lda  D1SDR          ;Get the data byte
      13bb 91 fa          277         sta  (buffer),y     ; and save it
                          278                             ;  while next byte is being transmitted.
                          279
      13bd 0d 130d        280         ora  flag           ;Update  zero  sector flag.
```

```
error addr  code          seq   source statement

      13c0 8d 1300       281          sta  flag
                         282
      13c3 c8            283          iny                    ;Any more in this page
      13c4 d0 ec         284          bne  3$
                         285
      13c6 e6 fb         286          inc BUFFER+1
      13c8 ca            287          dex                    ;Loop for the # of pages per sector.
      13c9 d0 e7         288          bne 3$
                         289
      13cb ce 1313       290          dec CMDLINE+5          ;Loop for the number of sectors.
      13ce d0 cd         291          bne 1$
                         292
      13d0 58            293    5$    cli
                         294
      13d1 68            295          pla                    ;Restore MMU to old configuration.
      13d2 8d ff00       296          sta $ff00
                         297
      13d5 60            298          rts
                         299
                         300    ;*************************************************************************
                         301
      13d6               302    FWRITE         ; This BURST subroutine writes physical sectors to device LF.
                         303                   ;
                         304                   ; Required Parameters:
                         305                   ;       LF      Logical file number to write to.
                         306                   ;       TRACK   Track to be written to.
                         307                   ;       SECTOR  Sector to be written to.
                         308                   ;       BUFLOC  Location of the beginning of the 1128 buffer
                         309                   ;               where the characters are to be read from.
                         310                   ;       SECSIZE Physical sector size (1=128, 2=256, 4=1024).
                         311                   ;       NUMSEC  The number of sectors to be written.
                         312                   ;       SIDE    Physical side of the disk to write to.
                         313                   ;               0 or 1.  SIDE is ignored if the disk is GCR.
                         314                   ;
                         315                   ; Returned Parameters:
                         316                   ;       STATUS  Status byte returned after write is attempted.
                         317
      13d6 a9 02         318          lda #FBURSTWR          ;Physical burst write command.
      13d8 ae 1309       319          ldx SIDE               ; Check which side to write to.
      13db f0 02         320          beq 1$
      13dd 09 10         321          ora #$10               ; If side 1, then set bit in the command byte.
      13df 8d 1310       322    1$    sta  cmdline+2
                         323
                         324
      13e2               325    WRITE
                         326
      13e2 ad ff00       327          lda $ff00              ;Save old MMU setup.
      13e5 48            328          pha
                         329
      13e6 a9 0e         330          lda #$0e               ;Set MMU for RAM0,SYSCFG,I/O.
      13e8 8d ff00       331          sta $ff00
                         332
      13eb 20 157a       333          jsr SETU0              ;Put "U0" at start of command string.
      13ee ad 1303       334          lda TRACK
      13f1 8d 1311       335          sta CMDLINE+3          ; track
      13f4 ad 1304       336          lda SECTOR
      13f7 8d 1312       337          sta CMDLINE+4          ; sector
```

```
error addr  code          seq   source statement

      13fa ad 1305        338           lda NUMSEC
      13fd 8d 1313        339           sta CMDLINE+5          ; Number of sectors to write.
                          340
      1400 a9 06          341           lda #$06
      1402 8d 131a        342           sta CMDLEN             ;Command length.
      1405 20 1555        343           jsr  sendcmd           ; send cmd string
                          344
      1408 ad 1306        345           lda  BUFLOC            ; Set up zero page indirect pointer.
      140b 85 fa          346           sta  BUFFER
      140d ad 1307        347           lda BUFLOC+1
      1410 85 fb          348           sta BUFFER+1
                          349
      1412 a9 40          350           lda  #clkin            ;Initial clock status.
      1414 8d 131b        351           sta  oldclk
                          352
      1417 a0 00          353           ldy  #0
      1419 78             354           sei                    ; no irq s during burst handshake
                          355
      141a ae 1308        356     1$    ldx  SECSIZE           ; Sector size gives # of pages per sector.
                          357
      141d 38             358           sec                    ;Turn fast serial to output mode.
      141e 20 ff47        359           jsr  spin+out
                          360
      1421 ad dd00        361     2$    lda  d2pra             ;Wait for state change.
      1424 4d 131b        362           eor  oldclk
      1427 29 40          363           and  #clkin
      1429 f0 f6          364           beq  2$
                          365
      142b 4d 131b        366           eor  oldclk            ;Change status of CLOCK.
      142e 8d 131b        367           sta  oldclk
                          368
      1431 b1 fa          369           lda  (buffer),y        ; get data
      1433 8d dc0c        370           sta  d1sdr             ;  & send it
                          371
      1436 20 15a2        372           jsr WAIT               ;Wait for the byte to be transmitted.
                          373
      1439 c8             374           iny
      143a d0 e5          375           bne  2$                ;Any more left in this page!
                          376
      143c e6 fb          377           inc  buffer+1
      143e ca             378           dex                    ;Loop for the # of pages per sector.
      143f d0 e0          379           bne  2$
                          380
      1441 18             381           clc                    ;Turn around to input mode to get STATUS.
      1442 20 ff47        382           jsr  spin+out
                          383
      1445 2c dc0d        384           bit  d1icr             ; clear pending
                          385
      1448 20 1585        386           jsr  clklo             ; set clock low when ready for status
                          387
      144b 20 15a2        388           jsr WAIT               ;Wait for the byte to be shifted in.
      144e ad dc0c        389           lda  d1sdr             ;Get the status byte.
      1451 8d 1300        390           sta STATUS             ;Save it.
      1454 48             391           pha
      1455 20 1590        392           jsr  clkhi             ;Release the slow clock line.
      1458 68             393           pla
                          394
```

```
error addr  code        source statement

      1459 29 0f               and  #15              ;Check for any error.
      145b c9 02               cmp  #2
      145d b0 05               bcs  7$               ;  branch if there was an error.

      145f ce 1313             dec  CMDLINE+5        ;Loop for the number of sectors.
      1462 d0 b6               bne  1$

      1464 58          7$      cli

      1465 68                  pla                   ;Restore old memory configuration.
      1466 8d ff00             sta $ff00

      1469 60                  rts

                       ;*******************************************************************

      146a             INQUIRE+FORMAT  ; This BURST subroutine sends an INQUIRE DISK command to
                                       ; drive indicated by LF.
                                       ;
                                       ; Required Parameters:
                                       ;       LF      Logical file number of device to inquire.
                                       ;       SIDE    Physical side of the disk to inquire about.
                                       ;               0 or 1.  SIDE is ignored if the disk is GCR.
                                       ;
                                       ; Returned Parameters:
                                       ;       STATUS  Status byte returned.

      146a 20 157a             jsr SETU0             ;Put "U0" at start of command string.
      146d a9 04               lda  #BURST+INQUIRE   ; inquire burst command
      146f 8d 1310             sta  cmdline+2

      1472 ae 1309             ldx SIDE              ; Check which side to check.
      1475 f0 02               beq 1$
      1477 09 10               ora #$10              ; If side 1, then set bit in the command byte.

      1479 a9 03       1$      lda #$03              ; length of command.
      147b 8d 131a             sta CMDLEN
      147e 20 1555             jsr  sendcmd          ; send cmd string

      1481 78                  sei                   ;Disable interrupts during handshake.

      1482 2c dc0d             bit D1ICR             ;Clear any byte ready that's pending.

      1485 20 1599             jsr CLK+CHNG          ;Change clock so 1571 sends next.
      1488 20 15a2             jsr WAIT              ;Wait for the byte to be shifted in.
      148b ad dc0c             lda D1SDR             ;Get the status byte.
      148e 8d 1300             sta STATUS            ;Save it off.

      1491 58                  cli
      1492 60                  rts

                       ;*******************************************************************

      1493             QUERY+FORMAT    ; This BURST subroutine sends a QUERY DISK FORMAT
                                       ; command to drive indicated by LF.
                                       ;
                                       ; Required Parameters:
```

```
error addr  code        seq   source statement

                        452                 ;       LF      Logical file number of device to query.
                        453                 ;       TRACK   Physical track number to query.
                        454                 ;       SIDE    Physical side of the disk to query.
                        455                 ;               0 or 1.  SIDE is ignored if the disk
                        456                 ;               is GCR.
                        457                 ;
                        458                 ; Returned Parameters:
                        459                 ;       NUMSEC  Number of sectors found on the track.
                        460                 ;       TRACK   Logical track number found in the sector headers.
                        461                 ;       MINSEC  Minimum logical sector number found
                        462                 ;               in the sector headers.
                        463                 ;       MAXSEC  Maximum logical sector number found in the
                        464                 ;               sector headers.
                        465                 ;       INTLV   Physical interleave between sectors.
                        466                 ;       STATUS  This is the byte that QUERY+FORMAT returns.
                        467                 ;               If an error was encountered in compiling
                        468                 ;               this information, then none of the
                        469                 ;               returned parameters are valid except STATUS.
                        470
                        471
     1493 20 157a       472                 jsr SETUO               ;Put "U0" at start of command string.
                        473
     1496 a9 86         474                 lda  #BURST+QUERY       ;QUERY DISK burst command
                        475
     1498 ae 1309       476                 ldx SIDE                ; Set the side bit accordingly.
     149b d0 02         477                 bne 4$
     149d 09 10         478                 ora #$10
                        479
     149f 8d 1310       480     4$          sta cmdline+2
                        481
     14a2 ad 1303       482                 lda TRACK               ; Physical track number.
     14a5 8d 1311       483                 sta cmdline+3
     14a8 a9 04         484                 lda #$04                ; length of command.
     14aa 8d 131a       485                 sta CMDLEN
     14ad 20 1555       486                 jsr  sendcmd            ; send cmd string
                        487
     14b0 78            488                 sei                     ;Disable interrupts during handshake.
                        489
     14b1 2c dc0d       490                 bit D1ICR               ;Clear any byte ready that is pending.
                        491
     14b4 20 1599       492                 jsr CLK+CHNG            ;Change state of clock so 1571 sends next.
     14b7 20 15a2       493                 jsr WAIT                ;Wait for the first status byte.
     14ba ad dc0c       494                 lda D1SDR               ;Get the status byte (status of track 0).
     14bd 8d 1300       495                 sta STATUS              ;Save it off.
                        496
     14c0 29 0f         497                 and #$0f                ;Was there an error?
     14c2 c9 02         498                 cmp #2
     14c4 b0 53         499                 bcs 5$                  ; branch if there was an error.
                        500
     14c6 ad 1300       501                 lda STATUS              ;Is the format GCR (if so no bytes follow)?
     14c9 10 4e         502                 bpl 5$
                        503
     14cb 20 1599       504                 jsr CLK+CHNG            ;Change state of clock, so 1571 sends next.
     14ce 20 15a2       505                 jsr WAIT                ;Wait for next status byte to be ready.
     14d1 ad dc0c       506                 lda D1SDR               ;Get it (status of track [TRACK]).
     14d4 8d 1300       507                 sta STATUS              ;Save it.
                        508
```

```
error addr  code          seq   source statement

      14d7 29 0f          509           and #$0f              ;Was there an error in compiling MFM into
      14d9 c9 02          510           cmp #2
      14db b0 3c          511           bcs 5$                ; branch if an error.
                          512
      14dd 20 1599        513           jsr CLK=CHNG          ;Change state of clock, so 1571 sends next.
      14e0 20 15a2        514           jsr WAIT              ;Wait for  number of sectors byte  to be ready
      14e3 ad dc0c        515           lda D1SDR             ;Get it.
      14e6 8d 1305        516           sta NUMSEC            ;Save it.
                          517
                          518
      14e9 20 1599        519           jsr CLK=CHNG          ;Change state of clock, so 1571 sends next.
      14ec 20 15a2        520           jsr WAIT              ;Wait for 'logical track #  byte to be ready.
      14ef ad dc0c        521           lda D1SDR             ;Get it.
      14f2 8d 1303        522           sta TRACK             ;Save it.
                          523
      14f5 20 1599        524           jsr CLK=CHNG          ;Change state of clock, so 1571 sends next.
      14f8 20 15a2        525           jsr WAIT              ;Wait for 'minimum sector #  byte to be ready.
      14fb ad dc0c        526           lda D1SDR             ;Get it.
      14fe 8d 1309        527           sta MINSEC            ;Save it.
                          528
      1501 20 1599        529           jsr CLK=CHNG          ;Change state of clock, so 1571 sends next.
      1504 20 15a2        530           jsr WAIT              ;Wait for 'maximum sector #  byte to be ready.
      1507 ad dc0c        531           lda D1SDR             ;Get it.
      150a 8d 130b        532           sta MAXSEC            ;Save it.
                          533
      150d 20 1599        534           jsr CLK=CHNG          ;Change state of clock, so 1571 sends next.
      1510 20 15a2        535           jsr WAIT              ;Wait for  interleave  byte to be ready.
      1513 ad dc0c        536           lda D1SDR             ;Get it.
      1516 8d 130c        537           sta INTLV             ;Save it.
                          538
      1519 58             539   5$      cli
      151a 60             540           rts
                          541
                          542   ;*************************************************************************
                          543
      151b                544   COMPARE=MEMORY  ; This subroutine compares memory blocks in C128 memory.
                          545                   ;
                          546                   ; Required Parameters:
                          547                   ;       .A      Number of pages to compare
                          548                   ;       .X      First page of first memory block
                          549                   ;       .Y      First page of second memory block
                          550                   ;
                          551                   ; Returned Parameters:
                          552                   ;       STATUS  This is the byte that COMPARE=MEMORY
                          553                   ;               returns, 0 if the two blocks are equal.
                          554
      151b 8d 131c        555           sta temp
                          556
      151e ad ff00        557           lda $ff00             ;Save old MMU setup.
      1521 48             558           pha
                          559
      1522 a9 0e          560           lda #$0e              ;Set MMU for BANK0,KERNEL,I/O.
      1524 8d ff00        561           sta $ff00
                          562
      1527 86 fb          563           stx buffer+1          ;Set up MSB of 1st memory pointer.
      1529 84 fd          564           sty buffer2+1         ;Set up MSB of 2nd memory pointer.
      152b ae 131c        565           ldx temp              ;Number of pages to compare.
```

```
error addr  code          seq   source statement

                          566
      152e a9 00          567           lda #0                  ;Set up LSB s of memory pointers.
      1530 85 fa          568           sta buffer
      1532 85 fc          569           sta buffer2
                          570
      1534 8d 1300        571           sta STATUS              ;Initialize STATUS.
                          572
      1537 a0 00          573           ldy #0
                          574
      1539 b1 fa          575   2$      lda (buffer),y
      153b d1 fc          576           cmp (buffer2),y
      153d f0 07          577           beq 1$
                          578
      153f a9 ff          579           lda #$ff                ;Not equal!  Load STATUS with nonzero.
      1541 8d 1300        580           sta STATUS
      1544 d0 0a          581           bne 99$                 ;  (branch always)
                          582
      1546 c8             583   1$      iny
      1547 d0 f0          584           bne 2$                  ;More in this page?
                          585
      1549 e6 fb          586           inc buffer+1
      154b e6 fd          587           inc buffer2+1
      154d ca             588           dex                     ;# of pages counter.
      154e d0 e9          589           bne 2$
                          590
      1550 68             591   99$     pla                     ;Restore old memory configuration.
      1551 8d ff00        592           sta $ff00
                          593
      1554 60             594           rts
                          595
                          596   ;*****************************************************************************
                          597
      1555                598   SENDCMD         ; This BURST subroutine sends a command to logical file LF.
                          599                   ;
                          600                   ; Required Parameters:
                          601                   ;       LF      Logical file number to send command to.
                          602                   ;       CMDLINE Command string to send.
                          603                   ;       CMDLEN  Length of command string.
                          604                   ;
                          605                   ; No parameter is returned.
                          606
      1555 ae 1302        607           ldx LF
      1558 20 ffc9        608           jsr  chkout             ; channel output pointed to by .X
      155b a2 00          609           ldx  #0
      155d ac 131a        610           ldy  cmdlen             ; send cmd
      1560 bd 130e        611   1$      lda  cmdline,x
      1563 20 ffd2        612           jsr  bsout
      1566 e8             613           inx
      1567 88             614           dey
      1568 d0 f6          615           bne  1$
                          616
      156a 20 ffcc        617           jsr  clrchn             ; send buffered char & eoi
      156d 60             618           rts
                          619
                          620   ;*****************************************************************************
                          621
      156e                622   OPEN+CHANNEL    ; This BURST subroutine opens up a channel to device .F
```

```
error addr  code        source statement

                                    ; and assigns it a logical file number with an optional
                                    ; secondary address.  This subroutine performs the same
                                    ; function as the BASIC open statement.
                                    ;
                                    ; Required Parameters:
                                    ;       .X      Device number
                                    ;       .A      Logical file number
                                    ;       .Y      This is the secondary address.  This
                                    ;               number should be set to $FF if no
                                    ;               secondary address is desired.
                                    ;
                                    ; Returned Parameters:
                                    ;       .A      This is the byte that OPEN+CHANNEL returns
                                    ;               1 if the routine was successful.
                                    ;               0 otherwise.  If an error was encountered,
                                    ;               then OPEN+CHANNEL will also set the CARRY
                                    ;               bit.
                                    ;

      156e 20 ffba                  jsr SETLFS              ;Setup the logical file.

      1571 a9 00                    lda #0                  ;No name/command string for OPEN (length=0).
      1573 20 ffbd                  jsr SETNAM              ;Setup the filename/command string for OPEN.

      1576 20 ffc0                  jsr OPEN                ;Open the logical file.

      1579 60                       rts

                            ;************************************************************************

      157a a9 55            SETU0   lda #85                 ; U'
      157c 8d 130e                  sta CMDLINE
      157f a9 30                    lda #48                 ; 0'
      1581 8d 130f                  sta CMDLINE+1
      1584 60                       rts


      1585                  CLKLO                   ; set clock low
      1585 48                       pha
      1586 ad dd00                  lda  d2pra
      1589 09 10                    ora  #clkout
      158b 8d dd00                  sta  d2pra
      158e 68                       pla
      158f 60                       rts

      1590                  CLKHI                   ; set clock high
      1590 ad dd00                  lda  d2pra
      1593 29 ef                    and  #$ff-clkout
      1595 8d dd00                  sta  d2pra
      1598 60                       rts

      1599                  CLK+CHNG                ;  change the state of the clock line output.
      1599 ad dd00                  lda D2PRA
      159c 49 10                    eor #clkout
      159e 8d dd00                  sta D2PRA
      15a1 60                       rts

```

```
error addr  code          seq   source statement

      15a2                680   WAIT                ;  wait for the shift register to be full or empty.
      15a2 a9 08          681   1$      lda #8
      15a4 2c dc0d        682           bit D1ICR
      15a7 f0 f9          683           beq 1$
      15a9 60             684           rts
                          685
                          686
                          687           .end
    0 errors detected
```

symbol table
(blank) = label, (=) = symbol, (+)= multibly defined

bsout	=ffd2	buffer	=00fa	buffer2	=00fc	bufloc	130b	burst←inquire	=0004
burst←query	=0086	chkout	=ffc9	clkhi	1590	clkin	=0040	clklo	1593
clrout	=001b	clk←chng	1599	clrchn	=ffcc	cmdlen	131a	cmdline	130e
compare←memory	151b	d1icr	=dc0d	d1sdr	=dc0c	d2pra	=dd00	dev	1301
flag	130d	inquire←format	146a	intlv	130c	j←compare←memory	134c	j←inquire←format	1340
j←open←channel	1352	j←physical←read	1343	j←physical←write	1346	j←query←format	1349	j←sendcmd	134f
lf	1302	maxsec	130b	minsec	130a	numsec	1305	oldclk	131b
open	=ffc0	open←channel	156e	pburstrd	=0000	pburstwr	=0002	pread	1355
pwrite	13d6	query←format	1493	read	1361	secsize	1308	sector	1304
sendcmd	1555	setlfs	=ffba	setnam	=ffbd	setup0	157a	side	1309
spin←out	=ff47	status	1300	temp	131c	track	1303	wait	15e2
write	13e2								

cross reference
(:#) = definition, <$> = write, <blank> = read)

beout	=ffd2	164#	612										
buffer	=00fa	180#	248$	250$	277$	286$	346$	348$	367	369$	563$	568$	575	584$
buffer2	=00fc	181#	564$	567$	576	587$								
bufloc	1306	88#	247	249	345	347								
burst+inquire	=0004	193#	423											
burst+query	=0006	194#	474											
chkout	=ffc9	160#	608											
clkhi	1590	392	668#											
clkin	=0040	177#	350	363										
clklo	1585	386	660#											
clkout	=0010	176#	663	670	676									
clk+chng	1599	259	271	275	438	442	504	513	519	524	529	534	674#	
clrchn	=ffcc	161#	617											
cmdlen	131a	115#	244$	342$	431$	485$	610							
cmdline	130e	112#	223$	237$	239$	241$	290$	322$	335$	337$	339$	399$	424$	463$
		483$	611	654$	656$									
compare+memory	151b	145	544#											
dlicr	=dc0d	178#	257	384	436	490	682							
dlsdr	=dc0c	179#	264	276	370$	389	440	494	506	515	521	526	531	536
dzpra	=dd00	175#	361	662	664$	669	671$	675	677$					
dev	1301	83#												
flag	130d	94#	253$	280	281$									
inquire+format	146a	137	411#											
intiv	130c	93#	537$											
j+compare+memory	134c	144#												
j+inquire+format	1340	136#												
j+open+channel	1352	148#												
j+physical+read	1343	138#												
j+physical+write	1346	140#												
j+query+format	1349	142#												
j+sendcmd	134f	146#												
lf	1302	84#	607											
maxsec	130b	92#	532$											
minsec	130a	91#	527$											
numsec	1305	87#	240	338	516$									
oldclk	131b	116#	351$	362	366	367$								
open	=ffc0	165#	647											
open+channel	156e	149	622#											
oburstrd	=0000	191#	219											
oburstwr	=0002	192#	318											
oread	1355	139	202#											
owrite	13d6	141	302#											
query+format	1493	143	448#											
read	1361	226#												
secsize	1308	89#	261	356										
sector	1304	86#	238	336										
sendcmd	1555	147	245	343	432									

cross reference
(.#. = definition, .$. = write, .blank. = read)

asout	=ffd2	164#	612											
buffer	=00fa	180#	248$	250$	277$	286$	346$	348$	368$	377$	567$	568$	579$	584$
buffer2	=00fc	181#	564$	569$	576	587$								
bufloc	1306	88#	247	249	345	347								
burst←inquire	=0004	193#	423											
burst←query	=0006	194#	474											
chkout	=ffc9	160#	608											
clkhi	1590	392	668#											
clkin	=0040	177#	350	363										
clklo	1585	386	660#											
clkout	=0010	176#	663	670	676									
clk←chng	1599	259	271	275	438	492	504	513	519	524	529	534	674#	
clrchn	=ffcc	161#	617											
cmdlen	131a	115#	244$	342$	431$	485$	610							
cmdline	130e	112#	223$	237$	239$	241$	290$	322$	335$	337$	339$	399$	424$	480$
		483$	611	654$	656$									
compare←memory	151b	145	544#											
dlicr	=dc0d	178#	257	384	436	490	682							
dlsdr	=dc0c	179#	264	276	370$	389	440	494	506	515	521	526	531	536
d2pra	=dd00	175#	361	662	664$	669	671$	675	677$					
dev	1301	83#												
flag	130d	94#	253$	280	281$									
inquire←format	146a	137	411#											
intlv	130c	93#	537$											
j←compare←memory	134c	144#												
j←inquire←format	1340	136#												
j←open←channel	1352	148#												
j←physical←read	1343	138#												
j←physical←write	1346	140#												
j←query←format	1349	142#												
j←sendcmd	134f	146#												
lf	1302	84#	607											
maxsec	130b	92#	532$											
minsec	130a	91#	527$											
numsec	1305	87#	240	338	516$									
oldclk	131b	116#	351$	362	366	367$								
open	=ffc0	165#	647											
open←channel	156e	149	622#											
oburstrd	=0000	191#	219											
oburstwr	=0002	192#	318											
pread	1355	139	202#											
pwrite	13d6	141	302#											
query←format	1493	143	448#											
read	1361	226#												
secsize	1308	89#	261	356										
sector	1304	86#	238	336										
sendcmd	1555	147	245	343	432	486	598#							
setlfs	=ffba	162#	642											
setnam	=ffbd	163#	645											
setup	157a	234	333	422	472	657#								
side	1309	90#	220	319	426	476								
spin←out	=ff47	166#	359	382										
status	1300	82#	265$	390$	441$	495$	501	507$	571$	584$				
temp	131c	117#	555$	565										
track	1303	85#	236	334	482	502$								
wait	15a2	263	273	372	388	439	492	505	514	520	525	530	579	664#
write	13e2	323#												

CHAPTER 12

1581 Burst Subroutines

Hardware: C128 with a 1581 disk drive

1) A set of subroutines for use with the 1581 Burst mode commands

Known bugs: None

This code includes a set of subroutines with jump table that make it easier to use the Burst mode commands of the 1581. Six commands are supported: logical read, logical write, physical read, physical write, query disk and inquire format.

In addition, there are four support routines: memory compare, memory read, memory write and dump cache. See the source listing for details on how to call these routines.

For an example of a BASIC program that calls these routines, see the file named "BURST EXAMPL.BAS" on the release disks. This is a 1581 2-drive backup program.

```
error addr  code        seq  source statement

                          1
                          2
                          3
                          4
                          5
                          6
                          7  ;*                        -----------------                          *
                          8  ;*  ---------------------  BURST  SUBROUTINES  -- (rev 2) ----------  *
                          9  ;*                        -----------------                          *
                         10  ;*                                                                   *
                         11  ;*   These assembly language sub-routines are designed for use in your *
                         12  ;* BASIC and machine code programs.  A BASIC program needs simply to   *
                         13  ;* POKE the appropriate values into the variable locations shown below, *
                         14  ;* and then SYS to the desired routine.  All of the BURST protocol and *
                         15  ;* handshaking is done for you.  The BASIC program can then PEEK any   *
                         16  ;* values returned.                                                   *
                         17  ;*
                         18  ;*  The routines in this listing that require data buffer storage areas *
                         19  ;* are passed the location of the buffer in BUFLOC.  BUFLOC points to  *
                         20  ;* RAM location in RAM bank 0 of the start of the buffer.  Since the   *
                         21  ;* KERNEL and I/O are needed, you must put BUFLOC below $C000.  But be *
                         22  ;* sure to put it in RAM above your BASIC text.  As a general rule,    *
                         23  ;* work your way back from $C000 and you'll be OK.                    *
                         24  ;*
                         25  ;* To use these routines, your BASIC program must first BLOAD the file *
                         26  ;* 'BURST SUBS.BIN'.  The routines load at $1300, so they are in a     *
                         27  ;* safe place below BASIC text area.                                  *
                         28  ;*
                         29  ;* There is no BURST FORMAT routine provided.  The following BASIC     *
                         30  ;* commands will format physical tracks 10 through 20 of the disk with *
                         31  ;* 5 1024 byte sectors:                                               *
                         32  ;*
                         33  ;*     OPEN 1,8,15                                                   *
                         34  ;*     PRINT#1,"U0";CHR$(3);CHR$(20);CHR$(5);CHR$(10);               *
                         35  ;*
                         36  ;*    Note the use of the semicolon (;) at the end of the statement.  *
                         37  ;* This is very important!  If there was no semicolon, the C128 would *
                         38  ;* send a carriage return after the last parameter.  Since the 1581    *
                         39  ;* counts the number of bytes sent to determine the number of optional *
                         40  ;* parameters that are being sent, it would misinterpret the carriage  *
                         41  ;* return as the next optional parameter.  In this case, it would be   *
                         42  ;* fill byte.  Any formatting errors can be checked via the command    *
                         43  ;* channel.                                                           *
                         44  ;*
                         45  ;*   Since the BURST commands make use of the command channel to the   *
                         46  ;* drive, the command channel must first be OPENed in your BASIC      *
                         47  ;* program.  The logical file number which you assigned to the command *
                         48  ;* channel should be poked to LF before calling any of these routines. *
                         49  ;*                                                                    *
                         50
                         51
                         52
                         53
                         54
                         55
                         56
                         57
```

```
error addr  code          seq   source statement

                          58    ;*************************************************************************
                          59    ;  Variables - Values from BASIC can be POKEd, PEEKed to these areas.
                          60    ;*************************************************************************
                          61
          =1300           62            *=$1300
                          63
      1300 00             64    STATUS  .byte 0             ; status byte
      1301 08             65    DEV     .byte 8             ; device number
      1302 08             66    LF      .byte 8             ; logical file number
      1303 =1304          67    TRACK   *=*+1               ; track
      1304 =1305          68    SECTOR  *=*+1               ; sector
      1305 =1306          69    NUMSEC  *=*+1               ; Number of sectors.
      1306 =1308          70    BUFLOC  *=*+2               ; Page # of buffer to get/put data.
      1308 =1309          71    SECSIZE *=*+1               ; Sector size (1=256, 2=512, 4=1024)
      1309 =130a          72    SIDE    *=*+1               ; Physical side of the disk (0 or 1).
      130a =130b          73    MINSEC  *=*+1               ; Minimum logical sector found in QUERY.
      130b =130c          74    MAXSEC  *=*+1               ; Maximum logical sector found in QUERY.
      130c =130d          75    INTLV   *=*+1               ; Physical interleave found in QUERY.
      130d =130e          76    FLAG    *=*+1               ; Empty track flag.
                          77                                ;  This flag is used to indicate that the
                          78    ;  track or data just read contains all 0's.  This is handy in some cases,
                          79    ; such as during a disk copy program.  When a disk is formatted, the sectors
                          80    ; are filled with 0's.  If a sector to be copied contains all 0's, then we
                          81    ; don't bother to write it to the destination disk
                          82
                          83
                          84    ;*******************************************************
                          85    ;  Other variables used in the following routines....
                          86    ;*******************************************************
                          87
                          88
      130e                89    cmdline
      130e 55 30          90            .byte 'u0'          ; Burst prefix.
          =131a           91            *=*+10              ; Parameter space for burst command.
      131a =131b          92    cmdlen  *=*+1               ; Length of the command string (# of bytes).
      131b =131c          93    oldclk  *=*+1               ; Status of clock line.
      131c =131d          94    temp    *=*+1
                          95
                          96    ;*************************************************************************
                          97    ;  JUMP TABLE of available burst routines.  SYS to these locations
                          98    ;  from BASIC.  The BURST routines themselves can then be modified
                          99    ;  or customized without affecting the SYS locations from BASIC.
                         100    ;*************************************************************************
                         101
          =1340          102           *=$1340
                         103
      1340                104   J←INQUIRE←FORMAT
      1340 4c 148d        105           jmp INQUIRE←FORMAT
      1343                106   J←PHYSICAL←READ
      1343 4c 136b        107           jmp PREAD
      1346                108   J←LOGICAL←READ
      1346 4c 135e        109           jmp LREAD
      1349                110   J←PHYSICAL←WRITE
      1349 4c 13f9        111           jmp PWRITE
      134c                112   J←LOGICAL←WRITE
      134c 4c 13ec        113           jmp LWRITE
                         114
```

```
error addr  code          seq   source statement

                          115
      134f                116   J←MEMORY←READ
      134f 4c 1532        117           jmp MEMORY←READ
      1352                118   J←MEMORY←WRITE
      1352 4c 1594        119           jmp MEMORY←WRITE
      1355                120   J←DUMP←CACHE
      1355 4c 15fb        121           jmp DUMP←CACHE
      1358                122   J←QUERY←FORMAT
      1358 4c 14af        123           jmp QUERY←FORMAT
      135b                124   J←COMPARE←MEMORY
      135b 4c 162a        125           jmp COMPARE←MEMORY
                          126
                          127   ;****************************************
                          128   ;  Locations of important C128 stuff...
                          129   ;****************************************
                          130
           =ffc9          131   chkout=$ffc9                ; kernel channel output
           =ffcc          132   clrchn=$ffcc                ; kernel clear channel
           =ffba          133   setlfs=$ffba                ; kernel set logical file number
           =ffbd          134   setnam=$ffbd                ; kernel set filename
           =ffd2          135   bsout =$ffd2                ; kernel basic input/output
           =ff47          136   spin←out=$ff47              ; Set up fast serial for input or output.
                          137                               ;  SEC for output, CLC for input.
           =dd00          138   d2pra =$dd00                ; C128 serial port location
           =0010          139   clkout=$10                  ; slow serial clock output bit mask
           =0040          140   clkin =$40                  ; slow serial clock input bit mask
           =dc0d          141   d1icr =$dc0d                ; 6526 CIA interrupt control register
           =dc0c          142   d1sdr =$dc0c                ; 6526 CIA serial data register
           =00fa          143   buffer=$fa                  ; zero page pointer variable
           =00fc          144   buffer2=$fc                 ; zero page pointer variable
                          145
                          146   ;******************************
                          147   ;  BURST command primitives
                          148   ;******************************
                          149
           =0000          150   PBURSTRD =$00               ; Physical burst read.
           =0002          151   PBURSTWR =$02               ; Physical burst write.
           =0080          152   LBURSTRD =$80               ; Logical burst read
           =0082          153   LBURSTWR =$82               ; Logical burst write.
           =0004          154   BURST←INQUIRE  =$04         ; Burst inquire.
           =009d          155   DUMPCACHE =$9D              ; Dump track cache ('force' bit set)
           =008a          156   BURST←QUERY =$8A            ; Query disk format.
                          157
                          158   ;************************************************************
                          159   ;* ---------------  BURST ROUTINES  ------------------- *
                          160   ;************************************************************
                          161
      135e                162   LREAD         ;Logical sector read from the device indicated by LF.
                          163                 ;The track and sector are in TRACK, SECTOR. The location
                          164                 ;Status byte from drive is returned in STATUS.
                          165
      135e a9 80          166           lda  #lburstrd        ;logical burst read command
      1360 8d 1310        167           sta  cmdline+2
      1363 a9 01          168           lda #$01
      1365 8d 1308        169           sta SECSIZE           ;Logical sector size is always 256 bytes.
      1368 4c 1377        170           jmp READ
                          171
```

```
error addr  code          seq   source statement

                          172
                          173
      136b                174   PREAD       ;Physical sector read from device indicated by LF.
                          175               ;The track and sector are in TRACK, SECTOR. The location
                          176               ;of start of the C128 buffer to put the read data in BUFLOC.
                          177               ;The physical sector size in SECSIZE (1=256,2=512,4=1024).
                          178               ;Number of sectors in NUMSEC.
                          179               ;Physical side of the disk in SIDE (0 or 1).
                          180               ;Status byte from drive is returned in STATUS.
                          181
      136b a9 00          182         lda #PBURSTRD       ;Physical burst read command.
      136d ae 1309        183         ldx SIDE            ; Check which side to read from.
      1370 f0 02          184         beq 1$
      1372 09 10          185         ora #$10            ; If side 1, then set bit in the command byte.
      1374 8d 1310        186   1$    sta  cmdline+2
                          187
                          188
      1377                189   READ
                          190
      1377 ad ff00        191         lda $ff00           ;Save old MMU setup.
      137a 48             192         pha
                          193
      137b a9 0e          194         lda #$0e            ;Set MMU for RAM0,KERNEL,I/O.
      137d 8d ff00        195         sta $ff00
                          196
      1380 20 167d        197         jsr SETU0           ;Put "U0" at start of command string.
                          198
      1383 ad 1303        199         lda TRACK
      1386 8d 1311        200         sta CMDLINE+3       ; track
      1389 ad 1304        201         lda SECTOR
      138c 8d 1312        202         sta CMDLINE+4       ; sector
      138f ad 1305        203         lda NUMSEC
      1392 8d 1313        204         sta CMDLINE+5       ; Number of sectors to read.
                          205
      1395 a9 06          206         lda #$06            ; Length of command string.
      1397 8d 131a        207         sta CMDLEN
      139a 20 1664        208         jsr  sendcmd        ; send cmd string
                          209
      139d ad 1306        210         lda  BUFLOC         ; Set up zero page indirect pointer.
      13a0 85 fa          211         sta  BUFFER
      13a2 ad 1307        212         lda  BUFLOC+1
      13a5 85 fb          213         sta  BUFFER+1
                          214
      13a7 a0 00          215         ldy  #0             ; clear the 'empty sector(s)' flag.
      13a9 8c 130d        216         sty  flag
                          217
      13ac 78             218         sei                 ; No irq's allowed during handshake.
      13ad 2c dc0d        219         bit  d1icr          ; clear pending
                          220
      13b0 20 169c        221         jsr CLK-CHNG        ;Change state or clock.
                          222
      13b3 ae 1308        223   1$    ldx  SECSIZE        ; Sector size gives # of pages per sector.
                          224
      13b6 20 16a5        225         jsr WAIT            ;Wait for fast byte (1st is status).
      13b9 ad dc0c        226         lda  d1sdr          ;Get status byte.
      13bc 8d 1300        227         sta STATUS
                          228
```

```
error addr  code        seq   source statement

                        229
                        230
      13bf 29 0f        231           and  #15             ;Was there an error?
      13c1 c9 02        232           cmp  #2              ;
      13c3 b0 21        233           bcs  5$              ; branch if error occured.
                        234
      13c5 20 169c      235           jsr CLK+CHNG         ;Change clock so next byte is sent.
                        236
      13c8 20 16a5      237   3$      jsr WAIT             ;Wait for the next byte.
                        238
      13cb 20 169c      239           jsr CLK+CHNG         ;Change state of clock so next byte is sent.
      13ce ad dc0c      240           lda  D1SDR           ;Get the data byte
      13d1 91 fa        241           sta  (buffer),y      ; and save it
                        242                                ;  while next byte is being transmitted.
                        243
      13d3 0d 130d      244           ora  flag            ;Update 'zero' sector flag.
      13d6 8d 130d      245           sta  flag
                        246
      13d9 c8           247           iny                  ;Any more in this page?
      13da d0 ec        248           bne  3$
                        249
      13dc e6 fb        250           inc BUFFER+1
      13de ca           251           dex                  ;Loop for the # of pages per sector.
      13df d0 e7        252           bne 3$
                        253
      13e1 ce 1313      254           dec CMDLINE+5        ;Loop for the number of sectors.
      13e4 d0 cd        255           bne 1$
                        256
      13e6 58           257   5$      cli
                        258
      13e7 68           259           pla                  ;Restore MMU to old configuration.
      13e8 8d ff00      260           sta $ff00
                        261
      13eb 60           262           rts
                        263
                        264   ;*********************************************************************
                        265
      13ec              266   LWRITE        ;Logical sector write to the device indicated by LF.
                        267                 ;The track and sector are in TRACK, SECTOR. The location
                        268                 ;of start of C128 buffer containing write data in BUFLOC.
                        269                 ;Status byte from drive is returned in STATUS.
                        270
      13ec a9 82        271           lda  #1burstwr
      13ee 8d 1310      272           sta  cmdline+2       ; burst write
      13f1 a9 01        273           lda #$01
      13f3 8d 1308      274           sta SECSIZE          ; Logical sector size is always 256 bytes.
      13f6 4c 1405      275           jmp WRITE
                        276
                        277
      13f9              278   PWRITE        ;Physical sector write to the device indicated by LF.
                        279                 ;The track and sector are in TRACK, SECTOR. The location
                        280                 ;of start of C128 buffer containing write data in BUFLOC.
                        281                 ;The physical sector size in SECSIZE (1=256,2=512,4=1024).
                        282                 ;Number of sectors in NUMSEC.
                        283                 ;Physical side in SIDE.
                        284                 ;Status byte from drive is returned in STATUS.
                        285
```

```
error addr  code          seq   source statement

                          286
                          287
      13f9 a9 02          288           lda #PBURSTWR         ;Physical burst write command.
      13fb ae 1309        289           ldx SIDE              ; Check which side to write to.
      13fe f0 02          290           beq 1$
      1400 09 10          291           ora #$10              ; If side 1, then set bit in the command byte.
      1402 8d 1310        292    1$     sta  cmdline+2
                          293
                          294
      1405                295    WRITE
                          296
      1405 ad ff00        297           lda $ff00             ;Save old MMU setup.
      1408 48             298           pha
                          299
      1409 a9 0e          300           lda #$0e              ;Set MMU for RAM0,KERNEL,I/O.
      140b 8d ff00        301           sta $ff00
                          302
      140e 20 167d        303           jsr SETU0             ;Put "U0" at start of command string.
      1411 ad 1303        304           lda TRACK
      1414 8d 1311        305           sta CMDLINE+3         ; track
      1417 ad 1304        306           lda SECTOR
      141a 8d 1312        307           sta CMDLINE+4         ; sector
      141d ad 1305        308           lda NUMSEC
      1420 8d 1313        309           sta CMDLINE+5         ; Number of sectors to write.
                          310
      1423 a9 06          311           lda #$06
      1425 8d 131a        312           sta CMDLEN            ;Command length.
      1428 20 1664        313           jsr  sendcmd          ; send cmd string
                          314
      142b ad 1306        315           lda  BUFLOC           ; Set up zero page indirect pointer.
      142e 85 fa          316           sta  BUFFER
      1430 ad 1307        317           lda BUFLOC+1
      1433 85 fb          318           sta BUFFER+1
                          319
      1435 a9 40          320           lda  #clkin           ;Initial clock status.
      1437 8d 131b        321           sta  oldclk
                          322
      143a a0 00          323           ldy  #0
      143c 78             324           sei                   ; no irq's during burst handshake
                          325
      143d ae 1308        326    1$     ldx  SECSIZE          ; Sector size gives # of pages per sector.
                          327
      1440 38             328           sec                   ;Turn fast serial to output mode.
      1441 20 ff47        329           jsr  spin=out
                          330
      1444 ad dd00        331    2$     lda  d2pra            ;Wait for state change.
      1447 4d 131b        332           eor  oldclk
      144a 29 40          333           and  #clkin
      144c f0 f6          334           beq  2$
                          335
      144e 4d 131b        336           eor  oldclk           ;Change status of OLDCLK.
      1451 8d 131b        337           sta  oldclk
                          338
      1454 b1 fa          339           lda  (buffer),y       ; get data
      1456 8d dc0c        340           sta  disdr            ;  & send it
                          341
      1459 20 16a5        342           jsr wAIT              ;Wait for the byte to be transmitted.
```

```
error addr  code        seq.  source statement

                        343
                        344
      145c c8           345          iny
      145d d0 e5        346          bne  2$              ;Any more left in this page?
                        347
      145f e6 fb        348          inc  buffer+1
      1461 ca           349          dex                  ;Loop for the # of pages per sector.
      1462 d0 e0        350          bne  2$
                        351
      1464 18           352          clc                  ;Turn around to input mode to get STATUS.
      1465 20 ff47      353          jsr  spin←out
                        354
      1468 2c dc0d      355          bit  d1icr           ; clear pending
                        356
      146b 20 1688      357          jsr  clklo           ; set clock low when ready for status
                        358
      146e 20 16a5      359          jsr WAIT             ;Wait for the byte to be shifted in.
      1471 ad dc0c      360          lda  disdr           ;Get the status byte.
      1474 8d 1300      361          sta STATUS           ;Save it.
      1477 48           362          pha
      1478 20 1693      363          jsr  clkhi           ;Release the slow clock line.
      147b 68           364          pla
                        365
      147c 29 0f        366          and  #15             ;Check for any error.
      147e c9 02        367          cmp  #2
      1480 b0 05        368          bcs  7$              ;  branch if there was an error.
                        369
      1482 ce 1313      370          dec  CMDLINE+5       ;Loop for the number of sectors.
      1485 d0 b6        371          bne  1$
                        372
      1487 58           373    7$    cli
      1488 68           374          pla                  ;Restore old memory configuration.
      1489 8d ff00      375          sta $ff00
      148c 60           376          rts
                        377
                        378    ;*************************************************************
                        379
      148d              380    INQUIRE←FORMAT  ;Sends an INQUIRE DISk command to the drive indicate by
                        381                    ;the logical file (LF).  Status is returned in STATUS.
                        382
      148d 20 167d      383          jsr SETU0            ;Put "U0" at start of command string.
      1490 a9 04        384          lda  #BURST←INQUIRE  ; inquire burst command
      1492 8d 1310      385          sta  cmdline+2
      1495 a9 03        386          lda #$03             ; length of command.
      1497 8d 131a      387          sta CMDLEN
      149a 20 1664      388          jsr  sendcmd         ; send cmd string
                        389
      149d 78           390          sei                  ;Disable interrupts during handshake.
                        391
      149e 2c dc0d      392          bit D1ICR            ;Clear any byte ready that's pending.
                        393
      14a1 20 169c      394          jsr CLK←CHNG         ;Change clock so 1581 sends next.
      14a4 20 16a5      395          jsr WAIT             ;wait for the byte to be shifted in.
      14a7 ad dc0c      396          lda D1SDR            ;Get the status byte.
      14aa 8d 1300      397          sta STATUS           ;Save it off.
      14ad 58           398          cli
      14ae 60           399          rts
```

```
error addr  code          seq   source statement

                          400
                          401
                          402   ;************************************************************
                          403
     14af                 404   QUERY←FORMAT  ;Sends a QUERY DISK FORMAT command to the drive indicate by
                          405                 ;the logical file (LF).  Physical track number to query
                          406                 ;should be provided in TRACK.  Physical side should
                          407                 ;be provided in SIDE.  Status is returned in STATUS.
                          408                 ;Number of sectors found on the track returned in NUMSEC.
                          409                 ;Logical track number found in the sector headers returned
                          410                 ;in TRACK.  Minimum logical sector number found in the
                          411                 ;sector headers is returned in MINSEC.  The maximium
                          412                 ;logical sector is returned MAXSEC.  Physical interleave
                          413                 ;is returned in INTLV.
                          414                 ;If an error is encountered in compiling this information
                          415                 ;(as indicated by STATUS), then none of the return values
                          416                 ;are valid (except STATUS).
                          417
     14af 20 167d         418           jsr SETU0             ;Put "U0" at start of command string.
                          419
     14b2 a9 8a           420           lda  #BURST←QUERY     ;QUERY DISK burst command
     14b4 ae 1309         421           ldx SIDE              ; Set the side bit accordingly.
     14b7 d0 02           422           bne 4$
     14b9 09 10           423           ora #$10
     14bb 8d 1310         424   4$      sta cmdline+2
                          425
     14be ad 1303         426           lda TRACK             ; Physical track offset.
     14c1 8d 1311         427           sta cmdline+3
     14c4 a9 04           428           lda #$04              ; length of command.
     14c6 8d 131a         429           sta CMDLEN
     14c9 20 1664         430           jsr  sendcmd          ; send cmd string
                          431
     14cc 78              432           sei                   ;Disable interrupts during handshake.
     14cd 2c dc0d         433           bit D1ICR             ;Clear any byte ready that's pending.
                          434
     14d0 20 169c         435           jsr CLK←CHNG          ;Change state of clock so 1581 sends next.
     14d3 20 16a5         436           jsr WAIT              ;Wait for the first status byte.
     14d6 ad dc0c         437           lda D1SDR             ;Get the status byte (status of track 0).
     14d9 8d 1300         438           sta STATUS            ;Save it off.
                          439
     14dc 29 0f           440           and #$0f              ;Was there an error?
     14de c9 02           441           cmp #2
     14e0 b0 4e           442           bcs 5$                ; branch if there was an error.
                          443
     14e2 20 169c         444           jsr CLK←CHNG          ;Change state of clock, so 1581 sends next.
     14e5 20 16a5         445           jsr WAIT              ;Wait for  ext status byte to be ready.
     14e8 ad dc0c         446           lda D1SDR             ;Get it (status of track TRACK).
     14eb 8d 1300         447           sta STATUS            ;Save it.
                          448
     14ee 29 0f           449           and #$0f              ;Was there an error?
     14f0 c9 02           450           cmp #2
     14f2 b0 3c           451           bcs 5$                ; branch if an error.
                          452
     14f4 20 169c         453           jsr CLK←CHNG          ;Change state of clock, so 1581 sends next.
     14f7 20 16a5         454           jsr WAIT              ;Wait for 'number of sectors byte' to be ready.
     14fa ad dc0c         455           lda D1SDR             ;Get it.
     14fd 8d 1305         456           sta NUMSEC            ;Save it.
```

```
error addr  code        seq   source statement

                        457
                        458
     1500 20 169c       459           jsr CLK←CHNG          ;Change state of clock, so 1581 sends next.
     1503 20 16a5       460           jsr WAIT              ;Wait for 'logical track #' byte to be ready.
     1506 ad dc0c       461           lda D1SDR             ;Get it.
     1509 8d 1303       462           sta TRACK             ;Save it.
                        463
     150c 20 169c       464           jsr CLK←CHNG          ;Change state of clock, so 1581 sends next.
     150f 20 16a5       465           jsr WAIT              ;Wait for 'minimum sector #' byte to be ready.
     1512 ad dc0c       466           lda D1SDR             ;Get it.
     1515 8d 130a       467           sta MINSEC            ;Save it.
                        468
     1518 20 169c       469           jsr CLK←CHNG          ;Change state of clock, so 1581 sends next.
     151b 20 16a5       470           jsr WAIT              ;Wait for 'maximum sector #' byte to be ready.
     151e ad dc0c       471           lda D1SDR             ;Get it.
     1521 8d 130b       472           sta MAXSEC            ;Save it.
                        473
     1524 20 169c       474           jsr CLK←CHNG          ;Change state of clock, so 1581 sends next.
     1527 20 16a5       475           jsr WAIT              ;Wait for 'interleave' byte to be ready.
     152a ad dc0c       476           lda D1SDR             ;Get it.
     152d 8d 130c       477           sta INTLV             ;Save it.
                        478
     1530 58            479   5$      cli
     1531 60            480           rts
                        481
                        482   ;*****************************************************************
                        483
     1532               484   MEMORY←READ   ;Burst memory read of the 1581.  Page in 1581 memory to
                        485                 ;start reading at in .X, number of pages to read in .Y.
                        486                 ;location to store data in C128 memory in BUFLOC.
                        487                 ;Logical file to be read from in LF.
                        488
     1532 ad ff00       489           lda $ff00             ;Save old MMU setup.
     1535 48            490           pha
                        491
     1536 a9 0e         492           lda #$0e              ;Set MMU for RAM0,KERNEL,I/O.
     1538 8d ff00       493           sta $ff00
                        494
     153b 20 167d       495           jsr SETU0             ;Put "U0" at start of command string.
     153e a9 3e         496           lda #$3E              ;('>') 'burst memory read' command string.
     1540 8d 1310       497           sta CMDLINE+2         ;  ( "U0>MR" )
     1543 a9 4d         498           lda #$4D              ;('M')
     1545 8d 1311       499           sta CMDLINE+3
     1548 a9 52         500           lda #$52              ;('R')
     154a 8d 1312       501           sta CMDLINE+4
     154d 8e 1313       502           stx CMDLINE+5         ; 1581 page to start reading from.
     1550 8c 1314       503           sty CMDLINE+6         ; # of pages to read.
                        504
     1553 a9 07         505           lda #$07              ; Length of command string.
     1555 8d 131a       506           sta CMDLEN
     1558 20 1664       507           jsr  sendcmd          ; send cmd string
                        508
     155b ad 1306       509           lda  BUFLOC           ; Set up zero page indirect pointer.
     155e 85 fa         510           sta  BUFFER
     1560 ad 1307       511           lda  BUFLOC+1
     1563 85 fb         512           sta  BUFFER+1
                        513
```

```
error addr  code        seq   source statement

                        514
      1565 a9 00        515           lda  #0               ; clear the 'empty sector(s)' flag.
      1567 8d 130d      516           sta  flag
                        517
      156a 78           518           sei                   ; No irq's allowed during handshake.
                        519
      156b 2c dc0d      520           bit  d1icr            ; clear pending
                        521
      156e 20 169c      522           jsr CLK+CHNG          ;Change state of clock.
                        523
      1571 a0 00        524           ldy #0
      1573 20 16a5      525    3$     jsr WAIT              ;Wait for the byte to be shifted in.
                        526
      1576 20 169c      527           jsr CLK+CHNG          ;Change clock so next byte is sent.
      1579 ad dc0c      528           lda  d1sdr            ; get data
      157c 91 fa        529           sta  (buffer),y       ;  and save it while next byte is transmitted.
                        530
      157e 0d 130d      531           ora  flag             ; Update 'zero' flag.
      1581 8d 130d      532           sta  flag
                        533
      1584 c8           534           iny
      1585 d0 ec        535           bne  3$               ;Any more in this page?
                        536
      1587 e6 fb        537           inc BUFFER+1
      1589 ce 1314      538           dec CMDLINE+6         ;Any more pages to do?
      158c d0 e5        539           bne 3$
                        540
      158e 58           541    5$     cli
                        542
      158f 68           543           pla                   ;Restore old memory configuration.
      1590 8d ff00      544           sta $ff00
                        545
      1593 60           546           rts
                        547
                        548    ;***************************************************************
                        549
      1594              550    MEMORY-WRITE  ;Burst memory write to the 1581's memory.  The
                        551                  ;location in C128 memory to send data from in BUFLOC.
                        552                  ;The page in 1581 memory to start writing to in .X.
                        553                  ;The number of pages to write in .Y.
                        554                  ;Logical file to be written to in LF.
                        555
      1594 ad ff00      556           lda $ff00             ;Save old MMU setup.
      1597 48           557           pha
                        558
      1598 a9 0e        559           lda #$0e              ;Set MMU for RAM0,KERNEL,I/O.
      159a 8d ff00      560           sta $ff00
                        561
      159d 20 167d      562           jsr SET00             ;Put "00" at start of command string.
      15a0 a9 3e        563           lda #$3E              ;('>') 'burst memory write' command string.
      15a2 8d 1310      564           sta CMDLINE+2         ;  to "00/MW" .
      15a5 a9 4d        565           lda #$4D              ;('M')
      15a7 8d 1311      566           sta CMDLINE+3
      15aa a9 57        567           lda #$57              ;('W')
      15ac 8d 1312      568           sta CMDLINE+4
      15af 8e 1313      569           stx CMDLINE+5         ; 1581 page to start writing to.
      15b2 8c 1314      570           sty CMDLINE+6         ; # of pages to write.
```

```
error addr  code          seq   source statement

                          571
                          572
      15b5 a9 07          573           lda #$07            ; Length of command string.
      15b7 8d 131a        574           sta CMDLEN
                          575
      15ba 20 1664        576           jsr  sendcmd        ; send cmd string
                          577
      15bd ad 1306        578           lda  BUFLOC         ; Set up zero page indirect pointer.
      15c0 85 fa          579           sta  BUFFER
      15c2 ad 1307        580           lda  BUFLOC+1
      15c5 85 fb          581           sta  BUFFER+1
                          582
      15c7 a9 40          583           lda  #clkin         ;Initial clock status.
      15c9 8d 131b        584           sta  oldclk
                          585
      15cc a0 00          586           ldy #0
      15ce 78             587           sei                 ;No IRQ's allowed during handshake.
                          588
      15cf 38             589           sec                 ; Set to output mode.
      15d0 20 ff47        590           jsr SPIN=OUT
                          591
      15d3 ad dd00        592    2$     lda d2pra           ;Wait for state (slow clock line) change.
      15d6 4d 131b        593           eor OLDCLK
      15d9 29 40          594           and #CLKIN
      15db f0 f6          595           beq 2$
                          596
      15dd 4d 131b        597           eor OLDCLK          ;Change status of OLDCLK variable.
      15e0 8d 131b        598           sta OLDCLK
                          599
      15e3 b1 fa          600           lda (BUFFER),y      ;Get data to write.
      15e5 8d dc0c        601           sta D1SDR           ;Send it.
                          602
      15e8 20 16a5        603           jsr WAIT            ;Wait for the byte to be sent.
                          604
      15eb c8             605           iny
      15ec d0 e5          606           bne 2$              ;More in this page?
                          607
      15ee e6 fb          608           inc BUFFER+1
      15f0 ce 1314        609           dec CMDLINE+6
      15f3 d0 de          610           bne 2$              ;Any more pages to send?
                          611
      15f5 58             612           cli
                          613
      15f6 68             614           pla                 ;Restore old memory configuration.
      15f7 8d ff00        615           sta $ff00
                          616
      15fa 60             617           rts
                          618
                          619
                          620    ;*************************************************************
                          621
                          622
      15fb                623    DUMP=CACHE    ;Dumps the track cache at 1581 $6000 to the physical track
                          624                  ;specified in TRACK, on the side specified in SIDE.  The
                          625                  ;'force' bit is set, so it is written whether dirty or not.
                          626
                          627
```

```
error addr  code          seq   source statement

                          628
      15fb 20 167d        629           jsr SETU0
      15fe a9 9d          630           lda #DUMPCACHE        ;Dump track cache command.
                          631
      1600 ae 1309        632           ldx SIDE              ;  Set SIDE bit accordingly.
      1603 f0 02          633           beq 1$
      1605 09 40          634           ora #$40
                          635
      1607 8d 1310        636   1$      sta CMDLINE+2         ;  Put the command byte into command string.
      160a ad 1303        637           lda TRACK
      160d 8d 1311        638           sta CMDLINE+3         ;  Physical track to dump cache to.
                          639
      1610 a9 04          640           lda #4
      1612 8d 131a        641           sta CMDLEN            ;Length of the command string.
      1615 20 1664        642           jsr SENDCMD
                          643
      1618 78             644           sei                   ;Disable interrupts during handshake.
                          645
      1619 2c dc0d        646           bit D1ICR             ;Clear any byte ready that's pending.
                          647
      161c 20 169c        648           jsr CLK←CHNG          ;Change clock so 1581 will send status byte.
      161f 20 16a5        649           jsr WAIT              ;Wait for byte to be shifted in.
      1622 ad dc0c        650           lda D1SDR             ;Get the status byte.
      1625 8d 1300        651           sta STATUS            ;Save it off.
                          652
      1628 58             653           cli
      1629 60             654           rts
                          655
                          656
                          657   ;*****************************************************************
                          658
      162a                659   COMPARE←MEMORY  ;Compares memory blocks in the C128 memory.
                          660                   ;Number of pages to compare in .A.
                          661                   ;First page of first memory block in .X.
                          662                   ;First page of second memory block in .Y.
                          663                   ;If they are equal, then STATUS=0.
                          664
      162a 8d 131c        665           sta temp
                          666
      162d ad ff00        667           lda $ff00             ;Save old MMU setup.
      1630 48             668           pha
                          669
      1631 a9 0e          670           lda #$0e              ;Set MMU for RAM0,KERNEL,I/O.
      1633 8d ff00        671           sta $ff00
                          672
      1636 86 fb          673           stx buffer+1          ;Set up MSB of 1st memory pointer.
      1638 84 fd          674           sty buffer2+1         ;Set up MSB of 2nd memory pointer.
      163a ae 131c        675           ldx temp              ;Number of pages to compare.
                          676
      163d a9 00          677           lda #0                ;Set up LSB's of memory pointers.
      163f 85 fa          678           sta buffer
      1641 85 fc          679           sta buffer2
                          680
      1643 8d 1300        681           sta STATUS            ;Initialize STATUS.
                          682
      1646 a0 00          683           ldy #0
                          684
```

```
error addr  code          seq   source statement

                          685
      1648 b1 fa          686   2$      lda (buffer),y
      164a d1 fc          687           cmp (buffer2),y
      164c f0 07          688           beq 1$
                          689
      164e a9 ff          690           lda #$ff              ;Not equal!  Load STATUS with nonzero.
      1650 8d 1300        691           sta STATUS
      1653 d0 0a          692           bne 99$               ;  (branch always)
                          693
      1655 c8             694   1$      iny
      1656 d0 f0          695           bne 2$                ;More in this page?
                          696
      1658 e6 fb          697           inc buffer+1
      165a e6 fd          698           inc buffer2+1
      165c ca             699           dex                   ;# of pages counter.
      165d d0 e9          700           bne 2$
                          701
      165f 68             702   99$     pla                   ;Restore old memory configuration.
      1660 8d ff00        703           sta $ff00
                          704
      1663 60             705           rts
                          706
                          707   ;*************************************************************
                          708
      1664                709   SENDCMD       ;Sends the command in CMDLINE to the logical file
                          710                 ;indicated by LF.  Length of the command should be in CMDLEN.
                          711
      1664 ae 1302        712           ldx LF
      1667 20 ffc9        713           jsr  chkout           ; channel output (pointed to by .X)
      166a a2 00          714           ldx  #0
      166c ac 131a        715           ldy  cmdlen           ; send cmd
      166f bd 130e        716   1$      lda  cmdline,x
      1672 20 ffd2        717           jsr  bsout
      1675 e8             718           inx
      1676 88             719           dey
      1677 d0 f6          720           bne  1$
                          721
      1679 20 ffcc        722           jsr  clrchn           ; send buffered char & eoi
      167c 60             723           rts
                          724
                          725   ;*************************************************************
                          726
      167d a9 55          727   SETU0   lda #85               ;'U'
      167f 8d 130e        728           sta CMDLINE
      1682 a9 30          729           lda #48               ;'0'
      1684 8d 130f        730           sta CMDLINE+1
      1687 60             731           rts
                          732
                          733
      1688                734   CLKL0                 ; set clock low
      1688 48             735           pha
      1689 ad dd00        736           lda  d2pra
      168c 09 10          737           ora  #clkout
      168e 8d dd00        738           sta  d2pra
      1691 68             739           pla
      1692 60             740           rts
                          741
```

```
error addr  code          seq   source statement

                          742
                          743
      1693                744   CLKHI                 ; set clock high
      1693 ad dd00        745           lda  d2pra
      1696 29 ef          746           and  #$ff-clkout
      1698 8d dd00        747           sta  d2pra
      169b 60             748           rts
                          749
      169c                750   CLK<CHNG              ;  change the state of the clock line output.
      169c ad dd00        751           lda D2PRA
      169f 49 10          752           eor #clkout
      16a1 8d dd00        753           sta D2PRA
      16a4 60             754           rts
                          755
      16a5                756   WAIT                  ;  wait for the shift register to be full or empty.
      16a5 a9 08          757   1$      lda #8
      16a7 2c dc0d        758           bit D1ICR
      16aa f0 f9          759           beq 1$
      16ac 60             760           rts
                          761
                          762
                          763           .end
   0 errors detected
```

symbol table
(blank) = label, (=) = symbol, (+)= multibly defined

bsout	=ffd2	buffer	=00fa	buffer2	=00fc	bufloc	1306	burst←inquire	=000a
burst←query	=008a	chkout	=ffc9	clkhi	1693	clkin	=0040	clklo	1688
clkout	=0010	clk←chng	169c	clrchn	=ffcc	cmdlen	131a	cmdline	130e
compare←memory	162a	d1icr	=dc0d	d1sdr	=dc0c	d2pra	=dd00	dev	1301
dumpcache	=009d	dump←cache	15fb	flag	130d	inquire←format	148d	intlv	130c
j←compare←memory	135b	j←dump←cache	1355	j←inquire←format	1340	j←logical←read	1346	j←logical←write	134c
j←memory←read	134f	j←memory←write	1352	j←physical←read	1343	j←physical←write	1349	j←query←format	1358
lburstrd	=0080	lburstwr	=0082	lf	1302	lread	135e	lwrite	13ec
maxsec	130b	memory←read	1532	memory←write	1594	minsec	130a	numsec	1305
oldclk	131b	pburstrd	=0000	pburstwr	=0002	pread	136b	pwrite	13f9
query←format	14af	read	1377	secsize	1308	sector	1304	sendcmd	1664
setlfs	=ffba	setnam	=ffbd	setup	167d	side	1309	spin←out	=ff47
status	1300	temp	131c	track	1303	wait	16a5	write	1405

cross reference
(<#> = definition, <$> = write, <blank> = read)

cross reference
((#) = definition, ($) = write, (blank) = read)

read	1377	170	189#											
secsize	1308	71#	169$	223	274$	326								
sector	1304	68#	201	306										
sendcmd	1664	208	313	388	430	507	576	642	709#					
setlfs	=ffba	133#												
setnam	=ffbd	134#												
setup	167d	197	303	383	418	495	562	629	727#					
side	1309	72#	183	289	421	632								
spin-out	=ff47	136#	329	353	590									
status	1300	64#	227$	361$	397$	436$	447$	651$	681$	691$				
temp	131c	94#	665$	675										
track	1303	67#	199	304	426	462$	637							
wait	16a5	225	237	342	359	395	436	445	454	460	465	470	475	515
		603	649	756#										
write	1405	275	295#											

CHAPTER 13

This document details the many user-callable routines available in the C128 BASIC 7.0 math package. The C128 represents the first Commodore 65xx system to fully vectorize and describe the calling parameters necessary to fully and safely utilize the available math subroutines. The table below summarizes the documented routines and provides the location of the jump vector in the BASIC 7.0 ROMs.

FRED BOWEN APRIL 22, 1986

Format Conversions

```
AF00        AYINT           ;convert f.p. to integer
AF03        GIVAYF          ;convert integer to f.p.
AF06        FOUT            ;convert f.p. to ascii string
AF09        VAL_1           ;convert ascii string to f.p.
AF0C        GETADR          ;convert f.p. to an address
AF0F        FLOATC          ;convert address to f.p.
```

Math Functions

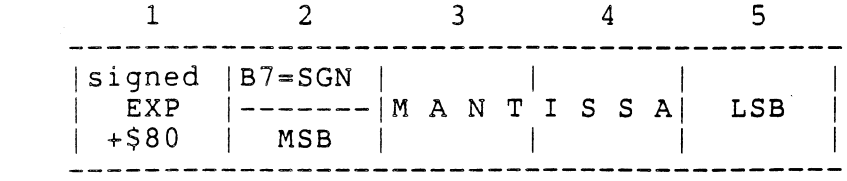
```
AF12        FSUB            ;MEM - FACC
AF15        FSUBT           ;ARG - FACC
AF18        FADD            ;MEM + FACC
AF1B        FADDT           ;ARG + FACC
AF1E        FMULT           ;MEM * FACC
AF21        FMULTT          ;ARG * FACC
AF24        FDIV            ;MEM / FACC
AF27        FDIVT           ;ARG / FACC
AF2A        LOG             ;natural logarithm of FACC
AF2D        INT             ;greatest integer of FACC
AF30        SQR             ;square root of FACC
AF33        NEGOP           ;negate (invert sign) of FACC
AF36        FPWR            ;ARG to the MEM power
AF39        FPWRT           ;ARG to the FACC power
AF3C        EXP             ;EXP of FACC
AF3F        COS             ;COS of FACC
AF42        SIN             ;SIN of FACC
AF45        TAN             ;TAN of FACC
AF48        ATN             ;ATN of FACC
AF4B        ROUND           ;round FACC
AF4E        ABS             ;absolute value of FACC
AF51        SIGN            ;sign of FACC
AF54        FCOMP           ;compare FACC with MEM
AF57        RND_0           ;generate random number in FACC
```

Movement

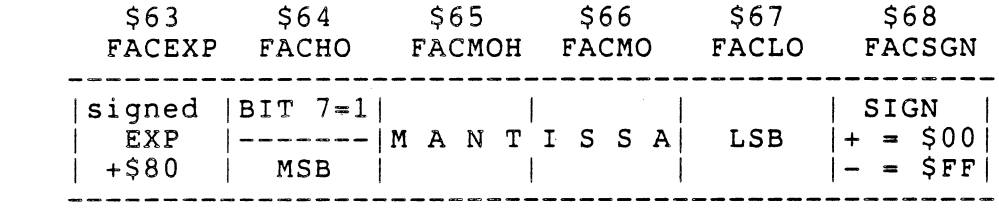
```
AF5A        CONUPK          ;move RAM MEM to ARG
AF5D        ROMUPK          ;move ROM MEM to ARG
AF60        MOVFRM          ;move RAM MEM to FACC
AF63        MOVFM           ;move ROM MEM to FACC
AF66        MOVMF           ;move FACC to MEM
AF69        MOVFA           ;move ARG to FACC
AF6C        MOVAF           ;move FACC to ARG
```

Floating Point Math Package Conventions

In BASIC memory the number is PACKED and looks like this:



Because the mantissa is normalized such that its msb is always 1, BASIC stores the SIGN of the mantissa here to save a byte of storage. It must be normalized when put in the FACC (see CONUPK). In the FACC the NORMALIZED number looks like this:



Negative exponents are not stored 2's complement. The maximum exponent is 10^38 ($FF) and the minimum is 10^-39 ($01). A zero value for the exponent means the number is zero. Since the exponent is a power of 2, it can be described as the number of left (EXP>$80) or right (EXP<=$80) shifts to be performed on the normalized mantissa to create the binary representation of the value. There is a second floating accumulator called ARG which has the same layout. It is located at $6A through $6F. Throughout the math package the floating point format is:

* the mantissa is 24 bits long.
* the binary point is to the left of the msb.
* the mantissa is always positive, and its msb is always 1.
* number = mantissa * 2^exponent, sign in FACSGN.
* the sign of the exponent is the msb of the exponent.
* the exponent is stored in excess $80 (i.e., it is a signed 8-bit number with $80 added to it.)
* an exponent of zero means the number is zero. (note that the rest of the accumulator cannot be assumed to be zero.)
* to keep the same number in the accumulator while shifting:
  right shifts --> increment exponent
  left shifts --> decrement exponent

Arithmetic routine calling conventions:

* For one argument functions:
  the argument is in the FACC.
  the result is left in the FACC.
* For two argument operations:
  the first argument is in MEMORY (packed) or ARG (unpacked).
  the second argument is in the FACC.
  the result is left in the FACC.
* Always call ROM routines with the SYSTEM memory configuration in context ($FF00=$00, ROMs, RAM-0, I/O) except as noted herein.

A note concerning precision. Since the mantissa is always normalized, the high order bit of the most significant byte is always one. This guarantees at least 40 bits (5 byte mantissa times 8 bits each) of precision, which is approximately 9 significant digits plus a few bits for rounding. In fact, there is a 'rounding' byte, FACOV ($71), which should, for the greatest degree of precision, be loaded whenever you load the FACC. The high order bit of FACOV is utilized in most of the math routines. While some of the 'movement' routines 'round' the loaded floating point number (i.e., FACOV = $00), others (such as CONUPK) do not- assuming the value of FACOV is the useful result of an operation in progress. In 99% of the cases you need not worry about it, as its significance is virtually nil. For the greatest degree of precision however, use it.

A few examples of normalized (FACC) floating point numbers:

VALUE		EXP	M A N T I S S A				SIGN
1E38	=	FF	96	76	99	53	00
4E10	=	A4	95	02	F9	00	00
2E10	=	A3	95	02	F9	00	00
1E10	=	A4	95	02	F9	00	00
10	=	84	A0	00	00	00	00
1	=	81	80	00	00	00	00
.5	=	80	80	00	00	00	00
.25	=	7F	80	00	00	00	00
.6	=	80	99	99	99	9A	00
1E-04	=	73	D1	B7	59	59	00
1E-37	=	06	88	1C	EA	15	00
1E-38	=	02	D9	C7	DC	EE	00
3E-39	=	01	82	AB	1E	2A	00
0	=	00	xx	xx	xx	xx	00
-1	=	81	80	00	00	00	FF
-5	=	83	A0	00	00	00	FF

Now for a simple example of deriving the actual binary from the FACC:

```
    5   =       83       A0       00       00       00       00
                 |         \
                 |          \
             ($83-$80)     ($A0)
                 |           |
which means:   2^3    *   .10100000,  or shift mantissa LEFT 3,

which gives:    101.00000 (binary) or 5.0 (hex)
```

```
NAME:           AYINT   ($AF00)
FUNCTION:       CONVERT FLOATING POINT TO INTEGER

PREPARATION:    FACC contains floating point number (-32768<=n<=32767)

RESULT:         FACMO ($66) contains signed integer (msb)
                FACLO ($67) contains signed integer (lsb)

ERROR:          ?ILLEGAL QUANTITY ERROR if FACC too big.

EXAMPLE:        JSR $AF00       ;INT(FACC)
                LDA $66         ;MSB
                LDY $67         ;LSB
```

==========

```
NAME:           GIVAYF  ($AF03)
FUNCTION:       CONVERT INTEGER TO FLOATING POINT

PREPARATION:    .A contains signed integer (msb)
                .Y contains signed integer (lsb)

RESULT:         FACC contains floating point number

EXAMPLE:        LDA #>INTEGER
                LDY #<INTEGER
                JSR $AF03       ;FLOAT (A,Y)
```

==========

```
NAME:           FOUT    ($AF06)
FUNCTION:       CONVERT FLOATING POINT TO ASCII STRING

PREPARATION:    FACC contains floating point number

RESULT:         FBUFFR ($100) contains ASCII string (null terminated)
                .A contains pointer to string (lsb)
                .Y contains pointer to string (msb)

EXAMPLE:        JSR $AF06       ;CONVERT FACC TO STRING AT $100
```

==========

NAME: VAL_1 ($AF09)
FUNCTION: CONVERT ASCII STRING TO FLOATING POINT

PREPARATION: INDEX1 ($24,$25) contains pointer to string
.A contains length of string

SPECIAL NOTES: String *must* be in RAM-1 or common RAM. Any invalid character terminates conversion when encountered (i.e., acts like a terminator).

```
**********************************************
*  *MUST* be called from ROM or common RAM-  *
*  this routine RTS's with RAM-1 in context!  *
**********************************************
```

RESULT: FACC contains floating point number

EXAMPLE:

```
LDA #<POINTER
LDY #>POINTER
STA INDEX1       ;SET POINTER TO STRING
STY INDEX1+1
LDA #LENGTH      ;SET STRING LENGTH
JSR $AF09        ;FACC = VAL(STRING)
```

NAME: GETADR ($AF0C)
FUNCTION: CONVERT FLOATING POINT TO ADDRESS

PREPARATION: FACC contains floating point number (0<=n<=65535)

RESULT: POKER ($16,$17) contains unsigned integer address

ERROR: ?ILLEGAL QUANTITY ERROR if FACC too big.

EXAMPLE:

```
JSR $AF0C        ;ADR(FACC)
LDA $16          ;LSB
LDY $17          ;MSB
```

NAME: FLOATC ($AF0F)
FUNCTION: CONVERT ADDRESS TO FLOATING POINT

PREPARATION: FACHO ($64) contains address (msb)
FACMOH ($65) contains address (lsb)
.X contains exponent ($90 always)
.C=1 if positive (always)

RESULT: FACC contains floating point number

ERROR: ?OVERFLOW ERROR if FACC too big.

EXAMPLE:

```
LDA #<ADDRESS
LDY #>ADDRESS
STA FACMOH       ;SET ADDRESS
STY FACHO
LDX #$90         ;EXPONENT
SEC              ;POSITIVE
JSR $AF0F        ;FLOAT ADDRESS
```

NAME: FSUB ($AF12)
FUNCTION: FACC = MEMORY - FACC

PREPARATION: FACC contains floating point subtrahend
.A = pointer (lsb) to packed floating point minuend
.Y = pointer (msb) to packed floating point minuend

SPECIAL NOTES: The minuend *MUST* be in RAM-1 or common RAM in packed format. FSUB calls CONUPK to normalize it.

RESULT: FACC contains floating point difference

ERROR: ?OVERFLOW ERROR if FACC too big.

EXAMPLE:

```
LDA #<POINTER
LDY #>POINTER     ;SET POINTER TO *PACKED* MINUEND
JSR $AF12         ;SUBTRACT MEMORY FROM FACC, DIFF IN FACC
```

=-=

NAME: FSUBT ($AF15)
FUNCTION: FACC = ARG - FACC

PREPARATION: FACC contains floating point subtrahend
ARG contains floating point minuend

SPECIAL NOTES: This routine is similar to FSUB. The only difference is the call to CONUPK- FSUBT assumes you have already loaded ARG with unpacked minuend.)

RESULT: FACC contains floating point difference

ERROR: ?OVERFLOW ERROR if FACC too big.

EXAMPLE:

```
JSR $AF12         ;SUBTRACT ARG FROM FACC, DIFF IN FACC
```

=-=

```
NAME:           FADD    ($AF18)
FUNCTION:       FACC = MEMORY + FACC

PREPARATION:    FACC contains floating point addend
                .A = pointer (lsb) to packed floating point addend
                .Y = pointer (msb) to packed floating point addend

SPECIAL NOTES:  The second addend *MUST* be in RAM-1 or common RAM in
                packed format. FADD calls CONUPK to normalize it.

RESULT:         FACC contains floating point sum

ERROR:          ?OVERFLOW ERROR if result too big

EXAMPLE:        LDA #<POINTER
                LDY #>POINTER   ;SET POINTER TO *PACKED* ADDEND
                JSR $AF18       ;ADD MEMORY TO FACC, SUM IN FACC
```

```
NAME:           FADDT   ($AF1B)
FUNCTION:       FACC = ARG + FACC

PREPARATION:    FACC contains floating point addend
                ARG  contains floating point addend
                ARISGN  ($70) contains EOR(FACSGN,ARGSGN)
                .A   contains FACEXP

SPECIAL NOTES:  This routine is similar to FADD. The only
                difference is the call to CONUPK.)

                ***********************************************
                *  You  *MUST*  put resultant sign in ARISGN. *
                *  You  *MUST*  load FACEXP ($63) immediately *
                *  before call so that status flags are set!  *
                ***********************************************

RESULT:         FACC contains floating point sum

ERROR:          ?OVERFLOW ERROR if result too big

EXAMPLE:        LDA FACSGN
                EOR ARGSGN
                STA ARISGN      ;SET RESULTANT SIGN
                LDA FACEXP      ;SET STATUS FLAGS PER FACEXP
                JSR $AF1B       ;ADD ARG TO FACC, SUM IN FACC
```

```
NAME:           FMULT   ($AF1E)
FUNCTION:       FACC = MEMORY * FACC

PREPARATION:    FACC contains floating point multiplier
                .A = pointer (lsb) to packed floating point multiplicand
                .Y = pointer (msb) to packed floating point multiplicand

SPECIAL NOTES:  The multiplicand *MUST* be in RAM-1 or common RAM in
                packed format. FMULT calls CONUPK to normalize it.

RESULT:         FACC contains floating point product

ERROR:          ?OVERFLOW ERROR if result too big

EXAMPLE:        LDA #<POINTER
                LDY #>POINTER   ;SET POINTER TO *PACKED* MULTIPLICAND
                JSR $AF1E       ;MULTIPLY MEMORY BY FACC, PRODUCT IN FACC
```

```
NAME:           FMULTT  ($AF21)
FUNCTION:       FACC = ARG * FACC

PREPARATION:    FACC contains floating point multiplier
                ARG  contains floating point multiplicand

SPECIAL NOTES:  This routine is similar to FMULT. The only difference
                is the call to CONUPK- FMULTT assumes you have already
                loaded ARG with unpacked multiplicand.)

RESULT:         FACC contains floating point product

ERROR:          ?OVERFLOW ERROR if result too big

EXAMPLE:        JSR $AF21       ;MULTIPLY ARG BY FACC, PRODUCT IN FACC
```

```
NAME:           FDIV    ($AF24)
FUNCTION:       FACC = MEMORY / FACC

PREPARATION:    FACC contains floating point divisor
                .A = pointer (lsb) to packed floating point dividend
                .Y = pointer (msb) to packed floating point dividend

SPECIAL NOTES:  The dividend *MUST* be in RAM-1 or common RAM in
                packed format. FDIV calls CONUPK to normalize it.

RESULT:         FACC contains floating point quotient

ERROR:          ?DIVISION BY ZERO ERROR if FACC zero

EXAMPLE:        LDA #<POINTER
                LDY #>POINTER   ;SET POINTER TO *PACKED* DIVIDEND
                JSR $AF24       ;DIVIDE MEMORY BY FACC, QUOTIENT IN FACC
```

```
NAME:           FDIVT   ($AF27)
FUNCTION:       FACC = ARG / FACC

PREPARATION:    FACC contains floating point divisor
                ARG  contains floating point dividend
                ARISGN  ($70) contains EOR(FACSGN,ARGSGN)
                .A   contains FACEXP

SPECIAL NOTES:  This routine is similar to FDIV. The only difference
                is the call to CONUPK- FDIVT assumes you have already
                loaded ARG with unpacked dividend.)

                ***********************************************
                *  You  *MUST*  put resultant sign in ARISGN. *
                *  You  *MUST*  load FACEXP ($63) immediately *
                *  before call so that status flags are set!  *
                ***********************************************

RESULT:         FACC contains floating point quotient

ERROR:          ?DIVISION BY ZERO ERROR if FACC zero

EXAMPLE:        LDA FACSGN
                EOR ARGSGN
                STA ARISGN      ;SET RESULTANT SIGN
                LDA FACEXP      ;SET STATUS FLAGS PER FACEXP
                JSR $AF27       ;DIVIDE ARG BY FACC, QUOTIENT IN FACC
```

```
NAME:           LOG     ($AF2A)
FUNCTION:       FACC = LOG(FACC)        natural logarithm (base e)

PREPARATION:    FACC contains floating point number

RESULT:         FACC contains floating point logarithm

ERROR:          ?ILLEGAL QUANTITY ERROR if FACC negative or zero

EXAMPLE:        JSR $AF2A       ;FACC = LOG(FACC)
```

=-=

```
NAME:           INT     ($AF2D)
FUNCTION:       FACC = INT(FACC)

PREPARATION:    FACC contains floating point number

RESULT:         FACC contains floating point greatest integer

EXAMPLE:        JSR $AF2D       ;FACC = INT(FACC)
```

=-=

```
NAME:           SQR     ($AF30)
FUNCTION:       FACC = SQR(FACC)

PREPARATION:    FACC contains floating point number

RESULT:         FACC contains floating point square root

ERROR:          ?ILLEGAL QUANTITY ERROR if FACC negative

EXAMPLE:        JSR $AF30       ;FACC = SQR(FACC)
```

=-=

```
NAME:           NEGOP   ($AF33)
FUNCTION:       FACC = -FACC            (invert sign of FACC)

PREPARATION:    FACC contains floating point number

RESULT:         FACC contains floating point number with sign inverted

EXAMPLE:        JSR $AF33       ;FACC = -FACC
```

=-=

```
NAME:             FPWR    ($AF36)
FUNCTION:         FACC = ARG ^ MEMORY

PREPARATION:      ARG contains floating point number
                  .A = pointer (lsb) to packed floating point power
                  .Y = pointer (msb) to packed floating point power

SPECIAL NOTES:    The power *MUST* be in ROM or common RAM in packed
                  format as FPWR calls MOVFM to unpack it into FACC.

RESULT:           FACC contains floating point result

ERROR:            ?ILLEGAL QUANTITY ERROR if ARG negative
                  ?OVERFLOW ERROR if result too big

EXAMPLE:          LDA #<POINTER
                  LDY #>POINTER   ;SET POINTER TO *PACKED* POWER
                  JSR $AF36       ;COMPUTE ARG ^ MEM, RESULT IN FACC

=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

NAME:             FPWRT   ($AF39)
FUNCTION:         FACC = ARG ^ FACC

PREPARATION:      ARG  contains floating point number
                  FACC contains floating point power
                  .A   contains FACEXP

SPECIAL NOTES:    This routine is similar to FPWR. The only difference
                  is the call to MOVFM- FPWRT assumes you have already
                  loaded FACC with unpacked power.
                  ***********************************************
                  *  You  *MUST*  load FACEXP ($63) immediately *
                  *  before call so that status flags are set!  *
                  ***********************************************

RESULT:           FACC contains floating point result

ERROR:            ?ILLEGAL QUANTITY ERROR if ARG negative
                  ?OVERFLOW ERROR if result too big

EXAMPLE:          LDA FACEXP      ;SET STATUS FLAGS PER FACEXP
                  JSR $AF39       ;COMPUTE ARG ^ FACC, RESULT IN FACC

=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

NAME:             EXP     ($AF3C)        (compute e ^ FACC)
FUNCTION:         FACC = EXP(FACC)

PREPARATION:      FACC contains floating point number

RESULT:           FACC contains floating point result

ERROR:            ?OVERFLOW ERROR if FACC too big

EXAMPLE:          JSR $AF3C       ;FACC = EXP(FACC)

=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

```
NAME:           COS     ($AF3F)
FUNCTION:       FACC = COS(FACC)

PREPARATION:    FACC contains floating point number

RESULT:         FACC contains floating point cosine (in radians)

EXAMPLE:        JSR $AF3F       ;FACC = COS(FACC)
```

=-=

```
NAME:           SIN     ($AF42)
FUNCTION:       FACC = SIN(FACC)

PREPARATION:    FACC contains floating point number

RESULT:         FACC contains floating point sine (in radians)

EXAMPLE:        JSR $AF42       ;FACC = SIN(FACC)
```

=-=

```
NAME:           TAN     ($AF45)
FUNCTION:       FACC = TAN(FACC)

PREPARATION:    FACC contains floating point number

RESULT:         FACC contains floating point tangent (in radians)

EXAMPLE:        JSR $AF45       ;FACC = TAN(FACC)
```

=-=

```
NAME:           ATN     ($AF48)
FUNCTION:       FACC = ATN(FACC)

PREPARATION:    FACC contains floating point number

RESULT:         FACC contains floating point arctangent (in radians)

EXAMPLE:        JSR $AF48       ;FACC = ATN(FACC)
```

=-=

```
NAME:           ROUND   ($AF4B)         (round to 40 bits of precision)
FUNCTION:       FACC = FACC + FACOV(msb)

PREPARATION:    FACC contains floating point number
                FACOV (msb) contains 'extra' precision

RESULT:         none if FACC zero or FACOV (msb) zero
                one extra bit ADDED to FACC lsb if FACOV (msb) is set

EXAMPLE:        JSR $AF4B       ;ROUND FACC
```

```
NAME:           ABS     ($AF4E)         (make FACSGN(msb) = $00)
FUNCTION:       FACC = ABS(FACC)

PREPARATION:    FACC contains (SIGNED) floating point number

RESULT:         FACC contains (POSITIVE) floating point

EXAMPLE:        JSR $AF4E       ;FACC = ABS(FACC)
```

```
NAME:           SGN     ($AF51)         (test SIGN of FACC)
FUNCTION:       .A = SGN(FACC)

PREPARATION:    FACC contains floating point number

RESULT:         .A  --> $FF if FACC negative (FACC < 0)
                        $00 if FACC zero     (FACC = 0)
                        $01 if FACC positive (FACC > 0)
                (status flags reflect contents of .A, carry invalid)

EXAMPLE:        JSR $AF51       ;SGN(FACC)
                                ; BEQ will trap =0
                                ; BNE will trap <>0
                                ; BMI will trap <0
                                ; BPL will trap >=0     etc.
```

NAME: FCOMP ($AF54) (compare FACC with MEMORY)
FUNCTION: .A = FCOMP(FACC,MEMORY)

PREPARATION: FACC contains floating point number
.A = pointer (lsb) to packed floating point number
.Y = pointer (msb) to packed floating point number

SPECIAL NOTES: The number *MUST* be in ROM, or RAM currently in context below ROM, in PACKED format. *** FACOV is significant!

RESULT:
```
.A  --> $FF if FACC < MEMORY
        $00 if FACC = MEMORY
        $01 if FACC > MEMORY
(status flags reflect contents of .A, carry invalid)
```

EXAMPLE:
```
LDA #<POINTER
LDY #>POINTER    ;SET POINTER TO *PACKED* NUMBER
JSR $AF54        ;COMPARE FACC WITH MEMORY
                 ; BEQ will trap FACC =  MEM
                 ; BNE will trap FACC <> MEM
                 ; BMI will trap FACC <  MEM
                 ; BPL will trap FACC >= MEM     etc.
```

NAME: RND0 ($AF57)
FUNCTION: FACC = random floating point number (0<n<1)

PREPARATION:
```
.A  --> $00 to generate a 'true' random number
        $01 to generate next random number in sequence
        $FF to start a new sequence of random numbers
            based upon current contents of FACC.
```

SPECIAL NOTES: *MUST* be called with the system (BANK 15) in context. *MUST* load .A immediately before call so that status flags reflect contents of .A

RESULT: FACC = floating point random number

EXAMPLE:
```
LDA #$FF         ;START REPRODUCEABLE SEQUENCE BASED ON FACC
JSR $AF57
LDA #$01
JSR $AF57        ;GENERATE (FIRST) RANDOM NUMBER IN SEQUENCE
```

NAME: CONUPK ($AF5A)
FUNCTION: ARG = UNPACK(RAM_CONSTANT)

PREPARATION: .A = pointer (lsb) to packed floating point number
.Y = pointer (msb) to packed floating point number

SPECIAL NOTES: The number *MUST* be in RAM-1 or common RAM in packed format.

RESULT:
ARG loaded with normalized floating point number
ARISGN ($6F) contains EOR(FACSGN,ARGSGN)
.A contains FACEXP (status reflects contents of .A)

EXAMPLE:

```
LDA #<POINTER
LDY #>POINTER      ;SET POINTER TO *PACKED* NUMBER IN RAM-1
JSR $AF5A          ;LOAD ARG
                   ; BEQ traps ARG = $00
```

=-=

NAME: ROMUPK ($AF5D)
FUNCTION: ARG = UNPACK(ROM_CONSTANT)

PREPARATION: .A = pointer (lsb) to packed floating point number
.Y = pointer (msb) to packed floating point number

SPECIAL NOTES: The number *MUST* be in ROM or RAM currently in context
(otherwise identical to CONUPK).

RESULT:
ARG loaded with normalized floating point number
ARISGN ($6F) contains EOR(FACSGN,ARGSGN)
.A contains FACEXP (status reflects contents of .A)

EXAMPLE:

```
LDA #<POINTER
LDY #>POINTER      ;SET POINTER TO *PACKED* NUMBER IN RAM-1
JSR $AF5D          ;LOAD ARG
                   ; BEQ traps ARG = $00
```

=-=

NAME: MOVFRM ($AF60)
FUNCTION: FACC = UNPACK(RAM_CONSTANT)

PREPARATION: .A = pointer (lsb) to packed floating point number
.Y = pointer (msb) to packed floating point number

SPECIAL NOTES: The number *MUST* be in RAM-1 or common RAM in packed format.
MOVFRM is *NEW*, and not a true part of the math package!

```
**********************************************
*  *MUST* be called from ROM or common RAM-  *
*  this routine RTS's with RAM-1 in context! *
**********************************************
```

RESULT:
FACC loaded with normalized floating point number
FACOV ($71) cleared

EXAMPLE:

```
LDA #<POINTER
LDY #>POINTER      ;SET POINTER TO *PACKED* NUMBER IN RAM-1
JSR $AF60          ;LOAD FACC
```

=-=

```
NAME:           MOVFM   ($AF63)
FUNCTION:       FACC = UNPACK(ROM_CONSTANT)

PREPARATION:    .A = pointer (lsb) to packed floating point number
                .Y = pointer (msb) to packed floating point number

SPECIAL NOTES:  The number *MUST* be in ROM or RAM currently in context.

RESULT:         FACC    loaded with normalized floating point number
                FACOV   ($71) cleared
                .A      contains FACEXP (status reflects contents of .A)

EXAMPLE:        LDA #<POINTER
                LDY #>POINTER   ;SET POINTER TO *PACKED* NUMBER IN ROM
                JSR $AF63       ;LOAD FACC
                                ; BEQ traps ARG = $00
```

==========

```
NAME:           MOVMF   ($AF66)
FUNCTION:       MEMORY = PACK(ROUND(FACC))

PREPARATION:    FACC contains floating point number
                .X = pointer (lsb) to destination
                .Y = pointer (msb) to destination

SPECIAL NOTES:  The destination will be to RAM currently in context
                (*YOU* must configure memory- routine assumes writes
                to ROM 'bleed through' to RAM in context, but be sure
                I/O is turned off!)

RESULT:         FACC    will be ROUNDed and FACOV cleared.
                MEMORY  thru MEMORY+4 will contain *PACKED* number.

EXAMPLE:        LDX #<POINTER
                LDY #>POINTER   ;POINTER TO RAM DESTINATION IN CURRENT BANK
                JSR $AF66       ;STORE FACC (PACKED)
```

==========

```
NAME:           MOVFA   ($AF69)
FUNCTION:       FACC = ARG

PREPARATION:    ARG contains floating point number

RESULT:         FACC    contains same number as ARG
                FACOV   ($71) cleared
                .A      contains FACEXP (but status invalid!)

EXAMPLE:        JSR $AF69       ;COPY ARG TO FACC
```

```
NAME:           MOVAF   ($AF6C)
FUNCTION:       ARG = FACC

PREPARATION:    FACC contains floating point number

RESULT:         FACC    will be ROUNDed and FACOV cleared.
                ARG     contains same number as FACC
                .A      contains FACEXP (but status invalid!)

EXAMPLE:        JSR $AF6C       ;COPY FACC TO ARG
```

*** End of MATH ROUTINE documentation ***

REVISION HISTORY:

v0.1	-	4/22/86	Original limited distribution	(FAB)
v0.2	-	6/17/86	Corrections to: EXP ($AF3C) MOVMF ($AF66)	(FAB)

Commodore®
AMIGA®

Commodore Business Machines, Inc.
1200 Wilson Drive
West Chester, PA 19380