Apple® IIc Programmer's Guide to the 3.5 ROM

Apple® IIc Programmer's Guide to the 3.5 ROM

030-1196-A

Reorder Apple Product #A2L4037

Contents

Preface

Read Me First

This reference manual is intended for experienced Apple® IIc assembly-language programmers and hardware designers. It is a supplement to the *Apple IIc Reference Manual*. If you're a beginning user, you should learn about your Apple IIc and some of the programs, languages, and devices that you intend to use before you start reading this manual. A list of some Apple manuals that will help you learn about the operation and function of Apple II family computers is given at the end of this preface.

This manual assumes that you have access to Apple manuals describing the programs, languages, and devices that you intend to use with your Apple IIc.

About This Manual

The Programmer's Guide that you are reading describes the differences between the original Apple IIc and the Apple IIc with 32-kilobyte (32K) ROM.

3.5 ROM

The name *3.5 ROM* refers to the ROM that is described in this book as the *32K ROM*.

Chapter 1 details the changes brought about by the ROM enhancement. It also refers to other sources of information about Apple IIc operation.

Chapter 2 describes Monitor enhancements, including the Mini-Assembler and the STEP and TRACE routines.

Chapter 3 describes the use of the Protocol Converter and Protocol Converter calls.

Appendix A is an assembly listing of the firmware for the 32K ROM.

Where to Look for More Information

The following manuals have information about the function and programming of Apple II-family computers and some important peripherals.

These books are available from your Apple dealer:

- *About Your Enhanced Apple IIe: Programmer's Guide* (030-1143-A)
- *Apple IIc Owner's Manual* (030-1030-A)
- *Apple Pascal 1.2 Update Manual* (030-0602)
- *Apple Pascal 1.3 Update Manual* (030-1206-A)
- *Synertek Programming Manual* (A2L0003)

These books are available at your local bookstore:

- *Apple IIc Reference Manual* (Addison-Wesley 17727-7)
- *ProDos Technical Reference Manual* (Addison-Wesley 17728-5)

Watch for These

Look for these throughout the manual:

By the Way: Text set off in this manner presents sidelights or interesting pieces of information.

Important Text set off in this manner—with a tag in the margin—presents important information.

▲Warning Warnings like this indicate potential problems or disasters.

Apple IIe Text set off in this manner—with the name of an Apple computer or peripheral device in the margin—applies specifically to that computer or device.

Chapter 1 Overview

This chapter tells you about the operating differences between the original Apple® IIc and the Apple IIc with 32K ROM.

The following topics are discussed in this chapter:

- Physical changes in the Apple IIc
- Machine identification
- Features removed
- Interrupt handler revision
- Starting up from drives other than a Disk II
- New serial port commands
- Monitor enhancements, including the Mini-Assembler and the STEP and TRACE routines
- The Protocol Converter
- The Protocol Converter Bus

The Monitor enhancements and the Protocol Converter are described in detail in the following chapters.

Physical Changes

The 32K ROM for the Apple IIc replaces the original 16K ROM. The 32K ROM is organized into two 16K segments called the main and alternate banks. To toggle between banks, read address $C028. A write to $C028 toggles the line twice, leaving it unchanged.

Installing the new ROM involves cutting one trace and jumping another on the Apple IIc main logic board. Once the Apple IIc has been modified for the new ROM, the old ROM cannot be used in it.

▲Warning Improperly done modifications can damage your Apple IIc. Modifications performed by anyone other than an authorized Apple dealer void your warranty.

Machine Identification

Your Apple II program can tell which member of the Apple II family it is running on by checking the values in four locations of ROM (the *identification bytes*). Table 1-1 lists the machines and their identification bytes.

Table 1-1. Apple II Family Identification Bytes

Machine	$FBB3	$FB1E	$FBC0	$FBBF
Apple II	$38			
Apple II Plus	$EA	$AD		
Apple III (emulation)	$EA	$8A		
Apple IIe (original)	$06		$EA	
Apple IIe (enhanced)	$06		$E0	
Apple IIc (original)	$06		$00	$FF
Apple IIc (32K ROM)	$06		$00	$00

The only difference between the identification bytes for the original Apple IIc and the Apple IIc with 32K ROM is at $FBBF. That location was $FF in the original ROM and is $00 in the current revision of the 32K ROM.

If you're not sure whether a particular Apple IIc has the 32K ROM installed, follow the instructions in Chapter 2 for starting the Mini-Assembler. If you are successful (the Monitor prompt character changes from * to !), then the 32K ROM is present.

Features Removed

PR#7 (from BASIC) or 7 CONTROL-P (from the Monitor) no longer causes the system to **boot** from the external Disk II drive because that ROM space is now used for new features.

Interrupt Handler Revision

If the alternate bank of the ROM is being used, the interrupt handler switches the ROM to the main bank before calling a user's interrupt routine, and restores the alternate bank before returning from the interrupt. The bank of ROM to be returned to is indicated by the least significant bit of the byte stored in address $0044 after a break: 0 for the main bank, and 1 for the alternate bank of the ROM.

Programs written for the Apple II, II Plus, or IIe sometimes read $C02x before a BRK instruction that attempts to jump to a monitor routine. Reading $C02x toggles the cassette output signal in the Apple II, II Plus, and IIe, but switches between the main and alternate banks of ROM in the Apple IIc with 32K ROM. When the interrupt handler detects that a program

has tried to jump to a monitor routine while in the alternate bank of the ROM, it automatically switches to the main bank of ROM and attempts to restart the program at the point before the break occurred. Note that this feature does not work for programs that try to read monitor data after a $C02x access because, in this case, no BRK instruction is executed.

Starting and Restarting

The Protocol Converter: see Chapter 3.

When the Apple IIc with 32K ROM is powered up, it attempts to boot from the built-in disk drive. If this fails, it attempts to boot from the first device attached to the Protocol Converter. If this fails too, the message `CHECK DISK DRIVE` is displayed and the system hangs.

The built-in Disk II can be booted with a `PR#6` command from BASIC, a 6 CONTROL-P from the Monitor, or `JMP $C600` from a machine-language program.

An external UniDisk™ 3.5 can be booted with a `PR#5` command from BASIC, a 5 CONTROL-P from the Monitor, or `JUMP $C500` from a machine-language program.

Important External UniDisk 3.5 startup works with ProDOS®-based and Pascal 1.3 programs, but not with earlier versions of Pascal or with DOS.

New Serial Port Commands

The serial ports and the use of serial port commands are discussed in Chapters 7 and 8 of the *Apple IIc Reference Manual*.

Several new commands that can be used with serial ports 1 and 2 have been included in the 32K ROM. Each of these commands consists of two letters: the first letter identifies the command while the second letter enables (E) or disables (D) the function. You may separate the command and the letter with a space if you wish.

For example, to cause a line feed character to be output after every carriage return, use the command LE (or L E). To disable this function, use the command LD (or L D).

C D/E
(Column overflow)

When enabled, this command causes a carriage return character to be sent automatically any time the column count exceeds the printer line width. Default = enabled.

F D/E
(Find keyboard)

When enabled, this command causes your Apple IIc to accept signals coming from its keyboard as well as those coming over the serial port. You can use this command to disable the keyboard before receiving data or sending data to the printer, to prevent accidental keystrokes from disrupting the data flow. Be sure your program re-enables the keyboard when the data transfer is complete. This feature is available only in BASIC. Default = enabled.

L D/E
(Line feed after carriage return)

When enabled, this command causes a line feed character to be output automatically after each carriage return character. LD and LE have the same effects as commands L (enable line feed) and K (disable line feed), which still work. Default = disabled.

M D/E
(Mask line feeds)

When enabled, all incoming line feed characters are removed from the data stream. Default = enabled.

X D/E
(XON/XOFF protocol)

When enabled, XON/XOFF protocol is used: the Apple IIc looks for any XOFF character (ASCII character DC3, decimal 19), and responds by halting transmission until an XON character (ASCII character DC1, decimal 17) is received. Default = disabled.

Monitor Enhancements

The Apple IIc with 32K ROM includes the Mini-Assembler, which lets you enter machine-language programs directly from the keyboard; and the STEP and TRACE Monitor routines, which facilitate debugging of machine-language programs. The Mini-Assembler and the STEP and TRACE routines are discussed in detail in Chapter 2.

Important | Monitor commands can't be executed directly from the Mini-Assembler on an Apple IIc with 32K ROM.

The Protocol Converter

The Protocol Converter is a set of routines used to support I/O devices, such as the UniDisk 3.5, that connect to the external disk port. The routines begin at address $C500 in ROM. ProDOS and Pascal 1.3 recognize the Protocol Converter as a block device. The Protocol Converter and calls to the Protocol Converter are described in detail in Chapter 3.

The Protocol Converter Bus

The Protocol Converter Bus (CBus) consists of hardware and software components that permit and control communications between the Apple IIc and intelligent I/O devices (such as UniDisk 3.5's) connected to its external disk port.

The software part of the CBus includes the Protocol Converter and the CBus communication protocol.

The hardware component of the CBus is a daisy chain made up of the following:

- The Apple IIc disk port, using the disk controller unit (IWM), see Chapter 11 in the *Apple IIc Reference Manual*.
- One or more intelligent I/O devices (*bus residents*).
- One Disk IIc (optional). If included, the Disk IIc must be the terminal member of the daisy chain, and remains dormant when a bus resident is addressed.

The maximum number of bus residents is limited by the Apple IIc's power supply and IWM drive capacity. The software can support up to 127 bus residents.

Chapter 2 Monitor Enhancements

This chapter is about enhancements made to the Monitor, including the addition of the Mini-Assembler (which allows machine-language programs to be entered directly from the keyboard), and debugging routines for assembly-language programs. The following topics are discussed in this chapter:

- Procedures for using the Mini-Assembler
- The Mini-Assembler commands
- The STEP and TRACE debugging routines

The Mini-Assembler

Without an assembler, you have to write your machine-language program, take the hexadecimal values for the opcodes and operands, and store them in memory using the monitor commands described in Chapter 10 of the *Apple IIc Reference Manual*.

The Mini-Assembler lets you enter machine-language programs directly from the keyboard of your Apple. You can type ASCII characters in Mini-Assembler programs, exactly as you type them in the Monitor.

Important The Mini-Assembler doesn't accept labels; you must use actual hexadecimal values and addresses.

Starting the Mini-Assembler

To start the Mini-Assembler, first call the Monitor by typing `CALL-151` and pressing RETURN. Then from the Monitor, type `!` and press RETURN. The Monitor prompt character then changes from `*` to `!`.

When you finish using the Mini-Assembler, press RETURN from a blank line to return to the Monitor.

To enter code into memory, type the address, a colon, and the instruction. For example, after entering the Mini-Assembler, type

```
!300:STA C030
```

You can enter a series of instructions by typing a space, then the instruction, followed by RETURN:

```
!300:STA C030
! LDA #A0
! INX
```

Each succeeding instruction is placed in the next consecutive memory location. As you type in instructions, each is replaced by the starting address of the instruction, the hexadecimal value(s) of the instruction, followed by mnemonics describing the instruction. For example, the sequence of instructions given above produces the following on your display screen:

```
0300-   8D 30 C0        STA $C030
0303-   A9 A0           LDA #$A0
0305-   E8              INX
```

When you're ready to execute your program, press RETURN to leave the Mini-Assembler and return to the Monitor.

Important Monitor commands can't be executed directly from the Mini-Assembler on an Apple IIc with 32K ROM.

Using the Mini-Assembler

The Mini-Assembler saves one address, that of the program counter. Before you start to type a program, you must set the program counter to point to the location where you want the Mini-Assembler to store your program. Do this by typing the address followed by a colon.

Formats for operands are listed in Table 2-1.

After the colon, type the mnemonic for the first instruction in your program, followed by a space and the operand of the instruction. Now press RETURN. The Mini-Assembler converts the line you typed into hexadecimal, stores it in memory beginning at the location of the program counter, and then disassembles it again and displays the disassembled line. It then displays a prompt character on the next line.

Now the Mini-Assembler is ready to accept the second instruction in your program. To tell it that you want the next instruction to follow the first, don't type an address or a colon: just type a space and the next instruction's mnemonic and operand, then press RETURN. The Mini-Assembler assembles that line and waits for another.

If the line you type has an error in it, the Mini-Assembler causes the Apple IIc to beep and display a caret (^) under or near the offending character in the input line. Most common errors are the result of typographical mistakes: misspelled mnemonics, missing parentheses, and so forth. The Mini-Assembler also rejects the input line if you forget the space before or after a mnemonic or include an extraneous character in a hexadecimal value or address. If the destination address of a branch instruction is out of the range of the branch (more than 127 locations distant from the address of the instruction), the Mini-Assembler flags this as an error.

> *Dollar Signs:* In this book, dollar signs ($) in addresses signify that the addresses are in hexadecimal notation. The dollar signs are ignored by the Mini-Assembler and may be omitted when typing programs.

```
!300:LDX #02

0300-   A2 02       LDX    #$02
! LDA $0,X

0302-   B5 00       LDA    $00,X
! STA $10,X

0304    95 10       STA    $10,X
! DEX

0306-   CA          DEX
! STA $C030

0307-   8D 30 C0    STA    $C030
! BPL $302

030A-   10 F6       BPL    $0302
! BRK

030C-   00          BRK
!
```

To leave the Mini-Assembler and return to the Monitor, press RETURN at a blank line.

Your assembly-language program is now stored in memory. You can display it with the LIST command:

```
*300L

0300-  A2 02        LDX   #$02
0302-  B5 00        LDA   $00,X
0304-  95 10        STA   $10,X
0306-  CA           DEX
0307-  8D 30 C0     STA   $C030
030A-  10 F6        BPL   $0302
030C-  00           BRK
030D-  00           BRK
030E-  00           BRK
030F-  00           BRK
0310-  00           BRK
0311-  00           BRK
0312-  00           BRK
0313-  00           BRK
0314-  00           BRK
0315-  00           BRK
0316-  00           BRK
0317-  00           BRK
0318-  00           BRK
0319-  00           BRK
*
```

Mini-Assembler Instruction Formats

The Apple IIc Mini-Assembler recognizes 66 mnemonics and 15 addressing formats. The mnemonics are standard, as used in the *Synertek Programming Manual*, but the addressing formats are somewhat different; see Table 2-1.

Table 2-1. Mini-Assembler Address Formats

Addressing Mode	Format
Accumulator	*
Implied	*
Immediate	#${value}
Absolute	${address}
Zero page	${address}
Indexed zero page	${address},X ${address},Y
Indexed absolute	${address},X ${address},Y
Relative	${address}
Indexed indirect	(${address},X)
Indirect indexed	(${address}),Y
Absolute indirect	(${address})

* These instructions have no operands.

An address consists of one or more hexadecimal digits. The Mini-Assembler interprets addresses the same way the Monitor does: if an address has fewer than four digits, the Mini-Assembler adds leading zeros; if the address has more than four digits, then it uses only the last four.

There is no syntactical distinction between the absolute and zero page addressing modes. If you give an instruction to the Mini-Assembler that can be used in both absolute and zero page mode, the Mini-Assembler assembles that instruction in absolute mode if the operand for that instruction is greater than $FF, and it assembles it in zero page mode if the operand is less than $100.

Instructions in accumulator mode and implied addressing mode need no operands.

Branch instructions, which use the relative addressing mode, require the target address of the branch. The Mini-Assembler automatically calculates the relative distance to use in the instruction. If the target address is more than 127 locations distant from the instruction, the Mini-Assembler sounds a bell (beep), displays a caret (^) under the target address, and does not assemble the line.

If you give the Mini-Assembler the mnemonic for an instruction and an operand, and the addressing mode of the operand cannot be used with the instruction you entered, then the Mini-Assembler will not accept the line.

STEP and TRACE

STEP and TRACE are Monitor facilities for debugging assembly-language programs. The STEP command decodes, displays, and executes one instruction at a time, and the TRACE command steps continuously through a program, stopping when a BRK instruction is executed or [apple] is pressed. You can press [T] to slow down the trace to one step per second.

Each STEP command causes the Monitor to execute the instruction in memory pointed to by the Program Counter. The instruction is displayed in its disassembled form, then executed. The contents of the 65C02's internal registers are displayed after the instruction is executed. After execution, the Program Counter is incremented to point to the next instruction in the program.

Here is an example of the STEP command, using the following program:

```
$0300:    LDX #02
$0302:    LDA $00,X
$0304:    STA $10,X
$0306:    DEX
$0307:    STA $C030
$030A:    BPL $0302
$030C:    BRK
```

To step through this program, first call the Monitor by typing `CALL-151` and pressing [RETURN], and then from the Monitor, type `300S` (to start the STEP routine at address $0300). Type [S] to advance each additional step

through the program. The Monitor keeps the Program Counter and the last opened address separate from one another, so you can examine or change the contents of memory while you are stepping through your program.

Here's what happens when you step through the program above, examining the contents of location $0012 after the third step. Note that in this example, what you type appears just after the * prompt, and the information on the next two lines—that begin without the * prompt—is what the computer displays on the screen in response.

```
*300S

0300-   A2 02     LDX #02
M=CA A=0A X=02 Y=D8 P=30 S=F8
*S

0302-   B5 00     LDA $00,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
*S

0304-   95 10     STA $10,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
*12

0012- 0C
*S

0306-   CA        DEX
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S

0307-   8D 30 C0 STA $C030
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S

030A-   10 F6     BPL $0302
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S

0302-   B5 00     LDA $00,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*S

0304-   95 10     STA $10,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*
```

The TRACE command is a continuous version of the STEP command; it will stop stepping through the program only when you press [apple], or when it encounters a BRK instruction in the program. Press [control] to slow the trace to one step per second.

Cautions

Keep the following cautions in mind when using the STEP and TRACE Monitor commands:

- If the program ends with an RTS instruction, the TRACE routine will continue to run indefinitely until stopped with [apple].
- You can't step or trace through routines that use the same zero page locations as the Monitor.

ASCII Input Mode

This mode lets you enter ASCII characters as well as their hexadecimal ASCII equivalents. This means that 'A is the same as C1 and 'B is the same as C2 to the Monitor. The ASCII value for *any* character following an apostrophe is used by the Monitor. For example, to enter the string "Hooray for sushi!" at $300 in memory, type

```
*300:'H 'o 'o 'r 'a 'y ' 'f 'o 'r ' 's 'u 's 'h 'i '!
```

Note that each character to be placed in memory is delimited by a leading ' and a trailing space. The only exception to this rule is that the last character in the line is followed by a RETURN character instead of a space.

Chapter 3

Calls to the Protocol Converter

This chapter is about the Protocol Converter, which is a set of assembly-language routines used to support external I/O devices, such as UniDisk 3.5. To ProDOS and Pascal 1.3, the Protocol Converter appears to be a block device.

The following topics are discussed in this chapter:

- How to locate the Protocol Converter
- How to issue a call to the Protocol Converter
- The use of each call
- The parameters required for each call
- Possible errors codes returned for each call
- The possible causes of the errors

At the end of this chapter is an example of an assembly-language program that uses a Protocol Converter call.

Locating the Protocol Converter

The code for the Protocol Converter always begins at address $C500 in the Apple IIc with 32K ROM. To ensure compatibility of your programs with the Apple IIe, however, your Protocol Converter routines should always begin with a search for the Protocol Converter by looking for the following bytes: CN01 = $20, CN03 = $00, CN05 = $03, and $CN07 = 00, where N can be from 1 to 7. The Protocol Converter entry point is then at address $CN00 + ($CNFF) + 3. The sample program at the end of this chapter illustrates such a search.

How to Issue a Call to the Protocol Converter

MLI calls: see the *ProDos Technical Reference Manual*, Chapter 4.

Protocol Converter calls are coded in a manner similar to ProDOS Machine Language Interface (MLI) calls: The program executes a JSR (jump to subroutine) to a dispatch routine at address $C500 + ($C5FF) + 3, where ($C5FF) refers to the value of the byte located at $C5FF.

The number of the Protocol Converter call and a two-byte pointer to the call's parameter list must immediately follow the call. Here is an example of a call to the Protocol Converter:

```
IWMCALL JSR DISPATCH ;Call PC command dispatcher
        DFB CMDNUM   ;This specifies the command type
        DW  CMDLIST  ;2-byte (low,high) pointer to parameter list
        BCS ERROR    ;Carry is set on an error
```

The command number determines the Protocol Converter call to be used. The length and contents of the parameter list depend on the call, as described in Section "Descriptions of the Protocol Converter Calls."

Upon completion of the call, the program resumes execution at the statement following the pointer to the parameter list. In this example, the DFB and DW statements are skipped, and execution resumes with the BCS statement. If the call is successful, the C flag (in the Processor Status register of the 65C02 microprocessor) is cleared (0), and the accumulator (the A register) is cleared to all zeros. If the call is unsuccessful, the C flag is set (1), and the error code is placed in the A register. After the Protocol Converter call, the contents of the 65C02's registers are as follows:

Register:	**Processor Status**							**X**	**Y**	**A**	**PC**	**S**
Bit:	**N**	**Z**	**C**	**D**	**V**	**I**	**B**					
Successful call:	x	x	0	0	x	u	u	x	x	0	JSR+3	u
Unsuccessful call:	x	x	1	0	x	u	u	x	x	Error	JSR+3	u

x = undefined, except in cases where index information is returned in X and Y

u = unchanged

Most Protocol Converter calls include a two-byte pointer to a parameter list, which may contain information to be used by the call, or provide space for information to be returned by the call.

Cautions

You *must* observe the following cautions when using the Protocol Converter, or *your program will crash*:

- The Protocol Converter requires up to 35 bytes of stack space. Be sure you take this into account when calculating the stack space used by your program.

  Failure to allow for the stack space used by the Protocol Converter can result in a stack overflow, causing your program to crash when it attempts to access the data that have been overwritten.
- Data cannot be read from the Protocol Converter into RAM that is not both read-enabled and write-enabled. The Protocol Converter must be able to read from the RAM after writing to it, to obtain a checksum. Failure to observe this rule results in an error (BUSERR $06).

Reading and writing to RAM: see Section "Bank-Switched Memory" in the *Apple IIc Reference Manual*.

- Do not attempt to use the Protocol Converter to put anything into zero page locations. These locations are reserved for temporary storage of data by the Protocol Converter.

Descriptions of the Protocol Converter Calls

Calls to the Protocol Converter are used

- to obtain status information about a device
- to reset a device
- to format the medium in a device
- to read from a device
- to write to a device
- to send control information to a device.

The Protocol Converter calls, in command-number sequence, are:

STATUS ($00)	Returns status information about a particular device, including general status (character or block device, read or write protection, format allowed, device on line); the device control block (set with the CONTROL call); the device newline status (character devices only); and device-specific information (number of blocks, ID string, device name, device type, device firmware version).
READ BLOCK ($01)	Reads one 512-byte block from a disk device, and writes it to memory.
WRITE BLOCK ($02)	Writes one 512-byte block from memory to a disk device.
FORMAT ($03)	Prepares all blocks on a block device for reading and writing.
CONTROL ($04)	Controls some device functions, including warm resets, setting the device control block (which controls global aspects of the device's operating environment), setting newline status (character devices only), and device interrupts. Several CONTROL calls are device-specific.
INIT ($05)	Resets all resident devices. A global reset is done automatically on startup or system resets from the keyboard; an application should never have to reset all devices.
OPEN ($06)	Prepares a character device for reading or writing.
CLOSE ($07)	Tells a character device that a sequence of reads or writes is over.

READ ($08)	Reads a specified number of bytes from a specified device.
WRITE ($09)	Writes a specified number of bytes from memory to a specified device.

Format of Call Descriptions

The following sections describe each protocol converter call, including the command number, the parameter list, and error codes. The calls are discussed in command-number order. Each call is shown in this format:

Command Name The name used to identify the call for descriptive purposes.

Command Number The number, in hexadecimal, that specifies the call to the Protocol Converter.

Parameter List A list of the parameters required for the call.

General Description The purpose and use of the call.

Parameter Descriptions A description of each parameter, and descriptions of data bytes pointed to by parameters. When a parameter is a status or control code, the meaning of each code number is discussed.

Possible Errors A list of the error codes that can be returned by this call. A complete list of Protocol Converter error codes is included at the end of this chapter.

STATUS

Command Number	$00
Parameter List	$03 (parameter count) Unit number Status list pointer (low byte, high byte) Status code

The STATUS call returns status information about a particular device. The type of information returned is determined by the status-code parameter, and the location to which it is returned is determined by the status list pointer.

After a STATUS call has been executed, the 65C02's X and Y registers contain the number of bytes of status information returned (the low byte of this number is in the X register, and the high byte is in the Y register).

Parameter Descriptions

Parameter Count
1-byte value

3 for this call.

Unit Number
1-byte value

The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01—$7E, and are assigned according to the devices' positions in the chain.

Important

Use a unit number of $00 and a status code of $00 in a STATUS call to obtain the status of the Protocol Converter itself (see the discussion under Status Code = $00, below).

Status List Pointer
2-byte value

Points to the buffer to which the status is to be returned. The length required for the buffer varies depending on the status request being made.

Status Code
1-byte value

Indicates what kind of status request is being made. Status codes are in the range $00-$FF, as follows:

Table 3-1. Status Codes

Code	Status Returned
$00	Return device status
$01	Return device control block (DCB)
$02	Return newline status (character devices only)
$03	Return device information block (DIB)
$05	Return UniDisk 3.5 status

UniDisk 3.5

Status codes $01 and $02 are not supported by the UniDisk 3.5 and result in an error (BADCTL $21). Device status for the UniDisk 3.5 can be obtained by using status code $05.

Status Code = $00, Return Device Status The device status consists of four bytes. The first is the general status byte:

Bit	Description
7	0 = character device, 1 = block device
6	1 = write allowed
5	1 = read allowed
4	1 = device on line or disk in drive
3	0 = format allowed
2	0 = medium write protected (block devices only)
1	1 = device currently interrupting
0	1 = device currently open (character devices only)

If the STATUS call is for a block device, the next three bytes (low byte first) are the size in 512-byte blocks. The maximum size is 16 million ($FFFFFF) blocks (about 8 gigabytes). If the call is for a character device, these three bytes must be set to zero.

Unit Number $00: A STATUS call with status code = $00 and unit number = $00 returns the status of the Protocol Converter itself. In this case, the status list consists of eight bytes, as follows:

```
STAT_LIST DFB  Number_Devices    ;Devices hooked to PC
          DFB  Interrupt_Status  ;Bit 6 clear = interrupt sent
          DFB                    ;Reserved
          DFB                    ;Reserved
          DFB                    ;Reserved
          DFB                    ;Reserved
          DFB                    ;Reserved
          DFB                    ;Reserved
```

The Number_Devices byte returns the total number of intelligent devices attached to the Protocol Converter. The Interrupt_Status byte is a copy of the Asynchronous Communications Interface Adapter (ACIA) status register at the time of the interrupt, and is used to indicate that a device requires interrupt servicing. If the sixth bit of this byte equals zero, one or more devices in the Protocol Converter Bus daisy chain must be serviced; your interrupt handler must poll each device on the chain to determine which ones.

ACIA status register: see Section "Firmware Handling of Interrupts" in the *Apple IIc Reference Manual.*

> ***About Interrupts:*** Devices that require interrupt servicing must use the EXTINT line on the external disk port connector of the Apple IIc to be supported by the Protocol Converter. UniDisk 3.5, for example, does not support this line, and so cannot generate interrupts to the Protocol Converter. See Section "CONTROL" for instructions on enabling Protocol Converter interrupts. See Appendix E in the *Apple IIc Reference Manual* for more information about programming with interrupts.

Status Code = $01, Return Device Control Block The device control block (DCB) is used to control various operating characteristics of a device, and is device dependent. Each device has a default DCB, which can be altered with a CONTROL call. The first byte (the *count byte*) gives the number of bytes in the control block (*not* including the count byte), so the length never exceeds 256 bytes (257 including the count byte).

UniDisk 3.5

> UniDisk 3.5 has no DCB, and returns an error (BADCTL $21) in response to this call.

Newline read mode: see Chapter 4 in the *PRODOS Technical Reference Manual.*

Status Code = $02, Return Newline Status Newline status applies only to character devices. Use of statcode = $02 with a block device results in error BADCTL ($21).

Status Code = $03, Return Device Information Block The device's information block contains information identifying the device, its type, and various other attributes. The returned status list has the following form:

```
STAT_LIST DFB   Device_Statbyte1 ;Same as byte 1 in Status Code=0
          DFB   Device_Size_Lo   ;Number of blocks (block device)
          DFB   Device_Size_Med  ;Number of blocks (middle byte)
          DFB   Device_Size_Hi   ;Number of blocks (high byte)
          DFB   ID_String_Length ;Length in bytes (16 max.)
          ASC   '<device name>'  ;7-bit ASCII, uppercase, padded
*                                 with spaces, eighth bit always=0
*                                 (16 bytes)
          DFB   Device_Type_Code
          DFB   Device_Subtype_Code
          DW    Version          ;Device firmware version number
```

Status Code = $05, Return UniDisk 3.5 Status This call allows the diagnostic program to get more detailed information about the cause of a read or write error, and to examine the contents of the 65C02's registers after a CONTROL Protocol Converter call with control code = $05 (see Section "CONTROL"). The returned status list has this form:

```
STAT_LIST DFB   $00
          DFB   Error     ;Soft Error byte (see below)
          DFB   Retries   ;Number of retries (see below)
          DFB   $00
          DFB   A_Value   ;Acc value after a CONTROL EXECUTE call
          DFB   X_Value   ;X value after EXECUTE
          DFB   Y_Value   ;Y value after EXECUTE
          DFB   P_Value   ;Processor Status value after EXECUTE
```

The Error byte returned by a STATUS call with status code = $05 (Return UniDisk 3.5 Status) contains the following bits:

Bit	Description
7	0
6	0
5	1 = address field mark or checksum error
4	1 = data field checksum error
3	1 = data field bitslip mark mismatch
2	1 = seek error; unexpected track value found in address field
1	0
0	0

The Retries byte returned by a STATUS call with status code = $05 (Return UniDisk 3.5 Status) specfies the number of address fields that had to be passed before the operation was completed. This information could be used, for example, to determine the number of passes necessary to read a data field correctly: If Retries is found to be greater than the number of sectors on the target track, then more than one pass was required.

The last four bytes of the status list are set only after a CONTROL call with control code = $05, and are zero after any other call (STATUS calls do not clear the status bytes).

Possible Errors

The following errors can be returned by the STATUS call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$21	BADCTL	Invalid status code
$30-$3F		Device-specific errors

READ BLOCK

Command Number	$01
Parameter List	$03 (parameter count) Unit number Data buffer (low byte, high byte) Block number (low byte, mid byte, high byte)

The READ BLOCK call reads one 512-byte block from the disk device specified by the unit-number parameter into memory starting at the address specified by the data-buffer parameter.

Parameter Descriptions

Parameter Count
1-byte value

3 for this call.

Unit Number
1-byte value

The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01—$7E, and are assigned according to the devices' positions in the chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Data Buffer
2-byte value

Points to the buffer into which the data is read. The buffer must be 512 or more bytes in length.

Block Number
3-byte value

The logical address of a block of data to be read. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical blocks is performed by the device. (The most significant byte is zero for all devices currently in use.)

Possible Errors

The following errors can be returned by the READ BLOCK call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$27	IOERROR	I/O error
$28	NODRIVE	No device connected
$2D	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off-line or no disk in drive

WRITE BLOCK

Command Number	$02
Parameter List	$03 (parameter count) Unit number Data buffer (low byte, high byte) Block number (low byte, mid byte, high byte)

The WRITE BLOCK call writes one 512-byte block from memory to the disk device specified by the unit-number parameter. The block in memory starts at the address specified by the data-buffer parameter.

Parameter Descriptions

Parameter Count
1-byte value

3 for this call.

Unit Number
1-byte value

The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01—$7E, and are assigned according to the devices' positions in the chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Data Buffer
2-byte value

Points to the buffer from which the data is to be written.

Block Number
3-byte value

The logical address of a block of data to be written. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical blocks is performed by the device. (The most significant byte is zero for all devices currently in use.)

Possible Errors

The following errors can be returned by the WRITE BLOCK call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$27	IOERROR	I/O error
$28	NODRIVE	No device connected
$2B	NOWRITE	Disk write protected
$2D	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off-line or no disk in drive

FORMAT

Command Number	$03
Parameter List	$01 (parameter count) Unit number

The FORMAT call prepares all blocks on the recording medium of a block device for reading and writing. The formatting done by this call is not linked to any operating system; for example, bitmaps and catalogs are not written by this call.

Parameter Descriptions

Parameter Count
1-byte value

1 for this call.

Unit Number
1-byte value

The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01—$7E, and are assigned according to the devices' positions in the chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Possible Errors

The following errors can be returned by the FORMAT call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$27	IOERROR	I/O error
$28	NODRIVE	No device connected
$2B	NOWRITE	Disk write protected
$2F	OFFLINE	Device off-line or no disk in drive

CONTROL

Command Number	$04
Parameter List	$03 (parameter count) Unit number Control list (low byte, high byte) Control code

The CONTROL call sends control information to the device. The information can be of a general nature (such as resets or interrupts), or device-specific (such as Download to UniDisk 3.5 RAM).

Important A CONTROL call to unit number $00 sends control information to the Protocol Converter itself. See the discussions under Control Code = $00 and Control Code = $01, below.

Parameter Descriptions

Parameter Count
1-byte value

3 for this call.

Unit Number
1-byte value

The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01—$7E, and are assigned according to the devices' positions in the chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Protocol Converter. Use a unit number of $00 in the CONTROL call to send control information to the Protocol Converter itself.

Control List
2-byte value

Points to the buffer from which the control information is read. The first two bytes (the *count bytes*, low byte first) of the control list specify the number of bytes in the list (*not* including the count bytes); the remainder of the list contains the control information passed to the device.

Important Every CONTROL call must have a control list; if no control information is being passed, then the control list consists of the count bytes only:

```
CTRL_LIST DW $00
```

Control Code
1-byte value

The number of the control request being made. Control codes are in the range $00—$FF. The following requests are not device-specific:

Code	Control Function
$00	Reset the device
$01	Set device control block (DCB)
$02	Set newline status (character devices only)
$03	Service device interrupt

Control requests to unit number $00 are sent to the Protocol Converter itself:

Code	Control Function
$00	Enable interrupts from Protocol Converter
$01	Disable interrupts from Protocol Converter

Specific devices may respond to some or all of these additional control requests:

Code	Control Function
$04	Eject disk
$05	Run a 65C02 subroutine
$06	Set download address
$07	Download to device RAM

Control Code = $00, Reset the Device Performs a warm reset of the device. Generally returns "housekeeping" values to some reset value. The control list for this call is device dependent.

UniDisk 3.5 The control list for this call for UniDisk 3.5 devices is:

```
CTRL_LIST DW $00 ;No parameters are passed
```

Unit Number $00: A CONTROL call with control code = $00 and unit number = $00 enables interrupts from the Protocol Converter. This call informs the firmware that external interrupts are possible, and directs it to call the user's interrupt handler if an interrupt occurs. It also turns on the Asynchronous Communications Interface Adapter (ACIA) for port 1.

When the user's interrupt handler identifies an external interrupt, you can determine if it came from the Protocol Converter by making a STATUS call with unit number = $00 and control code = $00 (see Section "STATUS"). See Appendix E in the *Apple IIc Reference Manual* for more information on handling interrupts.

Control Code = $01, Set Device Control Block Alters the contents of the device control block (DCB). The DCB is usually used to set global aspects of a device's operating environment. Each device has a default setting for the DCB, set on initialization. Since the length of the DCB is device dependent, you should first read in the DCB with the STATUS call, then alter the bits of interest, and finally, use the same byte string as the control block for the CONTROL call. The first byte (the *count byte*) of the DCB gives the number of bytes in the control block (*not* including the count byte), so the length never exceeds 257 bytes, including the count byte.

UniDisk 3.5

UniDisk 3.5 has no DCB; a Set DCB CONTROL call to UniDisk 3.5 returns an error (BADCTL $21).

Unit Number = $00: A CONTROL call with control code = $01 and unit number = $00 disables interrupts from the Protocol Converter. This call turns off the ACIA for port 1 and sets the least significant bit of the ACIA control register to zero.

Newline read mode: See Chapter 4 in the *PRODOS Technical Reference Manual.*

Control Code = $02, Set Newline Status Sets a character device to newline enabled or newline disabled.

Control Code = $03, Device Service Interrupt To be used as needed for interrupt-driven devices.

Control Code = $04, Eject Disk To be used for devices that support an auto-eject feature.

UniDisk 3.5

Causes UniDisk 3.5 to auto-eject a disk. There are no parameters in the control list, and no errors are returned if the disk ejected correctly or there was no disk in the drive. Error code $27 (I/O error) is returned if the eject failed, that is, a disk is still in the drive. The control list for UniDisk 3.5 is:

```
CTRL_LIST DW $00 ;No parameters are passed
```

▲Warning | Control codes $05 and higher are reserved; use of some of these codes can cause your system to crash.

Possible Errors

The following errors can be returned by the CONTROL call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$21	BADCTL	Invalid control code
$22	BADCTLPARM	Invalid parameter list
$30-$3F		Device-specific errors

INIT

Command Number	$05
Parameter List	$01 (parameter count) $00 (unit number)

The INIT call resets all intelligent devices attached to the Protocol Converter. The Protocol Converter goes through an initialization sequence, cold-resetting all devices and sending each its unit number. This call is made automatically on startup; an application should never have to make this call.

Parameter Descriptions

Parameter Count
1-byte value
1 for this call.

Unit Number
1-byte value
The unit number used in this call is always $00.

Possible Errors

The following errors can be returned by the INIT call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$28	NODRIVE	No device connected

OPEN

Command Number	$06
Parameter List	$01 (parameter count) Unit number

The OPEN call prepares a character device for reading or writing.

UniDisk 3.5

Since UniDisk 3.5 is a block device, it does not accept this call. An attempt to use an OPEN call with UniDisk 3.5 will result in an error (BADCMD $01).

Parameter Descriptions

Parameter Count
1-byte value

1 for this call.

Unit Number
1-byte value

The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01—$7E, and are assigned according to the devices' positions in the chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Possible Errors

The following errors can be returned by the OPEN call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$28	NODRIVE	No device connected
$2F	OFFLINE	Device off-line or no disk in drive

CLOSE

Command Number	$07
Parameter List	$01 (parameter count) Unit number

The CLOSE call tells a character device that a sequence of reads or writes is over.

UniDisk 3.5

Since UniDisk 3.5 is a block device, it does not accept this call. An attempt to use a CLOSE call with UniDisk 3.5 will result in an error (BADCMD $01).

Parameter Descriptions

Parameter Count
1-byte value

1 for this call.

Unit Number
1-byte value

The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01—$7E, and are assigned according to the devices' positions in the chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Possible Errors

The following errors can be returned by the CLOSE call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$28	NODRIVE	No device connected
$2F	OFFLINE	Device off-line or no disk in drive

READ

Command Number	$08
Parameter List	$04 (parameter count) Unit number Buffer pointer (low byte, high byte) Byte count (low byte, high byte) Address pointer (low byte, mid byte, high byte)

The READ call reads the number of bytes specified by the byte-count parameter into memory starting at the address specified by the buffer-pointer parameter.

Macintosh: This call can be used by UniDisk 3.5 devices to read 524-byte data blocks written by an Apple Macintosh™ Computer.

Parameter Descriptions

Parameter Count
1-byte value

4 for this call.

Unit Number
1-byte value

The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01—$7E, and are assigned according to the devices' positions in the chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Buffer Pointer
2-byte value

Points to the buffer into which the data is read. The buffer must be large enough to contain the number of bytes requested by the byte-count parameter.

Byte Count
2-byte value

Specifies the number of bytes to be transferred.

Macintosh: The byte count used to read Macintosh disks with a UniDisk 3.5 is always 524 bytes ($020C).

Address Pointer
3-byte value

Specifies the address to start reading from. The meaning of this parameter depends on the device being read.

Macintosh: When using a UniDisk 3.5 to read Macintosh disks, the address pointer specifies the number of the 524-byte Macintosh block to be read (from $00 to $031F for a single-sided disk).

Possible Errors

The following errors can be returned by the READ call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$27	IOERROR	I/O error
$28	NODRIVE	No device connected
$2D	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off-line or no disk in drive

WRITE

Command Number	$09
Parameter List	$04 (parameter count) Unit number Buffer pointer (low byte, high byte) Byte count (low byte, high byte) Address pointer (low byte, mid byte, high byte)

The WRITE call writes the number of bytes specified by the byte-count parameter to the specified unit from memory starting at the address indicated by the buffer-pointer parameter. The meaning of the address pointer depends on the type of device (see the parameter descriptions, below).

Macintosh: This call can be used by UniDisk 3.5 devices to write 524-byte blocks for use by an Apple Macintosh computer.

Parameter Descriptions

Parameter Count 1-byte value	4 for this call.
Unit Number 1-byte value	The Protocol Converter assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01—$7E, and are assigned according to the devices' positions in the chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Protocol Converter.
Buffer Pointer 2-byte value	Points to the buffer from which the data is to be written.
Byte Count 2-byte value	Specifies the number of bytes to be transferred.

Macintosh: The byte count used to write Macintosh disks with a UniDisk 3.5 is always 524 bytes ($020C).

Address Pointer
3-byte value

Specifies the address to start writing from. The meaning of this parameter depends on the device being written to.

Macintosh: When using a UniDisk 3.5 to write Macintosh disks, the address pointer specifies the number of the 524-byte Macintosh block to be written (from $00 to $031F for a single-sided disk).

Possible Errors

The following errors can be returned by the WRITE call:

$01	BADCMD	An unimplemented command was issued
$04	BADPCNT	Bad call parameter count
$06	BUSERR	Communications error
$27	IOERROR	I/O error
$28	NODRIVE	No device connected
$2D	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off-line or no disk in drive

An Example: Issuing a Protocol Converter Call

Here is an example of a program that issues a STATUS call to the Protocol Converter to obtain information about a device.

Apple IIe

The code for the Protocol Converter in the Apple IIc with 32K ROM always begins at address $C500; however, to ensure compatibility with the Apple IIe, your programs should always do a search for the Protocol Converter, as in the following example.

```
0000:                    1 *
0000:                    2 *
0000:                    3 *
0000:                    4 * This example shows how to find
0000:                    5 *  and use a PC interface.  A search
0000:                    6 *  is made for a PC, and when one is
0000:                    7 *  found, a vector is set up which
0000:                    8 *  points to the PC entry.  Then a
0000:                    9 *  Device Information Block STATUS call
0000:                   10 *  is made, and if successful, the name
0000:                   11 *  string embedded in the DIB is output
0000:                   12 *  to the screen. Only the first device
0000:                   13 *  in the chain is accessed.
0000:                   14 *
0000:                   15 *
0000:                   16           MSB   ON
0000:                   17 *
0000:                   18 *
0000:          0006     19 ZPTempL   equ   $0006     ;Temporary zero
0000:                   20 *                          page storage
0000:          0007     21 ZPTempH   equ   $0007
0000:                   22 *
0000:          FDED     23 COut      equ   $FDED     ;Console output
0000:          FD8E     24 CROut     equ   $FD8E     ;Carriage return
0000:                   25 *
0000:          0000     26 StatusCmd equ   0
0000:                   27 *
0000:                   28 *
0300:          0300     29           org   $300
0300:                   30 *
0300:                   31 * Find a Protocol Converter in one of the
0300:                   32 *  slots.
0300:                   33 *
0300:20 43 03           34           jsr   FindPC
0303:B0 1C    0321      35           bcs   Error
0305:                   36 *
0305:                   37 * Now make the DIB call to the first guy
0305:                   38 *
```

```
0305:20 67 03        39            jsr   Dispatch
0308:00              40            dfb   StatusCmd
0309:6A 03           41            dw    DParms
030B:B0 14   0321    42            bcs   Error
030D:                43 *
030D:                44 * Got the DIB; now print the name string
030D:                45 *
030D:A2 00           46            ldx   #0
030F:        030F    47 morechars  equ   *
030F:BD 74 03        48            lda   DIBName,x
0312:09 80           49            ora   #$80      ;COut wants high
0314                 50 *                            Bit set
0314:                51 *
0314:20 ED FD        52            jsr   COut
0317:E8              53            inx
0318:EC 73 03        54            cpx   DIBNameLen
031B:90 F2   030F    55            blt   morechars
031D:                56 *
031D:20 8E FD        57            jsr   CROut     ;Finish it off
0320:                58 *                            with a return
0320:                59 *
0320:60              60            rts
0321:                61 *
0321:                62 *
0321:        0321    63 Error      equ   *
0321:                64 *
0321:                65 * There's either no PC around, or there
0321:                66 * was no Unit #1... give message
0321:                67 *
0321:A2 00           68            ldx   #0
0323:        0323    69 err1       equ   *
0323:BD 2F 03        70            lda   Message,x
0326:F0 06   032E    71            beq   errout
0328:20 ED FD        72            jsr   COut
032B:E8              73            inx
032C:D0 F5   0323    74            bne   err1
032E:                75 *
032E:        032E    76 errout     equ   *
032E:60              77            rts
032F:                78 *
032F:CE CF A0 D0     79 Message    asc   'NO PC OR NO DEVICE'
0341:8D 00           80            dfb   $8D,0
0343:                81 *
0343:                82 *
0343:        0343    83 FindPC     equ   *
0343:                84 *
0343:                85 * Search slot 7 to slot 1 looking for
0343:                86 *  signature bytes
0343:                87 *
0343:A2 07           88            ldx   #7        ;Do for seven
0345:                89 *                            slots
0345:A9 C7           90            lda   #$C7
```

```
0347:85 07          91            sta   ZPTempH
0349:A9 00          92            lda   #$00
034B:85 06          93            sta   ZPTempL
034D:               94 *
034D:       034D    95 newslot    equ   *
034D:A0 07          96            ldy   #7
034F:               97 *
034F:       034F    98 again      equ   *
034F:B1 06          99            lda   (ZPTempL),y
0351:D9 70 03      100            cmp   sigtab,y      ;One of four
0354:              101 *                             byte signature
0354:F0 07  035D   102            beq   maybe         ;Found one
0356:              103 *                             signature byte
0356:C6 07         104            dec   ZPTempH
0358:CA            105            dex
0359:D0 F2  034D   106            bne   newslot
035B:              107 *
035B:              108 * If we get here, it's because we couldn't
035B:              109 *  find a Protocol Converter.
035B:              110 * Exit with the carry set.
035B:              111 *
035B:38            112            sec
035C:60            113            rts
035D:              114 *
035D:              115 * If we get here, it means that one or
035D:              116 *  more of the signature bytes
035D:              117 *  for this card are what we're looking
035D:              118 *  for.  Decrement the byte
035D:              119 *  counter and branch back to verify any
035D:              120 *  remaining bytes.
035D:              121 *
035D:       035D   122 maybe      equ   *
035D:88            123            dey
035E:88            124            dey                 ;If N=1 then
035F:              125 *                          all sig bytes okay
035F:10 EE  034F   126            bpl   again
0361:              127 *
0361:              128 * Found a Protocol Converter interface.
0361:              129 *  Set up the call address.
0361:              130 * We already have the high byte ($CN);
0361:              131 *  we just need the low byte.
0361:              132 *
0361:       0361   133 foundPC    equ   *
0361:A9 FF         134            lda   #$FF
0363:85 06         135            sta   ZPTempL
0365:A0 00         136            ldy   #0            ;For
0367:              137 *                             indirect load
0367:B1 06         138            lda   (ZPTempL),y   ;Get the
0369:              139 *                               byte
0369:              140 *
0369:              141 * Now the Acc has the low order ProDOS
0369:              142 *  entry point.  The PC entry is
```

```
0369:                 143 *  three locations past this...
0369:                 144 *
0369:18               145            clc
036A:69 03            146            adc   #3
036C:85 06            147            sta   ZPTempL
036E:                 148 *
036E:                 149 * Now ZPTempL has the PC entry point.
036E:                 150 *  Return with carry clear.
036E:                 151 *
036E:18               152            clc
036F:60               153            rts
0370:                 154 *
0370:                 155 *
0370:                 156 * These are the PC signature bytes in
0370:                 157 *  their relative order.
0370:                 158 *  The $FF bytes are filler bytes and
0370:                 159 *  are not compared.
0370:                 160 *
0370:FF 20 FF 00      161 sigtab     dfb   $FF,$20,$FF,$00
0374:FF 03 FF 00      162            dfb   $FF,$03,$FF,$00
0378:                 163 *
0378:                 164 *
0378:         0378    165 Dispatch   equ   *
0378:6C 06 00         166            jmp   (ZPTempL)      ;Simulate
037B:                 167 *                    an indirect JSR to PC
037B:                 168 *
037B:                 169 *
037B:         037B    170 DParms     equ   *
037B:03               171 DPParmCt   dfb   3              ;Status
037C:                 172 *              calls have three parameters
037C:01               173 DPUnit     dfb   1
037D:80 03            174 DPBuffer   dw    DIB
037F:03               175 DPStatCode dfb   3
0380:                 176 *
0380:                 177 *
0380:         0380    178 DIB        equ   *
0380:00               179 DIBStatByte1 dfb 0
0381:00 00 00         180 DIBDevSize dfb   0,0,0
0384:00               181 DIBNameLen dfb   0
0385:         0010    182 DIBName    ds    16,0
0395:00               183 DIBType    dfb   0
0396:00               184 DIBSubType dfb   0
0397:00 00            185 DIBVersion dw    0
0399:                 186 *
0399:                 187 *
```

Summary of Commands and Parameters

This is a summary of Protocol Converter calls. In each case, byte 0 of the command parameter list (CMDLST) specifies the number of parameters in the command list (not including byte 0). Parameters that require more than one byte (the status list pointer, for example) are entered low byte first. The meaning of the address-pointer parameter is device specific. See the sections on the individual calls in this chapter for a discussion of each parameter.

Figure 3-1. Summary of Protocol Converter Commands and Parameters

Command	**STATUS**	**READBLOCK**	**WRITEBLOCK**	**FORMAT**	**CONTROL**
CmdNum	$00	$01	$02	$03	$04
CmdList Byte					
0	$03	$03	$03	$01	$03
1	Unit Num	Unit Num	Unit Num	Unit Num	Unit Num
2	Stat List Ptr	Buffer Ptr	Buffer Ptr		Ctl List Ptr
3					
4	Stat Code	Block Num	Block Num		Ctl Code
5					
6					

Command	**INIT**	**OPEN**	**CLOSE**	**READ**	**WRITE**
CmdNum	$05	$06	$07	$08	$09
CmdList Byte					
0	$01	$01	$01	$04	$04
1	$00	Unit Num	Unit Num	Unit Num	Unit Num
2				Buffer Ptr	Buffer Ptr
3					
4				Byte Count	Byte Count
5					
6					
7				Address Ptr	Address Ptr
8					

Unused bytes

Summary of Error Codes

This is a summary of Protocol Converter call error codes, including a brief description of the possible causes for each. If there is no error, the C flag (in the Processor Status register of the 65C02 microprocessor) is cleared (0), and the accumulator (the A register) contains zeros. If the call was unsuccessful, the C flag is set (1), and the A register contains the error code.

$00		No error.
$01	BADCMD	A nonexistent command was issued. Check the command number in the Protocol Converter call.
$04	BADPCNT	Bad call parameter count. The call parameter list was not properly constructed. Make sure the parameter list has the correct number of parameters.
$06	BUSERR	A communications error between the device controller and the host. Make sure that RAM is both read-enabled and write-enabled. Check the hardware (cables and connectors) between the device and the host. Check for noise sources; make sure the cable is properly shielded.
$11	BADUNIT	Unit number $00 was used in a call other than STATUS, CONTROL, or INIT.
$21	BADCTL	The control or status code is not supported by the device.
$22	BADCTLPARM	The control parameter list contains invalid information. Make sure each value is within the range allowed for that parameter.
$27	IOERROR	The device encountered an I/O error when trying to read or write to the recording medium. Make sure that the medium in the device is formatted and not defective. Make sure the device is operating correctly.
$28	NODRIVE	The device is not connected. This can occur if the device is not connected but its controller is, or if there is no device with the unit number specified.

$2B	NOWRITE	The medium in the device is write protected.
$2D	BADBLOCK	The block number is outside the range allowed for the medium in the device. Note that this range depends on the type of device and the type of medium in the device (single-sided vs. double-sided disk, for example).
$2F	OFFLINE	Device off-line or no disk in drive. Check the cables and connections; make sure the medium is present in the drive, and that the drive is functioning correctly.
$30-$3F	DEVSPEC	Errors which differ from device to device. See the technical manual for the device in question for details.
$40-$4F		Reserved for future expansion.
$50-$7F	NONFATAL	A device-specific *soft* error. The operation completed successfully, but some *exception* condition was detected. See the technical manual for the device in question for details.

Appendix A

Firmware Listing

```
SOURCE   FILE #01 =>FIRM
 INCLUDE FILE #02 =>NAMES
 INCLUDE FILE #03 =>EQUATES
 INCLUDE FILE #04 =>SERIAL
 INCLUDE FILE #05 =>SER
 INCLUDE FILE #06 =>COMM
 INCLUDE FILE #07 =>C3SPACE
 INCLUDE FILE #08 =>MOUSE
 INCLUDE FILE #09 =>MCODE
 INCLUDE FILE #10 =>MISC
 INCLUDE FILE #11 =>BOOT
 INCLUDE FILE #12 =>SWITCHER
 INCLUDE FILE #13 =>IRQBUF
 INCLUDE FILE #14 =>MINI
 INCLUDE FILE #15 =>SCROLLING
 INCLUDE FILE #16 =>ESCAPE
 INCLUDE FILE #17 =>PASCAL
 INCLUDE FILE #18 =>MOREMISC
 INCLUDE FILE #19 =>AUTOST1
 INCLUDE FILE #20 =>AUTOST2
 INCLUDE FILE #21 =>BANK2
 INCLUDE FILE #22 =>MINT
 INCLUDE FILE #23 =>AUXSTUFF
 INCLUDE FILE #24 =>BANGER2
 INCLUDE FILE #25 =>SWITCHER2
 INCLUDE FILE #26 =>COMMAND
 INCLUDE FILE #27 =>MBASIC
 INCLUDE FILE #28 =>BANGER
 INCLUDE FILE #29 =>VECTORS2
```

```
0000:          0000     2           x6502
0000:                   3 ****************************************
0000:                   4 *
0000:                   5 * Firmware for the Apple //c
0000:                   6 *
0000:                   7 * December, 1983
0000:                   8 *
0000:                   9 *
0000:                  10 * Rich Williams
0000:                  11 * Ernie Beernink
0000:                  12 * James R Huston
0000:                  13 *
0000:                  14 * Revision 2 May, 1985
0000:                  15 * rom expanded to 32K in 2 16K banks
0000:                  16 * new features added:
0000:                  17 *   Protocol converter slot 5
0000:                  18 *   AppleTalk slot 7
0000:                  19 *   //e diagnostics
0000:                  20 *   Enhanced serial port commands
0000:                  21 *   Mini assembler
0000:                  22 *   Step and trace
0000:                  23 * most $F8 rom changes marked with a +
0000:                  24 *
0000:                  25 ****************************************
0000:          F800    26 F8ORG      EQU   $F800

 INCLUDE FILE #02 =>NAMES
C100:                  29           lst   on
C100:                  30           include equates      ;Equates for Video & Monitor ROM
```

```
C100:                  2 ********************************
C100:                  3 *
C100:                  4 * Apple //c
C100:                  5 * Video Firmware and
C100:                  6 * Monitor ROM Source
C100:                  7 *
C100:                  8 * COPYRIGHT 1977-1983 BY
C100:                  9 * APPLE COMPUTER, INC.
C100:                 10 *
C100:                 11 * ALL RIGHTS RESERVED
C100:                 12 *
C100:                 13 * S. WOZNIAK            1977
C100:                 14 * A. BAUM               1977
C100:                 15 * JOHN A           NOV 1978
C100:                 16 * R. AURICCHIO     SEP 1981
C100:                 17 * E. BEERNINK          1983
C100:                 18 *
C100:                 19 ********************************
C100:                 20 *
C100:                 21 * ZERO PAGE EQUATES
C100:                 22 *
C100:  0000    23 LOC0      EQU   $00          ;vector for autostart from disk
C100:  0001    24 LOC1      EQU   $01
C100:  0020    25 WNDLFT    EQU   $20          ;left edge of text window
C100:  0021    26 WNDWDTH   EQU   $21          ;width of text window
C100:  0022    27 WNDTOP    EQU   $22          ;top of text window
C100:  0023    28 WNDBTM    EQU   $23          ;bottom+1 of text window
C100:  0024    29 CH        EQU   $24          ;cursor horizontal position
C100:  0025    30 CV        EQU   $25          ;cursor vertical position
C100:  0026    31 GBASL     EQU   $26          ;lo-res graphics base addr.
C100:  0027    32 GBASH     EQU   $27
C100:  0028    33 BASL      EQU   $28          ;text base address
C100:  0029    34 BASH      EQU   $29
C100:  002A    35 BAS2L     EQU   $2A          ;temp base for scrolling
C100:  002B    36 BAS2H     EQU   $2B
C100:  002C    37 H2        EQU   $2C          ;temp for lo-res graphics
C100:  002C    38 LMNEM     EQU   $2C          ;temp for mnemonic decoding
C100:  002C    39 RTNL      equ   $2C          ;Step return address
C100:  002D    40 V2        EQU   $2D          ;temp for lo-res graphics
C100:  002D    41 RMNEM     EQU   $2D          ;temp for mnemonic decoding
C100:  002D    42 rtnh      equ   $2D          ;Step return address
C100:  002E    43 MASK      EQU   $2E          ;color mask for lo-res gr.
C100:  002E    44 FORMAT    EQU   $2E          ;temp for opcode decode
C100:  002F    45 LENGTH    EQU   $2F          ;temp for opcode decode
C100:  0030    46 COLOR     EQU   $30          ;color for lo-res graphics
C100:  0031    47 MODE      EQU   $31          ;Monitor mode
C100:  0032    48 INVFLG    EQU   $32          ;normal/inverse(/flash)
C100:  0033    49 PROMPT    EQU   $33          ;prompt character
C100:  0034    50 YSAV      EQU   $34          ;position in Monitor command
C100:  0035    51 YSAV1     EQU   $35          ;temp for Y register
C100:  0036    52 CSWL      EQU   $36          ;character output hook
C100:  0037    53 CSWH      EQU   $37
C100:  0038    54 KSWL      EQU   $38          ;character input hook
C100:  0039    55 KSWH      EQU   $39
C100:  003A    56 PCL       EQU   $3A          ;temp for program counter
C100:  003B    57 PCH       EQU   $3B
C100:  003C    58 XQT       EQU   $3C          ;Step and trace execute area
C100:  003C    59 A1L       EQU   $3C          ;Monitor temp
```

```
C100:          003D   60 A1H        EQU   $3D          ;Monitor temp
C100:          003E   61 A2L        EQU   $3E          ;Monitor temp
C100:          003F   62 A2H        EQU   $3F          ;Monitor temp
C100:          0040   63 A3L        EQU   $40          ;Monitor temp
C100:          0041   64 A3H        EQU   $41          ;Monitor temp
C100:          0042   65 A4L        EQU   $42          ;Monitor temp
C100:          0043   66 A4H        EQU   $43          ;Monitor temp
C100:          0044   67 A5L        EQU   $44          ;Monitor temp
C100:          0045   68 A5H        EQU   $45          ;Monitor temp
C100:                 69 *
C100:                 70 * Note: In Apple II, //e, both interrupts and BRK destroyed
C100:                 71 * location $45.  Now only BRK destroys $45 (ACC) and it
C100:                 72 * also destroys $44 (MACSTAT).
C100:                 73 *
C100:          0044   74 MACSTAT    EQU   $44          ;Machine state after BRK
C100:          0045   75 ACC        EQU   $45          ;Acc after BRK
C100:                 76 *
C100:          0046   77 XREG       EQU   $46          ;X reg after break
C100:          0047   78 YREG       EQU   $47          ;Y reg after break
C100:          0048   79 STATUS     EQU   $48          ;P reg after break
C100:          0049   80 SPNT       EQU   $49          ;SP after break
C100:          004E   81 RNDL       EQU   $4E          ;random counter low
C100:          004F   82 RNDH       EQU   $4F          ;random counter high
C100:                 83 *
C100:                 84 * Value equates
C100:                 85 *
C100:          0006   86 GOODF8     EQU   $06          ;value of //e, lolly ID byte
C100:          0095   87 PICK       EQU   $95          ;CONTROL-U character
C100:          009B   88 ESC        EQU   $9B          ;what ESC generates
C100:                 89 *
C100:                 90 * Characters read by GETLN are placed in
C100:                 91 * IN, terminated by a carriage return.
C100:                 92 *
C100:          0200   93 IN         EQU   $0200        ;input buffer for GETLN
C100:                 94 *
C100:                 95 * Page 3 vectors
C100:                 96 *
C100:          03F0   97 BRKV       EQU   $03F0        ;vectors here after break
C100:          03F2   98 SOFTEV     EQU   $03F2        ;vector for warm start
C100:          03F4   99 PWREDUP    EQU   $03F4        ;THIS MUST = EOR #$A5 OF SOFTEV+1
C100:          03F5  100 AMPERV     EQU   $03F5        ;APPLESOFT & EXIT VECTOR
C100:          03F8  101 USRADR     EQU   $03F8        ;APPLESOFT USR function vector
C100:          03FB  102 NMI        EQU   $03FB        ;NMI vector
C100:          03FE  103 IRQLOC     EQU   $03FE        ;Maskable interrupt vector
C100:          0400  104 LINE1      EQU   $0400        ;first line of text screen
C100:          07F8  105 MSLOT      EQU   $07F8        ;owner of $C8 space
C100:                106 *
C100:                107 * HARDWARE EQUATES
C100:                108 *
C100:          C000  109 IOADR      EQU   $C000        ;for IN#, PR# vector
C100:          C000  110 KBD        EQU   $C000        ;>127 if keystroke
C100:          C000  111 CLR80COL   EQU   $C000        ;disable 80 column store
C100:          C001  112 SET80COL   EQU   $C001        ;enable 80 column store
C100:          C002  113 RDMAINRAM  EQU   $C002        ;read from main 48K RAM
C100:          C003  114 RDCARDRAM  EQU   $C003        ;read from alt. 48K RAM
C100:          C004  115 WRMAINRAM  EQU   $C004        ;write to main 48K RAM
C100:          C005  116 WRCARDRAM  EQU   $C005        ;write to alt. 48K RAM
C100:          C008  117 SETSTDZP   EQU   $C008        ;use main zero page/stack
```

```
C100:          C009  118 SETALTZP   EQU   $C009          ;use alt. zero page/stack
C100:          C00C  119 CLR80VID   EQU   $C00C          ;disable 80 column hardware
C100:          C00D  120 SET80VID   EQU   $C00D          ;enable 80 column hardware
C100:          C00E  121 CLRALTCHAR EQU   $C00E          ;normal LC, flashing UC
C100:          C00F  122 SETALTCHAR EQU   $C00F          ;normal inverse, LC; no flash
C100:          C010  123 KBDSTRB    EQU   $C010          ;turn off key pressed flag
C100:          C011  124 RDLCBNK2   EQU   $C011          ;>127 if LC bank 2 is in
C100:          C012  125 RDLCRAM    EQU   $C012          ;>127 if LC RAM read enabled
C100:          C013  126 RDRAMRD    EQU   $C013          ;>127 if reading main 48K
C100:          C014  127 RDRAMWRT   EQU   $C014          ;>127 if writing main 48K
C100:          C016  128 RDALTZP    EQU   $C016          ;>127 if Alt ZP and LC switched in
C100:          C018  129 RD80COL    EQU   $C018          ;>127 if 80 column store
C100:          C019  130 RDVBLBAR   EQU   $C019          ;>127 if not VBL
C100:          C01A  131 RDTEXT     EQU   $C01A          ;>127 if text (not graphics)
C100:          C01B  132 RDMIX      EQU   $C01B          ;>127 if mixed mode on
C100:          C01C  133 RDPAGE2    EQU   $C01C          ;>127 if TXTPAGE2 switched in
C100:          C01D  134 RDHIRES    EQU   $C01D          ;>127 if HIRES is on
C100:          C01E  135 ALTCHARSET EQU   $C01E          ;>127 if alternate char set in use
C100:          C01F  136 RD80VID    EQU   $C01F          ;>127 if 80 column hardware in
C100:          C028  137 ROMBANK    EQU   $C028          ;Switches rombanks
C100:          C030  138 SPKR       EQU   $C030          ;clicks the speaker
C100:          C050  139 TXTCLR     EQU   $C050          ;switch in graphics (not text)
C100:          C051  140 TXTSET     EQU   $C051          ;switch in text (not graphics)
C100:          C052  141 MIXCLR     EQU   $C052          ;clear mixed-mode
C100:          C053  142 MIXSET     EQU   $C053          ;set mixed-mode (4 lines text)
C100:          C054  143 TXTPAGE1   EQU   $C054          ;switch in text page 1
C100:          C055  144 TXTPAGE2   EQU   $C055          ;switch in text page 2
C100:          C056  145 LORES      EQU   $C056          ;low-resolution graphics
C100:          C057  146 HIRES      EQU   $C057          ;high-resolution graphics
C100:          C058  147 CLRAN0     EQU   $C058
C100:          C059  148 SETAN0     EQU   $C059
C100:          C05A  149 CLRAN1     EQU   $C05A
C100:          C05B  150 SETAN1     EQU   $C05B
C100:          C05C  151 CLRAN2     EQU   $C05C
C100:          C05D  152 SETAN2     EQU   $C05D
C100:          C05E  153 CLRAN3     EQU   $C05E
C100:          C05F  154 SETAN3     EQU   $C05F
C100:          C060  155 RD40SW     EQU   $C060          ;>127 if 40/80 switch in 40 pos
C100:          C061  156 BUTN0      EQU   $C061          ;open apple key
C100:          C062  157 BUTN1      EQU   $C062          ;closed apple key
C100:          C064  158 PADDL0     EQU   $C064          ;read paddle 0
C100:          C070  159 PTRIG      EQU   $C070          ;trigger the paddles
C100:          C081  160 ROMIN      EQU   $C081          ;switch in $D000-$FFFF ROM
C100:          C083  161 LCBANK2    EQU   $C083          ;switch in LC bank 2
C100:          C08B  162 LCBANK1    EQU   $C08B          ;switch in LC bank 1
C100:          CFFF  163 CLRROM     EQU   $CFFF          ;switch out $C8 ROMs
C100:          E000  164 BASIC      EQU   $E000          ;BASIC entry point
C100:          E003  165 BASIC2     EQU   $E003          ;BASIC warm entry point
C100:                166 *
C100:          04FB  167 VMODE      EQU   $4F8+3         ;OPERATING MODE
C100:                168 *
C100:                169 * BASIC VMODE BITS
C100:                170 *
C100:                171 * 1....... - BASIC active
C100:                172 * 0....... - Pascal active
C100:                173 * .0......
C100:                174 * .1......
C100:                175 * ..0..... - Print control characters
```

```
C100:                  176 * ..1..... - Don't print ctrl chars
C100:                  177 * ...0.... -
C100:                  178 * ...1.... -
C100:                  179 * ....0... - Print control characters
C100:                  180 * ....1... - Don't print ctrl chars.
C100:                  181 * .....0.. -
C100:                  182 * .....1.. -
C100:                  183 * ......0. -
C100:                  184 * ......1. -
C100:                  185 * .......0 - Print mouse characters
C100:                  186 * .......1 - Don't print mouse characters
C100:                  187 *
C100:          0040    188 M.40        EQU   $40
C100:          0020    189 M.CTL2      EQU   $20           ;Don't print controls
C100:          0008    190 M.CTL       EQU   $08           ;Don't print controls
C100:          0001    191 M.MOUSE     EQU   $01           ;Don't print mouse chars
C100:                  192 *
C100:                  193 * Pascal Mode Bits
C100:                  194 *
C100:                  195 * 1....... - BASIC active
C100:                  196 * 0....... - Pascal active
C100:                  197 * .0......
C100:                  198 * .1......
C100:                  199 * ..0..... -
C100:                  200 * ..1..... -
C100:                  201 * ...0.... - Cursor always on
C100:                  202 * ...1.... - Cursor always off
C100:                  203 * ....0... - GOTOXY n/a
C100:                  204 * ....1... - GOTOXY in progress
C100:                  205 * .....0.. - Normal Video
C100:                  206 * .....1.. - Inverse Video
C100:                  207 * ......0. -
C100:                  208 * ......1. -
C100:                  209 * .......0 - Print mouse chars
C100:                  210 * .......1 - Don't print mouse chars
C100:                  211 *
C100:          0080    212 M.PASCAL    EQU   $80           ;Pascal active
C100:          0010    213 M.CURSOR    EQU   $10           ;Don't print cursor
C100:          0008    214 M.GOXY      EQU   $08           ;GOTOXY IN PROGRESS
C100:          0004    215 M.VMODE     EQU   $04
C100:                  216 *
C100:          0478    217 ROMSTATE    EQU   $478          ;temp store of ROM state
C100:          04F8    218 TEMP1       EQU   $4F8          ;used by CTLCHAR
C100:          0578    219 TEMPA       EQU   $578          ;used by scroll
C100:          05F8    220 TEMPY       EQU   $5F8          ;used by scroll
C100:                  221 *
C100:          047B    222 OLDCH       EQU   $478+3        ;last value of CH
C100:          057B    223 OURCH       EQU   $578+3        ;80-COL CH
C100:          05FB    224 OURCV       EQU   $5F8+3        ;CURSOR VERTICAL
C100:          067B    225 VFACTV      EQU   $678+3        ;Bit7=video firmware inactive
C100:          06FB    226 XCOORD      EQU   $6F8+3        ;X-COORD (GOTOXY)
C100:          077B    227 NXTCUR      EQU   $778+3        ;next cursor to display
C100:          07FB    228 CURSOR      EQU   $7F8+3        ;the current cursor char
C100:                  229 *
C100:                  230 * Disk II boot rom equates
C100:                  231 *
C100:          0356    232 DNIBL       EQU   $356
C100:          0300    233 NBUF1       EQU   $300
```

```
C100:          002B  234 SLOTZ     EQU   $2B
C100:          003C  235 BOOTTMP   EQU   $3C
C100:          004F  236 BOOTDEV   EQU   $4F


C100:                238 ****************************************
C100:                239 * Entry points for other modules
C100:                240 ****************************************
C100:          C880  241 pcnv      equ   $C880
C100:          C5F5  242 bootfail  equ   $C5F5         ;Boot fails message
C100:          C5F8  243 pcnvrst   equ   $C5F8         ;Protocol converter reset
C100:          C580  244 atalk     equ   $C580         ;Apple talk
C100:                 31           include serial      ;Equates for serial code
```

```
C100:                3 ****************************************
C100:                4 *
C100:                5 * Apple Lolly communications driver
C100:                6 *
C100:                7 * By
C100:                8 * Rich Williams
C100:                9 * August 1983
C100:               10 * November 5 - j.r.huston
C100:               11 *
C100:               12 ****************************************
C100:               13 *
C100:               14 * Command codes
C100:               15 *
C100:               16 * Default command char is ctrl-A (^A)
C100:               17 *
C100:               18 *      ^AnnB: Set baud rate to nn
C100:               19 *      ^AnnD: Set data format bits to nn
C100:               20 *      ^AI:   Enable video echo
C100:               21 *      ^AK:   Disable CRLF
C100:               22 *      ^AL:   Enable CRLF
C100:               23 *      ^AnnN: Disable video echo & set printer width
C100:               24 *      ^AnnP: Set parity bits to nn
C100:               25 *      ^AQ    Quit terminal mode
C100:               26 *      ^AR    Reset the ACIA, IN#0 PR#0
C100:               27 *      ^AS    Send a 233 ms break character
C100:               28 *      ^AT    Enter terminal mode
C100:               29 *      ^AZ:   Zap control commands
C100:               30 *      ^Ax:   Set command char to ^x
C100:               31 *      ^AnnCR:Set printer width (CR = carriage return)
C100:               32 *
C100:               33 * New commands added in rev 1 E = enable D = Disable
C100:               34 *
C100:               35 *      ^AC E/D Column overflow
C100:               36 *      ^AL E/D Linefeed same as L & K
C100:               37 *      ^AM E/D Mask incoming linefeeds
C100:               38 *      ^AX E/D Xon Xoff handshaking
C100:               39 *      ^AF E/D Find keyboard
C100:               40 *
C100:               41 ****************************************
C100:    C100       42 serslot   equ   $C100
C100:    C200       43 comslot   equ   $C200
C100:               44           msb   ON
C100:    00BF       45 cmdcur    equ   '?'          ;Cursor while in command mode
C100:    00DF       46 termcur   equ   '_'          ;Cursor while in terminal mode
C100:               47           msb   OFF
C100:    008A       48 lfeed     equ   $8A          ;Linefeed
C100:    0091       49 xon       equ   $91          ;XON character
C100:    0093       50 xoff      equ   $93          ;XOFF character
C100:    03B8       51 sermode   equ   $3B8         ;D7=1 if in command D6=1 if terminal
                                                     $479 & $47A
C100:    0438       52 astat     equ   $438         ;Acia status from int 4F9 & 4FA
C100:    04B8       53 pwdth     equ   $4B8         ;Printer width 579 & 57A
C100:    0538       54 extint    equ   $538         ;extint & typhed enable 5F9 & 5FA
C100:    05F9       55 extint2   equ   $5F9
C100:    05FA       56 typhed    equ   $5FA
C100:    0679       57 oldcur    equ   $679         ;Saves cursor while in command
C100:    067A       58 oldcur2   equ   $67A         ;Saves cursor while in terminal mode
C100:    0638       59 eschar    equ   $638         ;Current escape character 6F9 & 6FA
C100:    06B8       60 flags     equ   $6B8         ;D7 = Video echo D6 = CRLF 779 & 77A
```

```
C100:           0738   61 col       equ   $738          ;Current printer column 7F9 & 7FA
C100:           047E   62 number    equ   $47E          ;Number accumulated in command
C100:           04FF   63 aciabuf   equ   $4FF          ;Owner of serial buffer
C100:           057F   64 twser     equ   $57F          ;Storage pointer for serial buffer
C100:           05FF   65 twkey     equ   $5FF          ;Storage pointer for type ahead buffer
C100:           067F   66 trser     equ   $67F          ;Retrieve pointer for serial buffer
C100:           06FF   67 trkey     equ   $6FF          ;Retrieve buffer for type ahead buffer
C100:           0800   68 thbuf     equ   $800          ;Buffer in alt ram space
C100:           06F8   69 temp      equ   $6F8          ;Temp storage
C100:           05FE   70 charbuf   equ   $5FE          ;5FE, 67E are one byte character buffers
C100:           BFF8   71 sdata     equ   $BFF8         ;+$N0+$90 is output port
C100:           BFF9   72 sstat     equ   $BFF9         ;ACIA status register
C100:           BFFA   73 scomd     equ   $BFFA         ;ACIA command register
C100:           BFFB   74 scntl     equ   $BFFB         ;ACIA control register
C100:                  32           include ser         ;Printer port @ $C100
```

```
C100:                    3 *org serslot
C100:2C 89 C1            4            bit   serrts       ;Set V to indicate initial entry
C103:70 0C   C111        5            bvs   entr1        ;Always taken
C105:38                  6            sec                 ;Input entry point
C106:90                  7            dfb   $90          ;BCC opcode
C107:18                  8            clc
C108:B8                  9            clv                 ;V = 0 since not initial entry
C109:50 06   C111       10            bvc   entr1        ;Always taken

C10B:01                 12            dfb   $01          ;pascal signiture byte
C10C:31                 13            dfb   $31          ;device signiture
C10D:9E                 14            dfb   >p1init
C10E:A8                 15            dfb   >p1read
C10F:B4                 16            dfb   >p1write
C110:BB                 17            dfb   >p1status

C111:DA                 19 entr1      phx                 ;Save the reg
C112:A2 C1              20            ldx   #<serslot    ;X = Cn
C114:4C 1C C2           21            jmp   setup        ;Set mslot, etc
C117:90 03   C11C       22 serport    bcc   serisout     ;Only output allowed
C119:4C E5 C7           23            jmp   swzznm       ;Reset the hooks
C11C:0A                 24 serisout   asl   A            ;A = flags
C11D:7A                 25            ply                 ;Get char
C11E:5A                 26            phy
C11F:BD B8 04           27            lda   pwdth,x      ;Formatting enabled?
C122:F0 42   C166       28            beq   prnow
C124:A5 24              29            lda   ch           ;Get current horiz position
C126:B0 1C   C144       30            bcs   servid       ;Branch if video echo
C128:DD B8 04           31            cmp   pwdth,x      ;If CH >= PWIDTH, then CH = COL
C12B:90 03   C130       32            bcc   chok
C12D:BD 38 07           33            lda   col,x
C130:DD 38 07           34 chok       cmp   col,x        ;Must be > col for valid tab
C133:B0 0B   C140       35            bcs   fixch        ;Branch if ok
C135:C9 11              36            cmp   #$11         ;8 or 16?
C137:B0 11   C14A       37            bcs   prnt         ;If > forget it
C139:09 F0              38            ora   #$F0         ;Find next comma cheaply
C13B:3D 38 07           39            and   col,x        ;Don't blame me it's Dick's trick
C13E:65 24              40            adc   ch
C140:85 24              41 fixch      sta   ch           ;Save the new position
C142:80 06   C14A       42            bra   prnt
C144:C5 21              43 servid     cmp   wndwdth      ;If ch>= wndwdth go back to start of
                                                          line
C146:90 02   C14A       44            blt   prnt
C148:64 24              45            stz   ch           ;Go back to left edge


C14A:                   47 * We have a char to print
C14A:7A                 48 prnt       ply
C14B:5A                 49            phy
C14C:BD 38 07           50            lda   col,x        ;Have we exceeded width?
C14F:DD B8 04           51            cmp   pwdth,x
C152:B0 08   C15C       52            bge   toofar
C154:C5 24              53            cmp   ch           ;Are we tabbing?
C156:B0 0E   C166       54            bge   prnow
C158:A9 40              55            lda   #$40         ;Space * 2
C15A:80 02   C15E       56            bra   tab
C15C:A9 1A              57 toofar     lda   #$1A         ;CR * 2
C15E:C0 80              58 tab        cpy   #$80         ;C = High bit
```

```
C160:6A              59            ror   A             ;Shift it into char
C161:20 9B C1        60            jsr   goser3        ;Out it goes
C164:80 E4   C14A    61            bra   prnt
C166:98              62 prnow      tya
C167:20 8A C1        63            jsr   serout        ;Print the actual char
C16A:BD B8 04        64            lda   pwdth,x       ;Formatting enabled
C16D:F0 17   C186    65            beq   done
C16F:3C B8 06        66            bit   flags,x       ;In video echo?
C172:30 12   C186    67            bmi   done
C174:BD 38 07        68            lda   col,x         ;Check if within 8 chars of right edge
C177:FD B8 04        69            sbc   pwdth,x       ;So BASIC can format output
C17A:C9 F8           70            cmp   #$F8
C17C:90 04   C182    71            bcc   setch         ;If not within 8, we're done
C17E:18              72            clc
C17F:65 21           73            adc   wndwdth
C181:AC              74            dfb   $AC           ;Dummy LDY to skip next two bytes
C182:A9 00           75 setch      lda   #0            ;Keep cursor at 0 if video off
C184:85 24           76            sta   ch
C186:68              77 done       pla                  ;Restore regs
C187:7A              78            ply
C188:FA              79            plx
C189:60              80 serrts     rts


C18A:        C18A    82 serout     equ   *             ;Serial output
C18A:20 A9 C7        83            jsr   swcmd         ;Check if command
C18D:90 FA   C189    84            bcc   serrts        ;All done if it is
C18F:        C18F    85 serout2    equ   *
C18F:3C B8 06        86            bit   flags,x       ;N=1 iff video on
C192:10 07   C19B    87            bpl   goser3
C194:C9 91           88            cmp   #xon          ;Don't echo ^Q
C196:F0 03   C19B    89            beq   goser3
C198:20 F0 FD        90            jsr   cout1         ;Echo it
C19B:4C CD C7        91 goser3     jmp   swser3        ;Go to serout3


C19E:                93 * Pascal support stuff
C19E:5A              94 p1init     phy
C19F:48              95            pha
C1A0:20 B6 C2        96            jsr   default       ;set defaults, enable acia
C1A3:9E B8 06        97            stz   flags,x
C1A6:80 07   C1AF    98            bra   p1read2       ;all done...

C1A8:5A             100 p1read     phy
C1A9:20 D9 C7       101            jsr   swread        ;read data from serial port (or buffer)
C1AC:90 FA   C1A8   102            bcc   p1read        ;Branch if data not ready
C1AE:90             103            dfb   $90           ;BCC to skip pla
C1AF:68             104 p1read2    pla
C1B0:7A             105            ply
C1B1:A2 00          106            ldx   #0
C1B3:60             107            rts

C1B4:5A             109 p1write    phy
C1B5:48             110            pha
C1B6:20 8A C1       111            jsr   serout        ;Go output character
C1B9:80 F4   C1AF   112            bra   p1read2
```

```
C1BB:5A            113 p1status  phy
C1BC:48            114           pha
C1BD:4A            115           lsr   A             ;C = 0 output, 1 input
C1BE:D0 15   C1D5  116           bne   p1err           ;Branch if bad call
C1C0:08            117           php
C1C1:20 D3 C7      118           jsr   swgetst         ;Get status in A
C1C4:28            119           plp
C1C5:90 05   C1CC  120           bcc   p1stwr
C1C7:29 28         121           and   #$28            ;Test DCD = 0 & rcvr full
C1C9:0A            122           asl   A               ;$08 -> $10
C1CA:80 02   C1CE  123           bra   p1strd
C1CC:29 30         124 p1stwr    and   #$30            ;Test DCD = 0 & xmit empty
C1CE:C9 10         125 p1strd    cmp   #$10            ;Is it what we want?
C1D0:F0 DD   C1AF  126           beq   p1read2         ;C = 1 if equal
C1D2:18            127           clc                    ;Not ready
C1D3:80 DA   C1AF  128           bra   p1read2
C1D5:A2 40         129 p1err     ldx   #$40            ;Bad call
C1D7:68            130           pla
C1D8:7A            131           ply
C1D9:18            132           clc
C1DA:60            133           rts

C1DB:        0025  135           ds    comslot-*,$00
C200:               33           include comm          ;Communications port @ $C200
```

```
C200:2C 89 C1     3           bit   serrts       ;Set V to indicate initial entry
C203:70 14  C219  4           bvs   entr
C205:38           5 sin       sec                 ;Input entry point
C206:90           6           dfb   $90          ;BCC opcode to skip next byte
C207:18           7 sout      clc                 ;Output entry point
C208:B8           8           clv                 ;Mark not initial entry
C209:50 0E  C219  9           bvc   entr          ;Branch around pascal entry stuff

C20B:01           11          dfb   $01          ;pascal signiture byte
C20C:31           12          dfb   $31          ;device signiture
C20D:11           13          dfb   >p2init
C20E:13           14          dfb   >p2read
C20F:15           15          dfb   >p2write
C210:17           16          dfb   >p2status

C211:             18 * Pascal support stuff

C211:80 8B  C19E  20 p2init   bra   p1init
C213:80 93  C1A8  21 p2read   bra   p1read
C215:80 9D  C1B4  22 p2write  bra   p1write
C217:80 A2  C1BB  23 p2status bra   p1status

C219:DA           25 entr     phx
C21A:A2 C2        26          ldx   #<comslot     ;X = <CN00
C21C:       C21C  27 setup    equ   *
C21C:5A           28          phy
C21D:48           29          pha
C21E:8E F8 07     30          stx   mslot
C221:50 22  C245  31          bvc   sudone        ;First call?
C223:A5 36        32          lda   cswl          ;If both hooks CN00 setup defaults
C225:45 38        33          eor   kswl
C227:F0 06  C22F  34          beq   sudodef
C229:A5 37        35          lda   cswh          ;If both hooks CN then don't do def
C22B:C5 39        36          cmp   kswh          ;since it has already been done
C22D:F0 03  C232  37          beq   sunodef
C22F:20 B6 C2     38 sudodef  jsr   default       ;Set up defaults
C232:8A           39 sunodef  txa
C233:45 39        40          eor   kswh          ;Input call?
C235:05 38        41          ora   kswl
C237:D0 07  C240  42          bne   suout         ;Must be Cn00
C239:A9 05        43          lda   #>sin         ;Fix the input hook
C23B:85 38        44          sta   kswl
C23D:38           45          sec                 ;C = 1 for input call
C23E:80 05  C245  46          bra   sudone
C240:A9 07        47 suout    lda   #>sout        ;Fix output hook
C242:85 36        48          sta   cswl          ;Note C might not be 0
C244:13           49          clc                 ;C=0 for output
C245:BD B8 06     50 sudone   lda   flags,x       ;Check if serial or comm port
C248:89 01        51          bit   #1            ;Leave flags in a for serport
C24A:D0 03  C24F  52          bne   commport
C24C:4C 17 C1     53 comout   jmp   serport
C24F:90 FB  C24C  54 commport bcc   comout        ;Output?
C251:68           55          pla                 ;Get the char
C252:80 28  C27C  56          bra   term1         ;Input
C254:3C B8 03     57 noesc    bit   sermode,x     ;In terminal mode?
C257:50 1C  C275  58          bvc   exit1         ;If not, return key
C259:20 8F C1     59          jsr   serout2       ;Out it goes
C25C:80 1E  C27C  60          bra   term1
```

```
C25E:          C25E   61 testkbd   equ   *
C25E:68               62           pla                ;Get current char
C25F:20 70 CC         63           jsr   update       ;Update cursor & check keyboard
C262:10 1B    C27F    64           bpl   serin        ;N=0 if no new key
C264:20 A9 C7         65           jsr   swcmd        ;Test for command
C267:B0 EB    C254    66           bcs   noesc        ;Branch if not
C269:29 5F            67           and   #$5f         ;upshift for following tests
C26B:C9 51            68           cmp   #'Q'         ;Quit?
C26D:F0 04    C273    69           beq   exitX
C26F:C9 52            70           cmp   #'R'         ;Reset?
C271:D0 09    C27C    71           bne   term1        ;Go check serial
C273:A9 98            72 exitX     lda   #$98         ;return a CTRL-X
C275:7A               73 exit1     ply
C276:FA               74           plx
C277:60               75           rts
C278:18               76 goremote  clc                ;Into remote mode
C279:20 A3 C7         77 goterm    jsr   swsttm       ;Into terminal mode
C27C:          C27C   78 term1     equ   *
C27C:20 4C CC         79           jsr   showcur      ;Get current char on screen
C27F:48               80 serin     pha
C280:20 D9 C7         81 sinokbd   jsr   swread       ;Is it ready?
C283:B0 09    C28E    82           bcs   sidata      ;Branch if we got data
C285:BD B8 06         83           lda   flags,x      ;Is keyboard enabled?
C288:29 10            84           and   #$10
C28A:F0 D2    C25E    85           beq   testkbd      ;Branch if enabled
C28C:80 F2    C280    86           bra   sinokbd      ;Go test acia again
C28E:A8               87 sidata    tay                ;Save new input in y for now
C28F:68               88           pla
C290:5A               89           phy                ;Save new char on stack
C291:20 B8 C3         90           jsr   storch       ;Fix the screen
C294:68               91           pla                ;Get the new data
C295:BC 38 06         92           ldy   eschar,x     ;If 0, don't modify char
C298:F0 12    C2AC    93           beq   sinomod
C29A:09 80            94           ora   #$80         ;Apple loves the high bit
C29C:C9 91            95           cmp   #xon
C29E:F0 DC    C27C    96           beq   term1        ;Ignore ^Q
C2A0:C9 FF            97           cmp   #$FF         ;Ignore FFs
C2A2:F0 D8    C27C    98           beq   term1
C2A4:C9 92            99           cmp   #$92         ;^R for remote?
C2A6:F0 D0    C278   100           beq   goremote
C2A8:C9 94           101           cmp   #$94         ;^T for terminal mode?
C2AA:F0 CD    C279   102           beq   goterm
C2AC:3C B8 03        103 sinomod   bit   sermode,x    ;In terminal mode?
C2AF:50 C4    C275   104           bvc   exit1        ;Return to user if not A = char
C2B1:20 ED FD        105           jsr   cout         ;Onto the screen with it
C2B4:80 C6    C27C   106           bra   term1
C2B6:          C2B6  107 default    equ   *            ;Set up the defaults
C2B6:20 9A CF        108           jsr   moveirq      ;make sure irq vectors ok
C2B9:BC 29 C2        109           ldy   defidx-$C1,x ;Index into alt screen. Table in command
C2BC:20 7C C3        110 defloop   jsr   getalt       ;Get default from alt screen
C2BF:48              111           pha
C2C0:88              112           dey
C2C1:30 04    C2C7   113           bmi   defff        ;Done if minus
C2C3:C0 03           114           cpy   #3
C2C5:D0 F5    C2BC   115           bne   defloop      ;Or if 2
C2C7:20 9A CF        116 defff     jsr   moveirq      ;Jam irq vector into LC
C2CA:68              117           pla                ;Command, control & flags on stack
C2CB:BC 2B C2        118           ldy   devno,x
```

```
C2CE:99 FB BF      119           sta   scntl,y       ;Set command reg
C2D1:68            120           pla
C2D2:99 FA BF      121           sta   scomd,y
C2D5:68            122           pla
C2D6:9D B8 06      123           sta   flags,x       ;And the flags
C2D9:29 01         124           and   #1            ;A = $01 (^A) if comm mode
C2DB:D0 02   C2DF  125           bne   defcom
C2DD:A9 09         126           lda   #9            ;^I for serial port
C2DF:9D 38 06      127 defcom    sta   eschar,x
C2E2:68            128           pla                  ;Get printer width
C2E3:9D B8 04      129           sta   pwdth,x
C2E6:9E B8 03      130           stz   sermode,x
C2E9:60            131           rts
C2EA:03 07         132 defidx    dfb   3,7
C2EC:        00C1  133 sltdmy    equ   <serslot      ;Make table for hardware access
C2EC:        C22B  134 devno     equ   *-sltdmy
C2EC:A0 B0         135           dfb   $A0,$B0
C2EE:        0012  136           ds    $C300-*,$00
C300:               34           include c3space     ;80 column card @ $C300
```

```
C300:                 2 ****************************************
C300:                 3 *
C300:                 4 * THIS IS THE $C3XX ROM SPACE:
C300:                 5 *
C300:                 6 ****************************************
C300:48               7 C3ENTRY   PHA                 ;save regs
C301:DA               8           PHX
C302:5A               9           PHY
C303:80 12    C317   10           BRA   BASICINIT   ;and init video firmware
C305:38               11 C3KEYIN  SEC                 ;Pascal 1.1 ID byte
C306:90               12          DFB   $90         ;BCC OPCODE (NEVER TAKEN)
C307:18               13 C3COUT1  CLC                 ;Pascal 1.1 ID byte
C308:80 1A    C324    14          BRA   BASICENT    ;=>go print/read char
C30A:EA               15          NOP
C30B:                 16 *
C30B:                 17 * PASCAL 1.1 FIRMWARE PROTOCOL TABLE:
C30B:                 18 *
C30B:01               19          DFB   $01         ;GENERIC SIGNATURE BYTE
C30C:88               20          DFB   $88         ;DEVICE SIGNATURE BYTE
C30D:                 21 *
C30D:2C               22          DFB   >JPINIT     ;PASCAL INIT
C30E:2F               23          DFB   >JPREAD     ;PASCAL READ
C30F:32               24          DFB   >JPWRITE    ;PASCAL WRITE
C310:35               25          DFB   >JPSTAT     ;PASCAL STATUS
C311:                 26 ****************************************
C311:                 27 *
C311:                 28 * 128K SUPPORT ROUTINE ENTRIES:
C311:                 29 *
C311:4C AF C7         30          JMP   SWAUX       ;MEMORY MOVE ACROSS BANKS
C314:4C B5 C7         31          JMP   SWXFER      ;TRANSFER ACROSS BANKS
C317:                 32 ****************************************
C317:                 33 *
C317:                 34 ****************************************
C317:                 35 * BASIC I/O ENTRY POINT:
C317:                 36 ****************************************
C317:                 37 *
C317:20 20 CE         38 BASICINIT JSR  HOOKUP        ;COPYROM if needed, sethooks
C31A:20 BE CD         39          JSR   SET80       ;setup 80 columns
C31D:20 58 FC         40          JSR   HOME        ;clear screen
C320:7A               41          PLY
C321:FA               42          PLX               ;restore X
C322:68               43          PLA               ;restore char
C323:18               44          CLC               ;output a character
C324:                 45 *
C324:B0 03    C329    46 BASICENT  BCS   BINPUT       ;=>carry me to input
C326:4C F6 FD         47 BPRINT    JMP   COUTZ       ;print a character
C329:4C 1B FD         48 BINPUT    JMP   KEYIN       ;get a keystroke
C32C:                 49 *
C32C:4C 41 CF         50 JPINIT    JMP   PINIT       ;pascal init
C32F:4C 35 CF         51 JPREAD    JMP   PASREAD     ;pascal read
C332:4C C2 CE         52 JPWRITE   JMP   PWRITE      ;pascal write
C335:4C B1 CE         53 JPSTAT    JMP   PSTATUS     ;pascal status call
C338:                 54 *
C338:                 55 * COPYROM is called when the video firmware is
C338:                 56 * initialized.  If the language card is switched
C338:                 57 * in for reading, it copies the F8 ROM to the
C338:                 58 * language card and restores the state of the
C338:                 59 * language card.
```

```
C338:                60 *
C338:A9 06           61 COPYROM   LDA   #GOODF8       ;get the ID byte
C33A:                62 *
C33A:                63 * Compare ID bytes to whatever is readable.  If it
C33A:                64 * matches, all is ok.  If not, need to copy.
C33A:                65 *
C33A:CD B3 FB        66           CMP   F8VERSION     ;does it match?
C33D:F0 3C   C37B    67           BEQ   ROMOK
C33F:20 60 C3        68           JSR   SETROM        ;read ROM, write RAM, save state
C342:A9 F8           69           LDA   #$F8          ;from F800-FFFF
C344:85 37           70           STA   CSWH
C346:64 36           71           STZ   CSWL
C348:B2 36           72 COPYROM2  LDA   (CSWL)        ;get a byte
C34A:92 36           73           STA   (CSWL)        ;and save a byte
C34C:E6 36           74           INC   CSWL
C34E:D0 F8   C348    75           BNE   COPYROM2
C350:E6 37           76           INC   CSWH
C352:D0 F4   C348    77           BNE   COPYROM2      ;fall into RESETLC
C354:                78 *
C354:                79 * RESETLC resets the language card to the state
C354:                80 * determined by SETROM.  It always leaves the card
C354:                81 * write enabled.
C354:                82 *
C354:DA              83 RESETLC   PHX                 ;save X
C355:AE 78 04        84           LDX   ROMSTATE      ;get the state
C358:3C 81 C0        85           BIT   ROMIN,X       ;set bank & ROM/RAM read
C35B:3C 81 C0        86           BIT   ROMIN,X       ;set write enable
C35E:FA              87           PLX                 ;restore X
C35F:60              88           RTS
C360:                89 *
C360:                90 * SETROM switches in the ROM for reading, the RAM
C360:                91 * for writing, and it saves the state of the
C360:                92 * language card.  It does not save the write
C360:                93 * protect status of the card.
C360:                94 *
C360:DA              95 SETROM    PHX                 ;save x
C361:A2 00           96           LDX   #0            ;assume write enable,bank2,ROMRD
C363:2C 11 C0        97           BIT   RDLCBNK2      ;is bank 2 switched in?
C366:30 02   C36A    98           BMI   NOT1          ;=>yes
C368:A2 08           99           LDX   #$8           ;indicate bank 1
C36A:2C 12 C0       100 NOT1      BIT   RDLCRAM       ;is LC RAM readable?
C36D:10 02   C371   101           BPL   NOREAD        ;=>no
C36F:E8             102           INX                 ;indicate RAM read
C370:E8             103           INX
C371:2C 81 C0       104 NOREAD    BIT   $C081         ;ROM read
C374:2C 81 C0       105           BIT   $C081         ;RAM write
C377:8E 78 04       106           STX   ROMSTATE      ;save state
C37A:FA             107           PLX                 ;restore X
C37B:60             108 ROMOK     RTS
C37C:               109 *
C37C:               110 * GETALT reads a byte from aux memory screenholes.
C37C:               111 * Y is the index to the byte (0-7) indexed off of
C37C:               112 * address $478.
C37C:               113 *
C37C:AD 13 C0       114 GETALT    LDA   RDRAMRD       ;save state of aux memory
C37F:0A             115           ASL   A
C380:AD 18 C0       116           LDA   RD80COL       ;and of the 80STORE switch
C383:08             117           PHP
```

```
C384:8D 00 C0      118           STA   CLR80COL      ;no 80STORE to get page 1
C387:8D 03 C0      119           STA   RDCARDRAM     ;pop in the other half of RAM
C38A:B9 78 04      120           LDA   $478,Y        ;read the desired byte
C38D:28            121           PLP                 ;and restore memory
C38E:B0 03   C393  122           BCS   GETALT1
C390:8D 02 C0      123           STA   RDMAINRAM
C393:10 03   C398  124 GETALT1   BPL   GETALT2
C395:8D 01 C0      125           STA   SET80COL
C398:60            126 GETALT2   RTS
C399:              127 *
C399:09 80         128 UPSHIFT0  ORA   #$80          ;set high bit for execs
C39B:C9 FB         129 UPSHIFT   CMP   #$FB
C39D:B0 06   C3A5  130           BCS   X.UPSHIFT
C39F:C9 E1         131           CMP   #$E1
C3A1:90 02   C3A5  132           BCC   X.UPSHIFT
C3A3:29 DF         133           AND   #$DF
C3A5:60            134 X.UPSHIFT RTS
C3A6:              135 *
C3A6:              136 * GETCOUT performs COUT for GETLN.  It disables the
C3A6:              137 * echoing of control characters by clearing the
C3A6:              138 * M.CTL mode bit, prints the char, then restores
C3A6:              139 * M.CTL.  NOESC is used by the RDKEY routine to
C3A6:              140 * disable escape sequences.
C3A6:              141 *
C3A6:48            142 GETCOUT   PHA                 ;save char to print
C3A7:A9 08         143           LDA   #M.CTL        ;disable control chars
C3A9:1C FB 04      144           TRB   VMODE         ;by clearing M.CTL
C3AC:68            145           PLA                 ;restore character
C3AD:20 ED FD      146           JSR   COUT          ;and print it
C3B0:4C 44 FD      147           JMP   NOESCAPE      ;enable control chars
C3B3:              148 *
C3B3:              149 * STORCH determines loads the current cursor position,
C3B3:              150 *        inverts the character, and displays it
C3B3:              151 * STORCHAR inverts the character and displays it at the
C3B3:              152 *        position stored in Y
C3B3:              153 * STORY  determines the current cursor position, and
C3B3:              154 *        displays the character without inverting it
C3B3:              155 * STORE  displays the char at the position in Y
C3B3:              156 *
C3B3:              157 * If mouse characters are enabled (VMODE bit 0 = 0)
C3B3:              158 * then mouse characters ($40-$5F) are displayed when
C3B3:              159 * the alternate character set is switched in.  Normally
C3B3:              160 * values $40-$5F are shifted to $0-$1F before display.
C3B3:              161 *
C3B3:              162 * Calls to GETCUR trash Y
C3B3:              163 *
C3B3:20 9D CC      164 STORY     JSR   GETCUR        ;get newest cursor into Y
C3B6:80 09   C3C1  165           BRA   STORE
C3B8:              166 *
C3B8:20 9D CC      167 STORCH    JSR   GETCUR        ;first, get cursor position
C3BB:24 32         168           BIT   INVFLG        ;normal or inverse?
C3BD:30 02   C3C1  169           BMI   STORE         ;=>normal, store it
C3BF:29 7F         170           AND   #$7F          ;inverse it
C3C1:5A            171 STORE     PHY                 ;save real Y
C3C2:09 00         172           ORA   #0            ;does char have high bit set?
C3C4:30 15   C3DB  173           BMI   STORE1        ;=>yes, don't do mouse check
C3C6:48            174           PHA                 ;save char
C3C7:AD FB 04      175           LDA   VMODE         ;is mouse bit set?
```

```
C3CA:6A              176            ROR   A
C3CB:68              177            PLA                     ;restore char
C3CC:90 0D   C3DB   178            BCC   STORE1            ;=>no, don't do mouse shift
C3CE:2C 1E C0       179            BIT   ALTCHARSET        ;no shift if ][ char set
C3D1:10 08   C3DB   180            BPL   STORE1            ;=> it is!
C3D3:49 40          181            EOR   #$40              ;$40-$5F=>0-$1f
C3D5:89 60          182            BIT   #$60
C3D7:F0 02   C3DB   183            BEQ   STORE1
C3D9:49 40          184            EOR   #$40
C3DB:2C 1F C0       185 STORE1     BIT   RD80VID           ;80 columns?
C3DE:10 19   C3F9   186            BPL   STORE5            ;=>no, store char
C3E0:48             187            PHA                     ;save (shifted) char
C3E1:8D 01 C0       188            STA   SET80COL          ;hit 80 store
C3E4:98             189            TYA                     ;get proper Y
C3E5:45 20          190            EOR   WNDLFT            C=1 if char in main ram
C3E7:4A             191            LSR   A
C3E8:B0 04   C3EE   192            BCS   STORE2            ;=>yes, main RAM
C3EA:AD 55 C0       193            LDA   TXTPAGE2          ;else flip in aux RAM
C3ED:C8             194            INY                     ;do this for odd left, aux bytes
C3EE:98             195 STORE2     TYA                     ;divide pos'n by 2
C3EF:4A             196            LSR   A
C3F0:A8             197            TAY
C3F1:68             198            PLA                     ;get (shifted) char
C3F2:91 28          199 STORE3     STA   (BASL),Y          ;stuff it
C3F4:2C 54 C0       200            BIT   TXTPAGE1          ;else restore page1
C3F7:7A             201 STORE4     PLY                     ;restore real Y
C3F8:60             202            RTS                     ;und exit
C3F9:               203 *
C3F9:91 28          204 STORE5     STA   (BASL),Y          ;do 40 column store
C3FB:7A             205            PLY                     ;restore Y
C3FC:60             206            RTS                     ;and exit
C3FD:         0003  207            DS    $C400-*,$00
C400:                35            include mouse           ;Equates for the mouse
```

```
C400:                2          msb   ON
C400:                3 ****************************************
C400:                4 *
C400:                5 * Mouse firmware for the Chels
C400:                6 *
C400:                7 *  by Rich Williams
C400:                8 *  July, 1983
C400:                9 *
C400:               10 ****************************************


C400:               12 ****************************************
C400:               13 *
C400:               14 * Equates
C400:               15 *
C400:               16 ****************************************


C400:               18 * Input bounds are in scratch area
C400:  0478         19 moutemp   equ   $478          ;Temporary storage
C400:  0478         20 minl      equ   $478
C400:  04F8         21 maxl      equ   $4F8
C400:  0578         22 minh      equ   $578
C400:  05F8         23 maxh      equ   $5F8
C400:               24 * Mouse bounds in slot 5 screen area
C400:  047D         25 minxl     equ   $47D
C400:  04FD         26 minyl     equ   $4FD
C400:  057D         27 minxh     equ   $57D
C400:  05FD         28 minyh     equ   $5FD
C400:  067D         29 maxxl     equ   $67D
C400:  06FD         30 maxyl     equ   $6FD
C400:  077D         31 maxxh     equ   $77D
C400:  07FD         32 maxyh     equ   $7FD
C400:               33 * Mouse holes in slot 4 screen area
C400:  047C         34 mouxl     equ   $47C          ;X position low byte
C400:  04FC         35 mouyl     equ   $4FC          ;Y position low byte
C400:  057C         36 mouxh     equ   $57C          ;X position high byte
C400:  05FC         37 mouyh     equ   $5FC          ;Y position high byte
C400:  067C         38 mouarm    equ   $67C          ;Arm interrupts from movement or button
C400:  077C         39 moustat   equ   $77C          ;Mouse status
C400:               40 * Moustat provides the following
C400:               41 *  D7= Button pressed
C400:               42 *  D6= Status of button on last read
C400:               43 *  D5= Moved since last read
C400:               44 *  D4= Reserved
C400:               45 *  D3= Interrupt from VBL
C400:               46 *  D2= Interrupt from button
C400:               47 *  D1= Interrupt from movement
C400:               48 *  D0= Reserved
C400:  07FC         49 moumode   equ   $7FC          ;Mouse mode
C400:               50 *  D7 = 1 if user wants control of mouse interrupts
C400:               51 *  D6-D4= Unused
C400:               52 *  D3= VBL active
C400:               53 *  D2= VBL interrupt on button
C400:               54 *  D1= VBL interrupt on movement
C400:               55 *  D0= Mouse active
```

```
C400:          0020   56 movarm    equ   $20
C400:          000C   57 vblmode   equ   $0C
C400:          0004   58 butmode   equ   $04          ;D2 mask
C400:          0002   59 movmode   equ   $02          ;D1 mask


C400:                 61 * Hardware addresses
C400:          C015   62 mouxint   equ   $C015        ;D7 = x interrupt
C400:          C017   63 mouyint   equ   $C017        ;D7 = y interrupt
C400:          C019   64 vblint    equ   $C019        ;D7 = vbl interrupt
C400:          C078   65 ioudsbl   equ   $C078        ;Disable iou access
C400:          C079   66 iouenbl   equ   $C079        ;Enable iou access
C400:          C048   67 mouclr    equ   $C048        ;Clear mouse interrupt
C400:          C058   68 iou       equ   $C058        ;IOU interrupt switches
C400:          C058   69 moudsbl   equ   $C058        ;Disable mouse interrupts
C400:          C059   70 mouenbl   equ   $C059        ;Enable mouse interrupts
C400:          C063   71 moubut    equ   $C063        ;D7 = Mouse button
C400:          C066   72 moux1     equ   $C066        ;D7 = X1
C400:          C067   73 mouy1     equ   $C067        ;D7 = Y1
C400:          C070   74 vblclr    equ   $C070        ;Clear VBL interrupt
C400:                 75 *
C400:                 76 * Other addresses
C400:                 77 *
C400:          0200   78 inbuf     equ   $200         ;Input buffer
C400:          0214   79 binl      equ   inbuf+20     ;Temp for binary conversion
C400:          0215   80 binh      equ   inbuf+21
C400:                 36          include mcode      ;Mouse @ $C400
```

```
C400:                  2 ****************************************
C400:                  3 *
C400:                  4 * Entry points for mouse firmware
C400:                  5 *
C400:                  6 ****************************************
C400:80 05   C407      7 mbasic    bra   outent
C402:A2 03             8 pnull     ldx   #3
C404:60                9           rts                  ;Null for pascal entry
C405:38               10 inent     sec                  ;Signature bytes
C406:90               11           dfb   $90
C407:18               12 outent    clc
C408:4C CF C5         13           jmp   xmbasic        ;Go do basic entry
C40B:01               14           dfb   $01            ;More signature stuff
C40C:20               15           dfb   $20
C40D:02               16           dfb   >pnull
C40E:02               17           dfb   >pnull
C40F:02               18           dfb   >pnull
C410:02               19           dfb   >pnull
C411:00               20           dfb   $0
C412:3B               21           dfb   >xsetmou       ;SETMOUSE
C413:DC               22           dfb   >xmtstint      ;SERVEMOUSE
C414:93               23           dfb   >xmread        ;READMOUSE
C415:82               24           dfb   >xmclear       ;CLEARMOUSE
C416:69               25           dfb   >noerror       ;POSMOUSE
C417:BD               26           dfb   >xmclamp       ;CLAMPMOUSE
C418:6B               27           dfb   >xmhome        ;HOMEMOUSE
C419:1A               28           dfb   >initmouse     ;INITMOUSE
C41A:                 29 *  dfb >pnull
C41A:                 30 *  dfb >goxmint
```

```
C41A:                 32 ****************************************
C41A:                 33 *
C41A:                 34 * Initmouse - resets the mouse
C41A:                 35 *  Also clears all of the mouse holes
C41A:                 36 * note that iou access fires pdlstrb & makes mouse happy
C41A:                 37 *
C41A:                 38 ****************************************
C41A:         C41A    39 initmouse equ   *
C41A:9C 7C 07         40           stz   moustat        ;Clear status
C41D:A2 80            41           ldx   #$80
C41F:A0 01            42           ldy   #1
C421:9E 7D 04         43 xrloop    stz   minxl,x        ;Minimum = $0000
C424:9E 7D 05         44           stz   minxh,x
C427:A9 FF            45           lda   #$FF           ;Maximum = $03FF
C429:9D 7D 06         46           sta   maxxl,x
C42C:A9 03            47           lda   #03
C42E:9D 7D 07         48           sta   maxxh,x
C431:A2 00            49           ldx   #0
C433:88               50           dey
C434:10 EB   C421     51           bpl   xrloop
C436:20 6B C4         52           jsr   xmhome         ;Clear the mouse holes
C439:A9 00            53           lda   #0             ;Fall into SETMOU


C43B:                 55 ****************************************
C43B:                 56 *
C43B:                 57 * XSETMOU - Sets the mouse mode to A
C43B:                 58 *
C43B:                 59 ****************************************
C43B:         C43B    60 xsetmou   equ   *
C43B:AA               61           tax
C43C:20 9A CF         62           jsr   moveirq        ;Make sure interrupt vector is right
C43F:8A               63           txa                   ;Only x preserved by moveirq
C440:8D 78 04         64           sta   moutemp
C443:4A               65           lsr   A              ;D0 = 1 if mouse active
C444:0D 78 04         66           ora   moutemp        ;D2 = 1 if vbl active
C447:C9 10            67           cmp   #$10           ;If >=$10 then invalid mode
C449:B0 1F   C46A     68           bcs   sminvalid
C44B:29 05            69           and   #5             ;Extract VBL & Mouse
C44D:F0 01   C450     70           beq   xsoff          ;Turning it off?
C44F:58               71           cli                   ;If not, ints active
C450:69 55            72 xsoff     adc   #$55           ;Make iou byte C=0


C452:                 74 ****************************************
C452:                 75 *
C452:                 76 * SETIOU - Sets the IOU interrupt modes to A
C452:                 77 *  Inputs: A = Bits to change
C452:                 78 *  D7 = Y int on falling edge
C452:                 79 *  D6 = Y int on rising edge
C452:                 80 *  D5 = X int on falling edge
C452:                 81 *  D4 = X int on rising edge
C452:                 82 *  D3 = Enable VBL int
C452:                 83 *  D2 = Disable VBL int
C452:                 84 *  D1 = Enable mouse int
C452:                 85 *  D0 = Disable mouse int
```

```
C452:                 86 *
C452:                 87 *
C452:                 88 ****************************************
C452:          C452   89 setiou     equ   *
C452:08               90            php
C453:78               91            sei                  ;Don't allow ints while iou enabled
C454:8E FC 07         92            stx   moumode
C457:8D 79 C0         93            sta   iouenbl        ;Enable iou access
C45A:A2 08            94            ldx   #8
C45C:CA               95 siloop     dex
C45D:0A               96            asl   A              ;Get a bit to check
C45E:90 03   C463     97            bcc   sinoch         ;No change if C=0
C460:9D 58 C0         98            sta   iou,x          ;Set it
C463:D0 F7   C45C     99 sinoch     bne   siloop         ;Any bits left in A?
C465:8D 78 C0        100            sta   ioudsbl        ;Turn off iou access
C468:28              101            plp
C469:18              102 noerror    clc
C46A:60              103 sminvalid  rts


C46B:                105 ****************************************
C46B:                106 *
C46B:                107 * XMHOME- Clears mouse position & status
C46B:                108 *
C46B:                109 ****************************************
C46B:          C46B  110 xmhome     equ   *
C46B:A2 80           111            ldx   #$80           ;Point mouse to upper left
C46D:80 02   C471    112            bra   xmh2
C46F:A2 00           113 xmhloop    ldx   #0
C471:BD 7D 04        114 xmh2       lda   minxl,x
C474:9D 7C 04        115            sta   mouxl,x
C477:BD 7D 05        116            lda   minxh,x
C47A:9D 7C 05        117            sta   mouxh,x
C47D:CA              118            dex
C47E:10 EF   C46F    119            bpl   xmhloop
C480:80 0C   C48E    120            bra   xmcdone


C482:                122 ****************************************
C482:                123 *
C482:                124 * XMCLEAR - Sets the mouse to 0,0
C482:                125 *
C482:                126 ****************************************
C482:          C482  127 xmclear    equ   *
C482:9C 7C 04        128            stz   mouxl
C485:9C 7C 05        129            stz   mouxh
C488:9C FC 04        130            stz   mouyl
C48B:9C FC 05        131            stz   mouyh
C48E:9C 7C 06        132 xmcdone    stz   mouarm
C491:18              133            clc
C492:60              134            rts
```

```
C493:                 136 ****************************************
C493:                 137 *
C493:                 138 * XMREAD - Updates the screen holes
C493:                 139 *
C493:                 140 ****************************************
C493:          C493   141 xmread     equ   *
C493:A9 20            142           lda   #movarm       ;Has mouse moved?
C495:1C 7C 07         143           trb   moustat       ;Clear moved bit in stat
C498:2D 7C 06         144           and   mouarm
C49B:1C 7C 06         145           trb   mouarm        ;Clear arm bit
C49E:2C FC 07         146           bit   moumode       ;If D7 = 1 leave buttons alone
C4A1:30 13    C4B6    147           bmi   xmrd2
C4A3:2C 63 C0         148           bit   moubut        ;Button pressed?
C4A6:30 02    C4AA    149           bmi   xrbut
C4A8:09 80            150           ora   #$80
C4AA:2C 7C 07         151 xrbut     bit   moustat       ;Pressed last time?
C4AD:10 02    C4B1    152           bpl   xrbut2
C4AF:09 40            153           ora   #$40
C4B1:8D 7C 07         154 xrbut2    sta   moustat
C4B4:18               155           clc
C4B5:60               156           rts
C4B6:          C4B6   157 xmrd2      equ   *             ;Leave button bits alone
C4B6:0D 7C 07         158           ora   moustat
C4B9:29 E0            159           and   #$E0          ;Button bits
C4BB:80 F4    C4B1    160           bra   xrbut2


C4BD:                 162 ****************************************
C4BD:                 163 *
C4BD:                 164 * XMCLAMP - Store new bounds
C4BD:                 165 *  Inputs A = 1 for Y, 0 for X axis
C4BD:                 166 *    minl, minh, maxl, maxh = new bounds
C4BD:                 167 *
C4BD:                 168 ****************************************
C4BD:          C4BD   169 xmclamp    equ   *
C4BD:6A               170           ror   A             ;1 -> 80
C4BE:6A               171           ror   A
C4BF:29 80            172           and   #$80
C4C1:AA               173           tax
C4C2:AD 78 04         174           lda   minl
C4C5:9D 7D 04         175           sta   minxl,x
C4C8:AD 78 05         176           lda   minh
C4CB:9D 7D 05         177           sta   minxh,x
C4CE:AD F8 04         178           lda   maxl
C4D1:9D 7D 06         179           sta   maxxl,x
C4D4:AD F8 05         180           lda   maxh
C4D7:9D 7D 07         181           sta   maxxh,x
C4DA:18               182           clc                 ;No error
C4DB:60               183           rts


C4DC:                 185 ****************************************
C4DC:                 186 * XMTSTINT - Checks mouse status bits
C4DC:                 187 * Used for user mouse interrupt
C4DC:                 188 ****************************************
C4DC:          C4DC   189 xmtstint  equ   *
```

```
C4DC:48             190           pha
C4DD:18             191           clc
C4DE:A9 0E          192           lda   #$0E
C4E0:2D 7C 07       193           and   moustat
C4E3:D0 01   C4E6   194           bne   nostat2
C4E5:38             195           sec
C4E6:68             196 nostat2   pla
C4E7:60             197           rts
C4E8:        0013   198           ds    $C4FB-*,0
C4FB:D6             199           dfb   $D6            ;Signature byte
C4FC:        0004   200           ds    $C500-*,$00
C500:                37           include misc        ;Miscellaneous junk
C500:        008E     1           ds    $C58E-*,0
```

```
C58E:                 3 ****************************************
C58E:                 4 *
C58E:                 5 * MAKTBL - Makes a deniblizing table for the disk II boot
C58E:                 6 *
C58E:                 7 ****************************************
C58E:A2 03            8 MAKTBL     LDX   #$03
C590:A0 00            9            LDY   #0
C592:86 3C           10 TBLLOOP    STX   BOOTTMP
C594:8A              11            TXA
C595:0A              12            ASL   A
C596:24 3C           13            BIT   BOOTTMP
C598:F0 10   C5AA    14            BEQ   NOPATRN
C59A:05 3C           15            ORA   BOOTTMP
C59C:49 FF           16            EOR   #$FF
C59E:29 7E           17            AND   #$7E
C5A0:B0 08   C5AA    18 TBLLOOP2   BCS   NOPATRN
C5A2:4A              19            LSR   A
C5A3:D0 FB   C5A0    20            BNE   TBLLOOP2
C5A5:98              21            TYA
C5A6:9D 56 03        22            STA   DNIBL,X
C5A9:C8              23            INY
C5AA:E8              24 NOPATRN    INX
C5AB:10 E5   C592    25            BPL   TBLLOOP
C5AD:A9 08           26            LDA   #$08
C5AF:85 27           27            STA   $27
C5B1:A0 7F           28            LDY   #$7F
C5B3:60              29            RTS


C5B4:                31 ****************************************
C5B4:                32 *
C5B4:                33 * GETUP - Get char from input buffer
C5B4:                34 *  iny and upshift it
C5B4:                35 *
C5B4:                36 ****************************************
C5B4:        C5B4    37 getup      equ   *
C5B4:B9 00 02        38            lda   in,y          ;Get character
C5B7:C8              39            iny
C5B8:4C 99 C3        40            jmp   upshift0


C5BB:                42 ****************************************
C5BB:                43 *
C5BB:                44 * This is who we are 9 letters
C5BB:                45 *
C5BB:                46 ****************************************
C5BB:C1 F0 F0 EC     47 apple2c    asc   'Apple        //c'


C5C4:                49 ****************************************
C5C4:                50 *
C5C4:                51 * SHOWINST - Disassemble an instruction and adjust the PC
C5C4:                52 *
C5C4:                53 ****************************************
C5C4:        C5C4    54 showinst   equ   *
```

```
C5C4:20 D0 F8      55           jsr   instdsp
C5C7:20 53 F9      56           jsr   pcadj
C5CA:85 3A         57           sta   pcl
C5CC:84 3B         58           sty   pch
C5CE:60            59           rts


C5CF:              61 ****************************************
C5CF:              62 *
C5CF:              63 * XMBASIC - Basic call to the mouse
C5CF:              64 *
C5CF:              65 ****************************************
C5CF:       C5CF   66 xmbasic   equ   *
C5CF:5A            67           phy
C5D0:B0 1C  C5EE   68           bcs   gobasicin      ;Input?
C5D2:A0 C4         69           ldy   #<mbasic       ;Input from $C400?
C5D4:C4 39         70           cpy   kswh
C5D6:D0 04  C5DC   71           bne   xmbout
C5D8:A4 38         72           ldy   kswl
C5DA:F0 12  C5EE   73           beq   gobasicin
C5DC:DA            74 xmbout    phx                   ;Save X too
C5DD:48            75           pha
C5DE:29 7F         76           and   #$7F            ;We don't care about high bit
C5E0:C9 02         77           cmp   #2
C5E2:B0 06  C5EA   78           bge   mbbad           ;Only 0,1 valid
C5E4:20 3B C4      79           jsr   xsetmou
C5E7:20 6B C4      80           jsr   xmhome
C5EA:68            81 mbbad     pla
C5EB:FA            82           plx
C5EC:7A            83           ply
C5ED:60            84           rts
C5EE:4C 9D C7      85 gobasicin jmp   swbasicin      ;Go to input routine
C5F1:       0004   86           ds    $C5F5-*,0      ;More disk stuff
C5F5:              38           include boot         ;Disk II boot @$C600
C5F5:       000B    1           ds    $C600-*,0      ;Disk II in slot 6
```

```
C600:                 4 ****************************************
C600:                 5 *
C600:                 6 * Disk II boot stuff
C600:                 7 * jumps to slot 5 if boot fails
C600:                 8 *
C600:                 9 ****************************************
C600:A2 20           10            LDX   #$20
C602:A0 00           11            LDY   #$00
C604:64 03           12            STZ   $03
C606:64 3C           13            STZ   $3C
C608:A9 60           14            LDA   #$60
C60A:AA              15            TAX
C60B:86 2B           16 DRV2ENT    STX   SLOTZ
C60D:85 4F           17            STA   BOOTDEV
C60F:5A              18            PHY                   ;Y=1 IF DRIVE 2 BOOT, ELSE Y=0
C610:BD 8E C0        19            LDA   $C08E,X
C613:BD 8C C0        20            LDA   $C08C,X
C616:7A              21            PLY
C617:B9 EA C0        22            LDA   $C0EA,Y         ;SELECT DRIVE 1 OR 2
C61A:BD 89 C0        23            LDA   $C089,X
C61D:A0 50           24            LDY   #$50
C61F:BD 80 C0        25 SEEKZERO   LDA   $C080,X
C622:98              26            TYA
C623:29 03           27            AND   #$03
C625:0A              28            ASL   A
C626:05 2B           29            ORA   SLOTZ
C628:AA              30            TAX
C629:BD 81 C0        31            LDA   $C081,X
C62C:A9 56           32            LDA   #$56
C62E:20 A8 FC        33            JSR   WAIT
C631:88              34            DEY
C632:10 EB   C61F    35            BPL   SEEKZERO
C634:85 26           36            STA   $26
C636:85 3D           37            STA   $3D
C638:85 41           38            STA   $41
C63A:20 8E C5        39            JSR   MAKTBL
C63D:64 03           40 EXTENT1    STZ   $03
C63F:18              41 RDADR      CLC
C640:08              42            PHP
C641:28              43 RETRY1     PLP
C642:A6 2B           44 RDDHDR     LDX   SLOTZ           ;RESTORE X TO $60
C644:C6 03           45            DEC   $03             ;UPDATE RETRY COUNT
C646:D0 0E   C656    46            BNE   RDHD0           ;BRANCH IF NOT OUT OF RETRIES
C648:BD 88 C0        47 FUGIT      LDA   $C088,X         ;SHUT OFF DISK AND QUIT!
C64B:A5 01           48            LDA   LOC1            ;Auto boot from slot6?
C64D:C9 C6           49            CMP   #$C6
C64F:D0 A4   C5F5    50            BNE   BOOTFAIL
C651:4C 00 C5        51            JMP   $C500           ;Maybe slot 5 will talk to us
C654:        0002    52            ds    $C656-*,0       ;Keep alignment
C656:08              53 RDHD0      PHP
C657:88              54 RETRY      DEY
C658:D0 04   C65E    55            BNE   RDHD1
C65A:F0 E5   C641    56            BEQ   RETRY1
C65C:80 DF   C63D    57 EXTENT     BRA   EXTENT1          ;Blows up if this is moved too
C65E:                58 * * * * * * * * * * * * * * * * * * * * *
C65E:                59 * The following code is sacred in it's  *
C65E:                60 * present form.  To change it would     *
C65E:                61 * cause volcanos to errupt, the ground  *
```

```
C65E:                    62 * to shake, and ProDOS not to boot!     *
C65E:                    63 * * * * * * * * * * * * * * * * * * * * *
C65E:BD 8C C0            64 RDHD1    LDA   $C08C,X
C661:10 FB    C65E       65          BPL   RDHD1
C663:49 D5               66 ISMRK1   EOR   #$D5
C665:D0 F0    C657       67          BNE   RETRY
C667:BD 8C C0            68 RDHD2    LDA   $C08C,X
C66A:10 FB    C667       69          BPL   RDHD2
C66C:C9 AA               70          CMP   #$AA
C66E:D0 F3    C663       71          BNE   ISMRK1
C670:EA                  72          NOP
C671:BD 8C C0            73 RDHD3    LDA   $C08C,X
C674:10 FB    C671       74          BPL   RDHD3
C676:C9 96               75          CMP   #$96
C678:F0 09    C683       76          BEQ   RDSECT
C67A:28                  77          PLP
C67B:90 C2    C63F       78          BCC   RDADR
C67D:49 AD               79          EOR   #$AD
C67F:F0 25    C6A6       80          BEQ   RDATA
C681:D0 BC    C63F       81          BNE   RDADR
C683:A0 03               82 RDSECT   LDY   #$03
C685:85 40               83 RDSEC1   STA   $40
C687:BD 8C C0            84 RDSEC2   LDA   $C08C,X
C68A:10 FB    C687       85          BPL   RDSEC2
C68C:2A                  86          ROL   A
C68D:85 3C               87          STA   BOOTTMP
C68F:BD 8C C0            88 RDSEC3   LDA   $C08C,X
C692:10 FB    C68F       89          BPL   RDSEC3
C694:25 3C               90          AND   BOOTTMP
C696:88                  91          DEY
C697:D0 EC    C685       92          BNE   RDSEC1
C699:28                  93          PLP
C69A:C5 3D               94          CMP   $3D
C69C:D0 A1    C63F       95          BNE   RDADR
C69E:A5 40               96          LDA   $40
C6A0:C5 41               97          CMP   $41
C6A2:D0 9B    C63F       98 BADRD1   BNE   RDADR
C6A4:B0 9C    C642       99          BCS   RDDHDR
C6A6:A0 56              100 RDATA    LDY   #$56
C6A8:84 3C              101 RDAT0    STY   BOOTTMP
C6AA:BC 8C C0           102 RDAT1    LDY   $C08C,X
C6AD:10 FB    C6AA      103          BPL   RDAT1
C6AF:59 D6 02           104          EOR   DNIBL-$80,Y
C6B2:A4 3C              105          LDY   BOOTTMP
C6B4:88                 106          DEY
C6B5:99 00 03           107          STA   NBUF1,Y
C6B8:D0 EE    C6A8      108          BNE   RDAT0
C6BA:84 3C              109 RDAT2    STY   BOOTTMP
C6BC:BC 8C C0           110 RDAT3    LDY   $C08C,X
C6BF:10 FB    C6BC      111          BPL   RDAT3
C6C1:59 D6 02           112          EOR   DNIBL-$80,Y
C6C4:A4 3C              113          LDY   BOOTTMP
C6C6:91 26              114          STA   ($26),Y
C6C8:C8                 115          INY
C6C9:D0 EF    C6BA      116          BNE   RDAT2
C6CB:BC 8C C0           117 RDAT4    LDY   $C08C,X
C6CE:10 FB    C6CB      118          BPL   RDAT4
C6D0:59 D6 02           119          EOR   DNIBL-$80,Y
```

```
C6D3:D0 CD   C6A2  120 BADREAD   BNE   BADRD1
C6D5:A0 00         121           LDY   #$00
C6D7:A2 56         122 DENIBL    LDX   #$56
C6D9:CA            123 DENIB1    DEX
C6DA:30 FB   C6D7  124           BMI   DENIBL
C6DC:B1 26         125           LDA   ($26),Y
C6DE:5E 00 03      126           LSR   NBUF1,X
C6E1:2A            127           ROL   A
C6E2:5E 00 03      128           LSR   NBUF1,X
C6E5:2A            129           ROL   A
C6E6:91 26         130           STA   ($26),Y
C6E8:C8            131           INY
C6E9:D0 EE   C6D9  132           BNE   DENIB1
C6EB:              133 * * * * * * * * * * * * * * * * *
C6EB:              134 * Code beyond this point is not *
C6EB:              135 * sacred... It may be perverted *
C6EB:              136 * in any manner by any pervert. *
C6EB:              137 * * * * * * * * * * * * * * * * *
C6EB:E6 27         138           INC   $27
C6ED:E6 3D         139           INC   $3D
C6EF:A5 3D         140           LDA   $3D
C6F1:CD 00 08      141           CMP   $0800
C6F4:A6 4F         142           LDX   BOOTDEV
C6F6:90 DB   C6D3  143           BCC   BADREAD
C6F8:4C 01 08      144           JMP   $0801
C6FB:        0005  145           DS    $C700-*,0     ;Last byte must be 0
C700:               39           include switcher    ;Bank switcher @ $C780
```

```
C700:          0080     2            ds    $C780-*,0
C780:                   3 ****************************************
C780:                   4 *
C780:                   5 * Code for switching between banks
C780:                   6 * This code appears in both banks of the rom
C780:                   7 *
C780:                   8 ****************************************
C780:8D 28 C0           9 swrti      sta   rombank        ;RTI to the other bank
C783:40                10            rti
C784:8D 28 C0          11 swrts      sta   rombank        ;RTS to the other bank
C787:60                12 swrtsop    rts
C788:8D 28 C0          13 swreset    sta   rombank        ;Reset routine
C78B:4C 62 FA          14            jmp   reset
C78E:8D 28 C0          15            sta   rombank        ;Interrupt routine
C791:2C 87 C7          16            bit   swrtsop        ;Set V = 1 for other bank
C794:4C 04 C8          17            jmp   irqent
C797:8D 28 C0          18 swpcnv     sta   rombank        ;Protocol converter
C79A:4C F1 C7          19            jmp   swsthk3        ;Jump to sethooks from other side
C79D:8D 28 C0          20 swbasicin  sta   rombank        ;Mouse BASIC routines
C7A0:4C F6 C7          21            jmp   swzzqt3        ;Jump to zzquit from other side
C7A3:8D 28 C0          22 swsttm     sta   rombank        ;Set terminal mode
C7A6:4C F1 C7          23            jmp   swsttm3
C7A9:8D 28 C0          24 swcmd      sta   rombank        ;Serial port command processor
C7AC:4C 06 C8          25            jmp   swcmd3
C7AF:8D 28 C0          26 swaux      sta   rombank        ;Moveaux
C7B2:4C 4E C3          27            jmp   moveaux
C7B5:8D 28 C0          28 swxfer     sta   rombank
C7B8:4C 97 C3          29            jmp   xfer
C7BB:8D 28 C0          30 swmint     sta   rombank        ;Mouse interrupt handler
C7BE:4C 00 C1          31            jmp   mouseint
C7C1:8D 28 C0          32 banger     sta   rombank
C7C4:4C A9 D4          33            jmp   diags
C7C7:8D 28 C0          34 swatalk    sta   rombank        ;Jump to appletalk
C7CA:4C 80 C5          35            jmp   atalk
C7CD:8D 28 C0          36 swser3     sta   rombank        ;Jump to serout3
C7D0:4C 4F C2          37            jmp   serout3
C7D3:8D 28 C0          38 swgetst    sta   rombank        ;Jump to getstat
C7D6:4C AC C2          39            jmp   getstat
C7D9:8D 28 C0          40 swread     sta   rombank        ;Jump to xrdser
C7DC:4C C3 C2          41            jmp   xrdser
C7DF:8D 28 C0          42 swgetb     sta   rombank        ;Jump to getbuf
C7E2:4C F7 C2          43            jmp   getbuf
C7E5:8D 28 C0          44 swzznm     sta   rombank
C7E8:4C E0 D4          45            jmp   zznm
C7EB:8D 28 C0          46 swxfgo     sta   rombank        ;Jump to users xfer dest
C7EE:6C ED 03          47            jmp   ($3ED)
C7F1:20 23 CE          48 swsthk3    jsr   sethooks
C7F4:80 8E   C784      49            bra   swrts
C7F6:20 4D CE          50 swzzqt3    jsr   zzquit
C7F9:80 89   C784      51            bra   swrts
C7FB:        0004      52            ds    $C7FF-*,0
C7FF:00                53            dfb   0              ;Appletalk version number
C800:                  40            include irqbuf       ;Interrupt stuff @$C800
```

```
C800:                3 ****************************************
C800:                4 *
C800:                5 * NEWIRQ - The main (only) IRQ handling routines
C800:                6 * IRQENT - Entry point from alternate rom bank
C800:                7 *
C800:                8 *
C800:                9 * This routine saves the memory state of the machine,
C800:               10 * checks for an internal interrupt, and then calls the user's
C800:               11 * interrupt handler at $3FE.
C800:               12 * The memory state is encoded as follows:
C800:               13 *  D7 = 1 if Alernate zero page / stack
C800:               14 *  D6 = 1 if 80 store and page 2
C800:               15 *  D5 = 1 if Read aux
C800:               16 *  D4 = 1 if Write Aux
C800:               17 *  D3 = 1 if L.C. enabled
C800:               18 *  D2 = 1 if L.C. and $D000 bank 1
C800:               19 *  D1 = 1 if L.C. and $D000 bank 2
C800:               20 *  D0 = 1 if Alternate rom bank
C800:               21 *
C800:               22 * New changes in the interrupt handler are marked with a +
C800:               23 *
C800:               24 ****************************************
C800:4C 9E C1       25           jmp   p1init        ;Pascal 1.0 Initialization
C803:        C803   26 NEWIRQ    EQU   *             ;+
C803:B8             27           CLV                  ;+ V=0 for main bank
C804:        C804   28 IRQENT    EQU   *             ;+ Entry point from other bank assumes
                                                       V=1
C804:48             29           PHA                  ;+ Save A on stack, not $45
C805:DA             30           PHX                  ;+ X too
C806:BA             31           TSX                  ;+ Save stack pointer
C807:68             32           PLA                  ;+ Skip past X
C808:68             33           PLA                  ;+ And A
C809:68             34           PLA                  ;+ Here is the status Oh boy!
C80A:9A             35           TXS                  ;+ Fix the stack pointer
C80B:5A             36           PHY                  ;Save Y too
C80C:AE 66 C0       37           LDX   MOUX1         ;Get mouse info
C80F:AC 67 C0       38           LDY   MOUY1         ;As soon as we can
C812:D8             39           CLD                  ;+ No decimal mode please
C813:29 10          40           AND   #$10          ;+ Test break bit
C815:C9 10          41           CMP   #$10          ;+ C=1 if break. V unchanged
C817:AD 18 C0       42           LDA   RD80COL       ;TEST FOR 80-STORE WITH
C81A:2D 1C C0       43           AND   RDPAGE2       ; PAGE 2 TEXT.
C81D:29 80          44           AND   #$80          ; MAKE IT ZERO OR $80
C81F:F0 05   C826   45           BEQ   IRQ2
C821:8D 54 C0       46           STA   TXTPAGE1
C824:A9 40          47           LDA   #$40          ;SET PAGE 2 RESET BIT.
C826:50 02   C82A   48 IRQ2      BVC   IRQ21         ;+ Which Rombank?
C828:09 01          49           ORA   #01           ;+ Mark other bank
C82A:2C 13 C0       50 IRQ21     BIT   RDRAMRD
C82D:10 05   C834   51           BPL   IRQ3          ;BRANCH IF MAIN RAM READ
C82F:8D 02 C0       52           STA   RDMAINRAM     ;ELSE, SWITCH IT IN
C832:09 20          53           ORA   #$20          ; AND RECORD THE EVENT!
C834:2C 14 C0       54 IRQ3      BIT   RDRAMWRT      ;DO THE SAME FOR RAM WRITE.
C837:10 05   C83E   55           BPL   IRQ4
C839:8D 04 C0       56           STA   WRMAINRAM
C83C:09 10          57           ORA   #$10
C83E:B0 08   C848   58 IRQ4      BCS   IRQ5          ;Branch if break
C840:48             59           PHA                  ;Save machine states so far...
C841:20 BB C7       60           JSR   SWMINT        ;+ Go Test Mouse & ACIA
```

```
C844:90 3C   C882   61            BCC   IRQLCOK       ;+ Branch if it was.  LC unchanged!
C846:68             62            PLA                  ;Restore states recorded so far
C847:18             63            CLC                  ;Reset break/interrupt handler
C848:2C 12 C0       64 IRQ5       BIT   RDLCRAM        ;DETERMINE IF LANGUAGE CARD ACTIVE
C84B:80 03   C850   65            bra   passkip1       ;Skip around pascal 1.0 stuff
C84D:        0000   66            ds    $C84D-*,$00
C84D:4C A8 C1       67            jmp   p1read
C850:        C850   68 passkip1    equ   *
C850:10 0C   C85E   69            BPL   IRQ7
C852:09 0C          70            ORA   #$C            ;SET TWO BITS SO RESTORED
C854:2C 11 C0       71            BIT   RDLCBNK2       ; LANGUAGE CARD IS WRITE ENABLED
C857:10 02   C85B   72            BPL   IRQ6           ;BRANCH IF NOT PAGE 2 OF $D000
C859:49 06          73            EOR   #$6            ;ENABLE READ FOR PAGE 2 ON EXIT
C85B:8D 81 C0       74 IRQ6       STA   ROMIN
C85E:2C 16 C0       75 IRQ7       BIT   RDALTZP        ;LAST...AND VERY IMPORTANT!
C861:10 0D   C870   76            BPL   IRQ8           ; UNLESS IT IS NOT ENABLED
C863:BA             77            TSX                  ;SAVE CURRENT STACK POINTER
C864:8E 01 01       78            STX   $101           ;AT BOTTOM OF STACK
C867:AE 00 01       79            LDX   $100           ;GET MAIN STACK POINTER
C86A:9A             80            TXS
C86B:8D 08 C0       81            STA   SETSTDZP
C86E:09 80          82            ORA   #$80
C870:B0 35   C8A7   83 IRQ8       BCS   GOBREAK
C872:48             84            PHA
C873:A9 C8          85            LDA   #<IRQDONE
C875:48             86            PHA
C876:A9 7F          87            LDA   #>IRQDONE      ;SAVE RETURN IRQ ADDR
C878:48             88            PHA
C879:A9 04          89            LDA   #4             ; SO WHEN INTERRUPT DOES RTI
C87B:48             90            PHA                  ; IT RETURNS TO IRQDONE.
C87C:6C FE 03       91            JMP   ($3FE)         ;PROCESS EXTERNAL INTERRUPT

;87F:               93 * The user's RTI returns here
C87F:               94 * BEWARE
C87F:               95 *  The rom must be reenabled with a LDA romin
C87F:               96 *  This way if the LC was write protected, it still is
C87F:               97 *   if it was write enabled, it still is
C87F:               98 *   if it was being write enabled ( 2 ldas), it still will be
C87F:               99 *  The restore loop uses an INC because some of the switches are read
C87F:              100 *  and some are write.  It must be an INC abs,x since both the 6502 and
C87F:              101 *  the 65C02 do two reads before the write (for different reasons).
C87F:AD 81 C0      102 IRQDONE   LDA   ROMIN          ;+ Did some clown bank out the rom?
C882:68            103 IRQLCOK   PLA                  ;Recover machine state
C883:10 07   C88C  104            BPL   IRQDN1         ;Branch if main zp was active
C885:8D 09 C0      105            STA   SETALTZP
C888:AE 01 01      106            LDX   $101           ;Restore alternate stack pointer
C88B:9A            107            TXS
C88C:A0 06         108 IRQDN1    LDY   #$06           ;+ Y = index into table of switches
C88E:10 06   C896  109 IRQDN2    BPL   IRQDN3         ;+ Branch if no change
C890:BE 86 CF      110            LDX   IRQTBLE,Y      ;+ Get soft switch address
C893:FE 00 C0      111            INC   $C000,X        ;+ Hit the switch. No page cross!!!
C896:88            112 IRQDN3    DEY
C897:30 03   C89C  113            BMI   IRQDN4         ;+ Branch if all done
C899:0A            114            ASL   A              ;Get next bit to check
C89A:D0 F2   C88E  115            BNE   IRQDN2         ;+ Fall through if all done
C89C:0A            116 IRQDN4    ASL   A              ;+ C = 1 if other rom bank
C89D:0A            117            ASL   A              ;+
```

```
C89E:7A             118           PLY
C89F:FA             119           PLX                  ;RESTORE ALL REGISTERS
C8A0:68             120           PLA
C8A1:B0 01   C8A4   121           BCS   IRQDN5         ;+ Which rom bank?
C8A3:40             122           RTI                  ;DO THE REAL RTI!
C8A4:4C 80 C7       123 IRQDN5    JMP   SWRTI          ;+ Go back to the other bank


C8A7:               125 ****************************************
C8A7:               126 *
C8A7:               127 * GOBREAK- If a braek instruction has occurred, we check
C8A7:               128 * if the BRK happened in the alternate rom bank.  If it has,
C8A7:               129 * some fool may have hit the rom switch by accident and the PC is
C8A7:               130 * decremented by two, the main rom is switched in and we resume
C8A7:               131 * where we think he wanted to go
C8A7:               132 *
C8A7:               133 ****************************************
C8A7:        C8A7   134 GOBREAK   EQU   *
C8A7:30 20   C8C9   135           BMI   GBBRK          ;Give up if alt zp
C8A9:89 09          136           BIT   #9             ;From alt rom and no lang card?
C8AB:F0 1C   C8C9   137           BEQ   GBBRK          ;If not then break
C8AD:29 FE          138           AND   #$FE           ;Force main rom
C8AF:48             139           PHA                  ;Save state
C8B0:BA             140           TSX                  ;Save stack pointer
C8B1:68             141           PLA                  ;Skip State
C8B2:68             142           PLA                  ;Skip Y
C8B3:68             143           PLA                  ;Skip X
C8B4:68             144           PLA                  ;Skip A
C8B5:68             145           PLA                  ;Skip P
C8B6:68             146           PLA                  ;> address
C8B7:7A             147           PLY                  ;< address
C8B8:C0 C1          148           CPY   #$C1           ;In the ROM?
C8BA:90 0B   C8C7   149           BCC   GBNOTROM       ;Branch if not
C8BC:E9 02          150           SBC   #2             ;PC = PC - 2
C8BE:B0 01   C8C1   151           BCS   GBNOC
C8C0:88             152           DEY                  ;Borrow from high byte
C8C1:5A             153 GBNOC     PHY                  ;Push new address
C8C2:48             154           PHA
C8C3:9A             155           TXS                  ;Fix stack pointer
C8C4:4C 7F C8       156           JMP   IRQDONE
C8C7:9A             157 GBNOTROM  TXS                  ;Fix stack pointer
C8C8:68             158           PLA                  ;Get state back
C8C9:4C 47 FA       159 GBBRK     JMP   NEWBRK         ;Go do the break
```

```
C8CC:                 162 *   The following routine is for reading key-
C8CC:                 163 * board from buffers or directly.
C8CC:                 164 *   Type-ahead buffering only occurs for non auto-
C8CC:                 165 * repeat keypresses.  When a key is pressed for
C8CC:                 166 * auto-repeat the buffer is first emptied, then the
C8CC:                 167 * repeated characters are returned.
C8CC:                 168 *   The minus flag is used to indicate if a keystroke
C8CC:                 169 * is being returned.
C8CC:                 170 *

C8CC:AD 00 C0         172 XRKBD1    LDA   KBD           ;test keyboard directly
C8CF:10 04   C8D5     173           BPL   XRDKBD        ;loop if buffered since test.
C8D1:8D 10 C0         174           STA   KBDSTRB       ;Clear keyboard strobe.
C8D4:60               175 XNOKEY    RTS                 ;Minus flag indicates valid character

C8D5:20 E6 C8         177 XRDKBD    JSR   XBITKBD       ;is keyboard input ready?
C8D8:10 FA   C8D4     178           BPL   XNOKEY        ;Branch if not.
C8DA:90 F0   C8CC     179           BCC   XRKBD1        ;Branch if direct KBD input.
C8DC:5A               180           PHY                 ;Save Y
C8DD:A0 80            181           LDY   #$80          ;Y=$80 for keyboard buffer
C8DF:20 DF C7         182           JSR   SWGETB        ;Get data from buffer
C8E2:7A               183           PLY
C8E3:09 00            184           ORA   #0            ;Set minus flag
C8E5:60               185           RTS

C8E6:2C FA 05         187 XBITKBD   BIT   TYPHED        ;This routine replaces "BIT KBD"
                                                         instructions
C8E9:10 10   C8FB     188           BPL   XBKB2         ; so as to function with type-ahead.
C8EB:38               189           SEC                 ;anticipate data in buffer is ready
C8EC:08               190           PHP                 ;save carry and minus flags
C8ED:48               191           PHA                 ;preserve accumulator
C8EE:AD FF 06         192           LDA   TRKEY
C8F1:CD FF 05         193           CMP   TWKEY         ;is there data to be read?
C8F4:F0 03   C8F9     194           BEQ   XBKB1         ;branch if type-ahead buffer empty
C8F6:68               195           PLA
C8F7:28               196           PLP
C8F8:60               197           RTS                 ;Carry and minus flag already set.
C8F9:                 198 *
C8F9:68               199 XBKB1     PLA
C8FA:28               200           PLP                 ;restore ACC and Status
C8FB:2C 00 C0         201 XBKB2     BIT   KBD           ;test KBD Directly
C8FE:18               202           CLC                 ;indicate direct test
C8FF:60               203           RTS



C900:                 205 ****************************************
C900:                 206 *
C900:                 207 * PADDLE patch
C900:                 208 * This routine returns the mouse position instead of
C900:                 209 * the paddle if the mouse is on
C900:                 210 *
C900:                 211 ****************************************
C900:       C900      212 mpaddle   equ   *
C900:AD FC 07         213           lda   moumode       ;Is the mouse active?
C903:C9 01            214           cmp   #01           ;Only transparent mode
C905:F0 06   C90D     215           beq   pdon
C907:AD 70 C0         216           lda   vblclr        ;Fire the strobe
C90A:4C 21 FB         217           jmp   $FB21
```

```
C90D:          C90D  218 pdon      equ   *
C90D:E0 01           219           cpx   #1            ;C=1 if X=1
C90F:6A              220           ror   A             ;A=80 or 0
C910:A8              221           tay
C911:B9 7C 05        222           lda   mouxh,y       ;Get high byte
C914:F0 02    C918   223           beq   pdok
C916:A9 FF           224           lda   #$FF
C918:19 7C 04        225 pdok      ora   mouxl,y
C91B:A8              226           tay
C91C:60              227           rts
C91D:                 41           include mini        ;Mini assembler & step routines
```

```
C91D:                    3 ****************************************
C91D:                    4 *
C91D:                    5 * Apple //c Mini Assembler
C91D:                    6 *
C91D:                    7 * Got mnemonic, check address mode
C91D:                    8 *
C91D:                    9 ****************************************
C91D:20 3B CA           10 AMOD1    JSR   NNBL          ;get next non-blank
C920:84 34              11          STY   YSAV          ;save Y
C922:DD BA F9           12          CMP   CHAR1,X
C925:D0 13    C93A      13          BNE   AMOD2
C927:20 3B CA           14          JSR   NNBL          ;get next non-blank
C92A:DD B4 F9           15          CMP   CHAR2,X
C92D:F0 0D    C93C      16          BEQ   AMOD3
C92F:BD B4 F9           17          LDA   CHAR2,X       ;done yet?
C932:F0 07    C93B      18          BEQ   AMOD4
C934:C9 A4              19          CMP   #$A4          ;if "$" then done
C936:F0 03    C93B      20          BEQ   AMOD4
C938:A4 34              21          LDY   YSAV          ;restore Y
C93A:18                 22 AMOD2    CLC
C93B:88                 23 AMOD4    DEY
C93C:26 44              24 AMOD3    ROL   A5L           ;shift bit into format
C93E:E0 03              25          CPX   #$03
C940:D0 0D    C94F      26          BNE   AMOD6
C942:20 A7 FF           27          JSR   GETNUM
C945:A5 3F              28          LDA   A2H           ;get high byte of address
C947:F0 01    C94A      29          BEQ   AMOD5         ;=>
C949:E8                 30          INX
C94A:86 35              31 AMOD5    STX   YSAV1
C94C:A2 03              32          LDX   #$03
C94E:88                 33          DEY
C94F:86 3D              34 AMOD6    STX   A1H
C951:CA                 35          DEX
C952:10 C9    C91D      36          BPL   AMOD1
C954:60                 37          RTS


C955:                   39 *
C955:                   40 *
C955:                   41 * Calculate offset byte for relative addresses
C955:                   42 *
C955:E9 81              43 REL      SBC   #$81          ;calc relative address
C957:4A                 44          LSR   A
C958:D0 14    C96E      45          BNE   GOERR         ;bad branch
C95A:A4 3F              46          LDY   A2H
C95C:A6 3E              47          LDX   A2L
C95E:D0 01    C961      48          BNE   REL1
C960:88                 49          DEY                 ;point to offset
C961:CA                 50 REL1     DEX                  ;displacement - 1
C962:8A                 51          TXA
C963:18                 52          CLC
C964:E5 3A              53          SBC   PCL           ;subtract current PCL
C966:85 3E              54          STA   A2L           ;and save as displacement
C968:10 01    C96B      55          BPL   REL2          ;check page
C96A:C8                 56          INY
C96B:98                 57 REL2     TYA                  ;get page
C96C:E5 3B              58          SBC   PCH           ;check page
```

```
C96E:D0 57   C9C7    59 GOERR     BNE   MINIERR       ;display error
C970:                60 *
C970:                61 * Move instruction to memory
C970:                62 *
C970:A4 2F           63 MOVINST   LDY   LENGTH        ;get instruction length
C972:B9 3D 00        64 MOV1      LDA   A1H,Y         ;get a byte
C975:91 3A           65           STA   (PCL),Y       ;and move it
C977:88              66           DEY
C978:10 F8   C972    67           BPL   MOV1
C97A:                68 *
C97A:                69 * Display instruction
C97A:                70 *
C97A:20 48 F9        71           JSR   PRBLNK        ;print blanks to make ProDOS work
C97D:20 1A FC        72           JSR   UP            ;move up 2 lines
C980:20 1A FC        73           JSR   UP
C983:        C983    74 DISLIN    EQU   *
C983:20 C4 C5        75           JSR   SHOWINST      ;Display line & get next instruction
C986:        C986    76 GETINST1  EQU   *             ;Get the next instruction
C986:A9 A1           77           LDA   #$A1          ;! for prompt
C988:85 33           78           STA   PROMPT
C98A:20 67 FD        79           JSR   GETLNZ        ;Get a line
C98D:80 49   C9D8    80           BRA   DOINST        ;Go do the instruction
C98F:                81 *
C98F:                82 * Compare disassembly of all known opcodes with
C98F:                83 * the one typed in until a match is found
C98F:                84 *
C98F:A5 3D           85 GETOP     LDA   A1H           ;get opcode
C991:20 8E F8        86           JSR   INSDS2        ;determine mnemonic index
C994:AA              87           TAX                  ;X = index
C995:BD 00 FA        88           LDA   MNEMR,X       ;get right half of index
C998:C5 42           89           CMP   A4L           ;does it match entry?
C99A:D0 21   C9BD    90           BNE   NXTOP         ;=>try next opcode
C99C:BD C0 F9        91           LDA   MNEML,X       ;get left half of index

C99F:80 0C   C9AD    93           bra   p1skip        ;Skip past pascal stuff
C9A1:        0009    94           ds    $C9AA-*,0     ;Hello I'm the pascal 1.0 entry point
C9AA:4C B4 C1        95           jmp   p1write       ;Just getting in the way
C9AD:        C9AD    96 p1skip    equ   *

C9AD:C5 43           98           CMP   A4H           ;does it match entry?
C9AF:D0 0C   C9BD    99           BNE   NXTOP         ;=>no, try next opcode
C9B1:A5 44          100           LDA   A5L           ;found opcode, check address mode
C9B3:A4 2E          101           LDY   FORMAT        ;get addr. mode format for that opcode
C9B5:C0 9D          102           CPY   #$9D          ;is it relative?
C9B7:F0 9C   C955   103           BEQ   REL           ;=>yes, calc relative address
C9B9:C5 2E          104           CMP   FORMAT        ;does mode match?
C9BB:F0 B3   C970   105           BEQ   MOVINST       ;=>yes, move instruction to memory
C9BD:C6 3D          106 NXTOP     DEC   A1H           ;else try next opcode
C9BF:D0 CE   C98F   107           BNE   GETOP         ;=>go try it
C9C1:E6 44          108           INC   A5L           ;else try next format
C9C3:C6 35          109           DEC   YSAV1
C9C5:F0 C8   C98F   110           BEQ   GETOP         ;=>go try next format
C9C7:               111 *
C9C7:               112 * Point to the error with a caret, beep, and fall
C9C7:               113 * into the mini-assembler.
C9C7:               114 *
C9C7:A4 34          115 MINIERR   LDY   YSAV          ;get position
C9C9:98             116 ERR2      TYA
```

```
C9CA:AA             117             TAX
C9CB:         C9CB  118 ERR3        EQU   *
C9CB:20 4A F9       119             JSR   PRBL2
C9CE:A9 DE          120             LDA   #$DE          ;^ to point to error
C9D0:20 ED FD       121             JSR   COUT
C9D3:20 3A FF       122             JSR   BELL          ;Beep cause we're mad
C9D6:80 AE   C986   123             BRA   GETINST1      ;try again
C9D8:               124 *
C9D8:               125 * Read a line of input.  If prefaced with " ", decode
C9D8:               126 * mnemonic. If "$" do monitor command.  Otherwise parse
C9D8:               127 * hex address before decoding mnemonic.
C9D8:               128 *
C9D8:20 C7 FF       129 DOINST    JSR   ZMODE         ;clear mode
C9DB:AD 00 02       130           LDA   $200          ;get first char in line
C9DE:C9 A0          131           CMP   #$A0          ;if blank,
C9E0:F0 12   C9F4   132           BEQ   DOLIN         ;=>go attempt disassembly
C9E2:C9 8D          133           CMP   #$8D          ;is it return?
C9E4:D0 01   C9E7   134           BNE   GETI1         ;=>no, continue
C9E6:60             135           RTS                 ;else return to Monitor
C9E7:               136 *
C9E7:20 A7 FF       137 GETI1     JSR   GETNUM        ;parse hexadecimal input
C9EA:C9 93          138           CMP   #$93          ;look for "ADDR:"
C9EC:D0 DB   C9C9   139 GOERR2    BNE   ERR2          ;no ":", display error
C9EE:8A             140           TXA                 ;X nonzero if address entered
C9EF:F0 D8   C9C9   141           BEQ   ERR2          ;no "ADDR", display error
C9F1:               142 *
C9F1:20 78 FE       143           JSR   A1PCLP        ;move address to PC
C9F4:A9 03          144 DOLIN     LDA   #$03          ;get starting opcode
C9F6:85 3D          145           STA   A1H           ;and save
C9F8:20 3B CA       146 NXTCH     JSR   NNBL          ;get next non-blank
C9FB:0A             147           ASL   A             ;validate entry
C9FC:E9 BE          148           SBC   #$BE
C9FE:C9 C2          149           CMP   #$C2
CA00:90 C7   C9C9   150           BCC   ERR2          ;=>flag bad mnemonic
CA02:               151 *
CA02:               152 * Form mnemonic for later comparison
CA02:               153 *
CA02:0A             154           ASL   A
CA03:0A             155           ASL   A
CA04:A2 04          156           LDX   #$04
CA06:0A             157 NXTMN     ASL   A
CA07:26 42          158           ROL   A4L
CA09:26 43          159           ROL   A4H
CA0B:CA             160           DEX
CA0C:10 F8   CA06   161           BPL   NXTMN
CA0E:C6 3D          162           DEC   A1H           ;decrement mnemonic count
CA10:F0 F4   CA06   163           BEQ   NXTMN
CA12:10 E4   C9F8   164           BPL   NXTCH
CA14:A2 05          165           LDX   #$5           ;index into address mode tables
CA16:20 1D C9       166           JSR   AMOD1         ;do this elsewhere
CA19:A5 44          167           LDA   A5L           ;get format
CA1B:0A             168           ASL   A
CA1C:0A             169           ASL   A
CA1D:05 35          170           ORA   YSAV1
CA1F:C9 20          171           CMP   #$20
CA21:B0 06   CA29   172           BCS   AMOD7
CA23:A6 35          173           LDX   YSAV1         ;get our format
CA25:F0 02   CA29   174           BEQ   AMOD7
```

```
CA27:09 80          175           ORA   #$80
CA29:85 44          176 AMOD7     STA   A5L          ;update format
CA2B:84 34          177           STY   YSAV         ;update position
CA2D:B9 00 02       178           LDA   $0200,Y      ;get next character
CA30:C9 BB          179           CMP   #$BB         ;is it a ";"?
CA32:F0 04    CA38  180           BEQ   AMOD8        ;=>yes, skip comment
CA34:C9 8D          181           CMP   #$8D         ;is it carriage return
CA36:D0 B4    C9EC  182           BNE   GOERR2
CA38:4C 8F C9       183 AMOD8     JMP   GETOP        ;get next opcode


CA3B:               185 ****************************************
CA3B:               186 *
CA3B:               187 * NNBL - Gets a non blank character for the mini assembler
CA3B:               188 *
CA3B:               189 ****************************************
CA3B:         CA3B  190 nnbl      equ   *
CA3B:20 B4 C5       191           jsr   getup        ;Get next upshifted character
CA3E:C9 A0          192           cmp   #$A0         ;Blank?
CA40:F0 F9    CA3B  193           beq   nnbl
CA42:60             194           rts
```

```
CA43:                 196 ****************************************
CA43:                 197 *
CA43:                 198 * Step and trace routines
CA43:                 199 *
CA43:                 200 ****************************************
CA43:          CA43   201 step      equ   *
CA43:2C 61 C0         202          bit   butn0          ;Open apple =  slow step
CA46:10 08    CA50    203          bpl   xqnobt0
CA48:A2 07            204          ldx   #7             ;Wait about a second
CA4A:20 A8 FC         205 xqwait   jsr   wait
CA4D:CA               206          dex
CA4E:D0 FA    CA4A    207          bne   xqwait
CA50:2C 62 C0         208 xqnobt0  bit   butn1
CA53:30 51    CAA6    209          bmi   xbrk           ;Closed apple = break
CA55:20 75 FE         210          jsr   a1pc           ;If user specified an address, move it
CA58:18               211          clc
CA59:20 0D CB         212          jsr   godsp          ;Disassemble one instruction
CA5C:68               213          pla                   ;At (PCL,H)
CA5D:85 2C            214          sta   rtnl           ;Adjust to user stack
CA5F:68               215          pla
CA60:85 2D            216          sta   rtnh           ;Save return address
CA62:A2 08            217          ldx   #$08
CA64:BD 04 CB         218 xqinit   lda   initbl-1,x     ;Init XEQ area
CA67:95 3C            219          sta   xqt,x
CA69:CA               220          dex
CA6A:D0 F8    CA64    221          bne   xqinit
CA6C:A1 3A            222          lda   (pcl,x)
CA6E:F0 36    CAA6    223          beq   xbrk           ;Special if break
CA70:A4 2F            224          ldy   length
CA72:C9 20            225          cmp   #$20
CA74:F0 4A    CAC0    226          beq   xjsr           ;Do JSR, RTS, JMP, JMP (), JMP (,X), RTI
CA76:C9 60            227          cmp   #$60
CA78:F0 36    CAB0    228          beq   xrts
CA7A:C9 4C            229          cmp   #$4C
CA7C:F0 4A    CAC8    230          beq   xjmp
CA7E:C9 6C            231          cmp   #$6C
CA80:F0 47    CAC9    232          beq   xjmpat
CA82:C9 7C            233          cmp   #$7C
CA84:F0 5D    CAE3    234          beq   xjmpatx
CA86:C9 40            235          cmp   #$40
CA88:F0 22    CAAC    236          beq   xrti
CA8A:C9 80            237          cmp   #$80           ;Make bra turn into bpl
CA8C:D0 02    CA90    238          bne   xqntbra
CA8E:A9 10            239          lda   #$10
CA90:29 1F            240 xqntbra  and   #$1F
CA92:49 14            241          eor   #$14
CA94:C9 04            242          cmp   #$04
CA96:F0 02    CA9A    243          beq   xq2            ;Copy user inst to xeq area
CA98:B1 3A            244 xq1      lda   (pcl),y        ;Change rel branch
CA9A:99 3C 00         245 xq2      sta   xqt,y          ;displacement to 4 for jmp to branch
CA9D:88               246          dey                   ;or jump to nbranch
CA9E:10 F8    CA98    247          bpl   xq1
CAA0:20 3F FF         248          jsr   restore        ;Restore user reg contents
CAA3:4C 3C 00         249          jmp   xqt            ;Xeq user op from ram
CAA6:A9 64            250 xbrk     lda   #>mon-1        ;Print registers and go to monitor
CAA8:A2 FF            251          ldx   #<mon-1
CAAA:80 2D    CAD9    252          bra   rtnjmp2        ;Display regs & go to monitor
CAAC:18               253 xrti     clc
```

```
CAAD:68            254           pla                    ;Simulate rti by geting status from
                                                           stack
CAAE:85 48         255           sta   status           ;Then doing rts
CAB0:68            256 xrts      pla                    ;Pop PC (not pc - 1)!
CAB1:85 3A         257           sta   pcl
CAB3:68            258           pla
CAB4:85 3B         259 pcinc2    sta   pch              ;Update Pc by 1 (Len = 0)
CAB6:A5 2F         260 pcinc3    lda   length           ;Update pc by length
CAB8:20 56 F9      261           jsr   pcadj3
CABB:84 3B         262           sty   pch
CABD:18            263           clc
CABE:90 11   CAD1  264           bcc   newpcl
CAC0:18            265 xjsr      clc
CAC1:20 54 F9      266           jsr   pcadj2
CAC4:5A            267           phy                    ;Push pc onto stack for jsr
CAC5:48            268           pha
CAC6:A0 02         269           ldy   #$02
CAC8:18            270 xjmp      clc
CAC9:B1 3A         271 xjmpat    lda   (pcl),y
CACB:AA            272           tax                    ;Load pc for jmp, (jmp) simulate
CACC:88            273           dey
CACD:B1 3A         274           lda   (pcl),y
CACF:86 3B         275           stx   pch
CAD1:85 3A         276 newpcl    sta   pcl
CAD3:B0 F3   CAC8  277           bcs   xjmp
CAD5:A6 2D         278 rtnjmp    ldx   rtnh
CAD7:A5 2C         279           lda   rtnl
CAD9:DA            280 rtnjmp2   phx
CADA:48            281           pha
CADB:A9 27         282           lda   #39              ;Move over
CADD:85 24         283           sta   ch
CADF:38            284           sec
CAE0:4C 0D CB      285           jmp   godsp
CAE3:18            286 xjmpatx   clc                    ;JMP (,X)
CAE4:A5 3A         287           lda   pcl              ;Add x to address
CAE6:65 46         288           adc   xreg
CAE8:85 3A         289           sta   pcl
CAEA:90 02   CAEE  290           bcc   xjxnoc
CAEC:E6 3B         291           inc   pch
CAEE:38            292 xjxnoc    sec                    ;C = 1 for indirect jump
CAEF:80 D8   CAC9  293           bra   xjmpat
CAF1:18            294 branch    clc                    ;Branch taken
CAF2:A0 01         295           ldy   #$01             ;Add len+2 to PC
CAF4:B1 3A         296           lda   (pcl),y
CAF6:20 56 F9      297           jsr   pcadj3
CAF9:85 3A         298           sta   pcl
CAFB:98            299           tya
CAFC:38            300           sec
CAFD:B0 B5   CAB4  301           bcs   pcinc2
CAFF:20 4A FF      302 nbrnch    jsr   save             ;Normal return from xeq
CB02:38            303           sec
CB03:B0 B1   CAB6  304           bcs   pcinc3           ;Go update PC

CB05:              306 ****************************************
CB05:              307 *
CB05:              308 * This is the table that is moved into zero page
CB05:              309 * when stepping and tracing
CB05:              310 *
CB05:              311 ****************************************
```

```
CB05:EA            312 initbl    nop
CB06:EA            313           nop
CB07:4C FF CA      314           jmp   nbrnch
CB0A:4C F1 CA      315           jmp   branch


CB0D:              317 ****************************************
CB0D:              318 *
CB0D:              319 * GODSP - Saves hooks, calls display routine and fixes hooks
CB0D:              320 * C = 0 instruction display
CB0D:              321 * C = 1 register display
CB0D:              322 * used by step and trace
CB0D:              323 *
CB0D:              324 ****************************************
CB0D:       CB0D   325 godsp     equ   *
CB0D:A5 36         326           lda   cswl
CB0F:48            327           pha
CB10:A5 37         328           lda   cswh          ;Save output hook
CB12:48            329           pha
CB13:A9 F0         330           lda   #>cout1
CB15:85 36         331           sta   cswl
CB17:A9 FD         332           lda   #<cout1
CB19:85 37         333           sta   cswh
CB1B:B0 05   CB22  334           bcs   godreg        ;Which display?
CB1D:20 D0 F8      335           jsr   instdsp
CB20:80 03   CB25  336           bra   goddone
CB22:20 DA FA      337 godreg    jsr   rgdsp1
CB25:68            338 goddone   pla
CB26:85 37         339           sta   cswh
CB28:68            340           pla
CB29:85 36         341           sta   cswl
CB2B:60            342           rts
CB2C:               42           INCLUDE SCROLLING   ;More Video stuff @$CB30
```

```
CB2C:          0004     3           ds    $CB30-*,0     ;ALign for fools with illegal entry
points
CB30:                   4 *
CB30:                   5 * SCROLLIT scrolls the screen either up or down, depending
CB30:                   6 * on the value of X.  It scrolls within windows with even
CB30:                   7 * or odd edges for both 40 and 80 columns.  It can scroll
CB30:                   8 * windows down to 1 characters wide.
CB30:                   9 *
CB30:DA                 10 SCROLLDN  PHX                 ;save X
CB31:A2 00              11           LDX   #0            ;direction = down
CB33:80 03    CB38      12           BRA   SCROLLIT      ;do scroll
CB35:                   13 *
CB35:DA                 14 SCROLLUP  PHX                 ;save X
CB36:A2 01              15           LDX   #1            ;direction = up
CB38:A4 21              16 SCROLLIT  LDY   WNDWDTH       ;get width of screen window
CB3A:2C 1F C0           17           BIT   RD80VID       ;in 40 or 80 columns?
CB3D:10 18    CB57      18           BPL   GETST         ;=>40, determine starting line
CB3F:8D 01 C0           19           STA   SET80COL      ;make sure this is enabled
CB42:98                 20           TYA                 ;get WNDWDTH for test
CB43:4A                 21           LSR   A             ;divide by 2 for 80 column index
CB44:A8                 22           TAY                 ;and save
CB45:A5 20              23           LDA   WNDLFT        ;test oddity of right edge
CB47:4A                 24           LSR   A             ;by rotating low bit into carry
CB48:B8                 25           CLV                 ;V=0 if left edge even
CB49:90 03    CB4E      26           BCC   CHKRT         ;=>check right edge
CB4B:2C C1 CB           27           BIT   SEV1          ;V=1 if left edge odd
CB4E:2A                 28 CHKRT     ROL   A             ;restore WNDLFT
CB4F:45 21              29           EOR   WNDWDTH       ;get oddity of right edge
CB51:4A                 30           LSR   A             ;C=1 if right edge even
CB52:70 03    CB57      31           BVS   GETST         ;if odd left, don't DEY
CB54:B0 01    CB57      32           BCS   GETST         ;if even right, don't DEY
CB56:88                 33           DEY                 ;if right edge odd, need one less
CB57:8C F8 05           34 GETST     STY   TEMPY         ;save window width
CB5A:AD 1F C0           35           LDA   RD80VID       ;N=1 if 80 columns
CB5D:08                 36           PHP                 ;save N,Z,V
CB5E:A5 22              37           LDA   WNDTOP        ;assume scroll from top
CB60:E0 00              38           CPX   #0            ;up or down?
CB62:D0 03    CB67      39           BNE   SETDBAS       ;=>up
CB64:A5 23              40           LDA   WNDBTM        ;down, start scrolling at bottom
CB66:3A                 41           DEC   A             ;really need one less
CB67:                   42 *
CB67:8D 78 05           43 SETDBAS   STA   TEMPA         ;save current line
CB6A:20 24 FC           44           JSR   VTABZ         ;calculate base with window width
CB6D:                   45 *
CB6D:A5 28              46 SCRLIN    LDA   BASL          ;current line is destination
CB6F:85 2A              47           STA   BAS2L
CB71:A5 29              48           LDA   BASH
CB73:85 2B              49           STA   BAS2H
CB75:                   50 *
CB75:AD 78 05           51           LDA   TEMPA         ;get current line
CB78:E0 00              52           CPX   #0            ;going up?
CB7A:D0 07    CB83      53           BNE   SETUP2        =>up, inc current line
CB7C:C5 22              54           CMP   WNDTOP        ;down. Reached top yet?
CB7E:F0 39    CBB9      55           BEQ   SCRL3         ;yes! clear top line, exit
CB80:3A                 56           DEC   A             ;no, go up a line
CB81:80 05    CB88      57           BRA   SETSRC        ;set source for scroll
CB83:1A                 58 SETUP2    INC   A             ;up, inc current line
CB84:C5 23              59           CMP   WNDBTM        ;at bottom yet?
CB86:B0 31    CBB9      60           BCS   SCRL3         ;yes! clear bottom line, exit
```

```
CB88:                   61 *
CB88:8D 78 05           62 SETSRC    STA   TEMPA          ;save new current line
CB8B:20 24 FC           63           JSR   VTABZ          ;get base for new current line
CB8E:AC F8 05           64           LDY   TEMPY          ;get width for scroll
CB91:28                 65           PLP                  ;get status for scroll
CB92:08                 66           PHP                  ;N=1 if 80 columns
CB93:10 1F    CBB4      67           BPL   SKPRT          ;=>only do 40 columns
CB95:AD 55 C0           68           LDA   TXTPAGE2       ;scroll aux page first (even bytes)
CB98:98                 69           TYA                  ;test Y
CB99:F0 07    CBA2      70           BEQ   SCRLFT         ;if Y=0, only scroll one byte
CB9B:B1 28              71 SCRLEVEN  LDA   (BASL),Y
CB9D:91 2A              72           STA   (BAS2L),Y
CB9F:88                 73           DEY
CBA0:D0 F9    CB9B      74           BNE   SCRLEVEN       ;do all but last even byte
CBA2:70 04    CBA8      75 SCRLFT    BVS   SKPLFT         ;odd left edge, skip this byte
CBA4:B1 28              76           LDA   (BASL),Y
CBA6:91 2A              77           STA   (BAS2L),Y
CBA8:AD 54 C0           78 SKPLFT    LDA   TXTPAGE1       ;now do main page (odd bytes)
CBAB:AC F8 05           79           LDY   TEMPY          ;restore width
CBAE:B0 04    CBB4      80           BCS   SKPRT          ;even right edge, skip this byte
CBB0:B1 28              81 SCRLODD   LDA   (BASL),Y
CBB2:91 2A              82           STA   (BAS2L),Y
CBB4:88                 83 SKPRT     DEY
CBB5:10 F9    CBB0      84           BPL   SCRLODD
CBB7:80 B4    CB6D      85           BRA   SCRLIN         ;scroll next line
CBB9:                   86 *
CBB9:20 A0 FC           87 SCRL3     JSR   CLRLIN         ;clear current line
CBBC:20 22 FC           88           JSR   VTAB           ;restore original cursor line
CBBF:28                 89           PLP                  ;pull status off stack
CBC0:FA                 90           PLX                  ;restore X
CBC1:60                 91 SEV1      RTS                  ;done!!!
```

```
CBC2:                 93 *
CBC2:                 94 * DOCLR is called by CLREOL.  It decides whether
CBC2:                 95 * to do a (quick) 40 or 80 column clear to end of line.
CBC2:                 96 *
CBC2:2C 1F C0         97 DOCLR      BIT   RD80VID       ;40 or 80 column clear?
CBC5:30 13    CBDA    98            BMI   CLR80        ;=>clear 80 columns
CBC7:91 28            99 CLR40      STA   (BASL),Y
CBC9:C8              100            INY
CBCA:C4 21           101            CPY   WNDWDTH
CBCC:90 F9    CBC7   102            BCC   CLR40
CBCE:60              103            RTS
CBCF:                104 *
CBCF:DA              105 CLRHALF    PHX                 ;clear right half of screen
CBD0:A2 D8           106            LDX   #$D8          ;for SCRN48
CBD2:A0 14           107            LDY   #20
CBD4:A5 32           108            LDA   INVFLG
CBD6:29 A0           109            AND   #$A0
CBD8:80 17    CBF1   110            BRA   CLR2          ;=>jump into middle
CBDA:                111 *
CBDA:DA              112 CLR80      PHX                 ;preserve X
CBDB:48              113            PHA                 ;and blank
CBDC:98              114            TYA                 ;get count for CH
CBDD:48              115            PHA                 ;save for left edge check
CBDE:38              116            SEC                 ;count=WNDWDTH-Y-1
CBDF:E5 21           117            SBC   WNDWDTH
CBE1:AA              118            TAX                 ;save CH counter
CBE2:98              119            TYA                 ;div CH by 2 for half pages
CBE3:4A              120            LSR   A
CBE4:A8              121            TAY
CBE5:68              122            PLA                  ;restore original CH
CBE6:45 20           123            EOR   WNDLFT        ;get starting page
CBE8:6A              124            ROR   A
CBE9:B0 03    CBEE   125            BCS   CLR0
CBEB:10 01    CBEE   126            BPL   CLR0
CBED:C8              127            INY                 ;iff WNDLFT odd, starting byte odd
CBEE:68              128 CLR0       PLA                 ;get blankity blank
CBEF:B0 0B    CBFC   129            BCS   CLR1          ;starting page is 1 (default)
CBF1:2C 55 C0        130 CLR2       BIT   TXTPAGE2      ;else do page 2
CBF4:91 28           131            STA   (BASL),Y
CBF6:2C 54 C0        132            BIT   TXTPAGE1      ;now do page 1
CBF9:E8              133            INX
CBFA:F0 06    CC02   134            BEQ   CLR3          ;all done
CBFC:91 28           135 CLR1       STA   (BASL),Y
CBFE:C8              136            INY                 ;forward 2 columns
CBFF:E8              137            INX                 ;next CH
CC00:D0 EF    CBF1   138            BNE   CLR2          ;not done yet
CC02:FA              139 CLR3       PLX                 ;restore X
CC03:60              140            RTS                 ;and exit
CC04:                141 *
CC04:9C FA 05        142 CLRPORT    STZ   TYPHED        ;disable typeahead
CC07:9C F9 05        143            STZ   EXTINT2       ;and external interrupts
CC0A:60              144            RTS
```

```
CC0B:                 146 *
CC0B:                 147 * PASINVERT is used by Pascal to display the cursor. Pascal
CC0B:                 148 * normally leaves the cursor on the screen at all times. It
CC0B:                 149 * is fleetingly removed while a character is displayed, then
CC0B:                 150 * promptly redisplayed.  CTL-F and CTL-E, respectively,
CC0B:                 151 * disable and enable display of the cursor when printed using
CC0B:                 152 * the Pascal 1.1 entry point (PWRITE).  Screen I/O is
CC0B:                 153 * significantly faster when the cursor is disabled.  This
CC0B:                 154 * feature is supported by Pascal 1.2 and later.
CC0B:                 155 *
CC0B:AD FB 04         156 PASINVERT LDA   VMODE          ;Called by pascal to
CC0E:29 10            157           AND   #M.CURSOR      ;display cursor
CC10:D0 0A    CC1C    158           BNE   INVX           ;=>cursor off, don't invert
CC12:         CC12    159 INVERT    EQU   *
CC12:20 1D CC         160           JSR   PICKY          ;load Y and get char
CC15:48               161           PHA
CC16:49 80            162           EOR   #$80           ;FLIP INVERSE/NORMAL
CC18:20 B3 C3         163           JSR   STORY          ;stuff onto screen
CC1B:68               164           PLA                  ;for RDCHAR
CC1C:60               165 INVX      RTS
CC1D:                 166 *
CC1D:                 167 * PICK lifts a character from the screen in either
CC1D:                 168 * 40 or 80 columns from the current cursor position.
CC1D:                 169 * If the alternate character set is switched in,
CC1D:                 170 * character codes $0-$1F are returned as $40-$5F (which
CC1D:                 171 * is what must have been originally printed to the location).
CC1D:                 172 *
CC1D:5A               173 PICKY     PHY                  ;save Y
CC1E:20 9D CC         174           JSR   GETCUR         ;get newest cursor into Y
CC21:AD 1F C0         175           LDA   RD80VID        ;80 columns?
CC24:10 17    CC3D    176           BPL   PICK1          ;=>no
CC26:8D 01 C0         177           STA   SET80COL       ;force 80STORE if 80 columns
CC29:98               178           TYA
CC2A:45 20            179           EOR   WNDLFT         ;C=1 if char in main RAM
CC2C:6A               180           ROR   A              ;get low bit into carry
CC2D:B0 04    CC33    181           BCS   PICK2          ;=>store in main memory
CC2F:AD 55 C0         182           LDA   TXTPAGE2       ;else switch in page 2
CC32:C8               183           INY                  ;for odd left, aux bytes
CC33:98               184 PICK2     TYA                  ;divide pos'n by 2
CC34:4A               185           LSR   A
CC35:A8               186           TAY                  ;and use as offset into line
CC36:B1 28            187           LDA   (BASL),Y       ;pick character
CC38:8D 54 C0         188           STA   TXTPAGE1       ;80 columns, switch in
CC3B:80 02    CC3F    189           BRA   PICK3          ;skip 40 column pick
CC3D:B1 28            190 PICK1     LDA   (BASL),Y       ;pick 40 column char
CC3F:2C 1E C0         191 PICK3     BIT   ALTCHARSET     ;only allow if alt set
CC42:10 06    CC4A    192           BPL   PICK4
CC44:C9 20            193           CMP   #$20
CC46:B0 02    CC4A    194           BCS   PICK4
CC48:09 40            195           ORA   #$40
CC4A:7A               196 PICK4     PLY                  ;restore real Y
CC4B:60               197           RTS
CC4C:                 198 *
CC4C:                 199 * SHOWCUR displays either a checkerboard cursor, a solid
CC4C:                 200 * rectangle, or the current cursor character, depending
CC4C:                 201 * on the value of the CURSOR location.  0=inverse cursor,
CC4C:                 202 * $FF=checkerboard cursor, anything else is displayed
CC4C:                 203 * after being anded with inverse mask.
```

```
CC4C:                  204 *
CC4C:AC FB 07          205 SHOWCUR   LDY   CURSOR        ;what's my type?
CC4F:D0 02    CC53     206           BNE   NOTINV        ;=>not inverse
CC51:80 BF    CC12     207           BRA   INVERT        ;else invert the char (exit)
CC53:                  208 *
CC53:                  209 * Exit with char in accumulator
CC53:                  210 *
CC53:20 1D CC          211 NOTINV    JSR   PICKY         ;get char on screen
CC56:48                212           PHA                ;preserve it
CC57:8D 7B 07          213           STA   NXTCUR       ;save for update
CC5A:98                214           TYA                ;test for checkerboard
CC5B:C8                215           INY
CC5C:F0 0D    CC6B     216           BEQ   NOTINV2      ;=>checkerboard, display it
CC5E:7A                217           PLY                ;test char
CC5F:5A                218           PHY
CC60:30 09    CC6B     219           BMI   NOTINV2      ;don't need inverse
CC62:AD 1E C0          220           LDA   ALTCHARSET   ;mask = $7F if alternate
CC65:09 7F             221           ORA   #$7F         ; character set,
CC67:4A                222           LSR   A            ;$3F if normal char set
CC68:2D FB 07          223 NOTINV1   AND   CURSOR       ;form char to display
CC6B:20 B3 C3          224 NOTINV2   JSR   STORY        ;and display it
CC6E:68                225           PLA                ;restore real char
CC6F:60                226           RTS
CC70:                  227 *
CC70:                  228 * The UPDATE routine increments the random seed.
CC70:                  229 * If a certain value is reached and we are in Apple II
CC70:                  230 * mode, the blinking check cursor is updated.  If a
CC70:                  231 * key has been pressed, the old char is replaced on the
CC70:                  232 * screen, and we return with BMI.
CC70:                  233 *
CC70:                  234 * NOTE: this routine used by COMM firmware!!
CC70:                  235 *
CC70:48                236 UPDATE     PHA                ;save char
CC71:E6 4E             237           INC   RNDL         ;update seed
CC73:D0 1E    CC93     238           BNE   UD2          ;check for key
CC75:A5 4F             239           LDA   RNDH
CC77:E6 4F             240           INC   RNDH
CC79:45 4F             241           EOR   RNDH
CC7B:29 10             242           AND   #$10         ;need to update cursor?
CC7D:F0 14    CC93     243           BEQ   UD2          ;=>no, check for key
CC7F:AD FB 07          244           LDA   CURSOR       ;what cursor are we using?
CC82:F0 0F    CC93     245           BEQ   UD2          ;=>//e cursor, leave alone
CC84:5A                246           PHY                ;+ Save Y
CC85:20 1D CC          247           JSR   PICKY        ;get the character into A
CC88:AC 7B 07          248           LDY   NXTCUR       ;get next character
CC8B:8D 7B 07          249           STA   NXTCUR       ;save next next character
CC8E:98                250           TYA
CC8F:20 B3 C3          251           JSR   STORY        ;and print it
CC92:7A                252           PLY                ;+
CC93:68                253 UD2       PLA                ;get real char
CC94:20 E6 C8          254           JSR   XBITKBD      ;was a key pressed?
CC97:10 26    CCBF     255           BPL   GETCURX      ;=>no key pressed
CC99:4C BD CF          256 CLRKBD    JMP   CLRKBD2      ;+ restore old key look for key and exit
CC9C:EA                257           NOP                ;+ Keep code alignedkey
CC9D:                  258 *
CC9D:                  259 * ON CURSORS.  Whenever the horizontal cursor position is
CC9D:                  260 * needed, a call to GETCUR is done. This is the equivalent
CC9D:                  261 * of a LDY CH.  This returns the current cursor for II and
```

```
CC9D:                 262 * //e mode, which may have been poked as either CH or OURCH.
CC9D:                 263 *
CC9D:                 264 * It also forces CH and OLDCH to 0 if 80 column mode active.
CC9D:                 265 * This prevents LDY CH, STA (BASL),Y from trashing non screen
CC9D:                 266 * memory. It works just like the //e.
CC9D:                 267 *
CC9D:                 268 * All routines that update the cursor's horizontal position
CC9D:                 269 * are here.  This ensures that the newest value of the cursor
CC9D:                 270 * is always used, and that 80 column CH is always 0.
CC9D:                 271 *
CC9D:                 272 * GETCUR only affects the Y register
CC9D:                 273 *
CC9D:A4 24            274 GETCUR    LDY   CH            ;if CH=OLDCH, then
CC9F:CC 7B 04         275           CPY   OLDCH         ;OURCH is valid
CCA2:D0 03   CCA7     276           BNE   GETCUR1       ;=>else CH must have been changed
CCA4:AC 7B 05         277           LDY   OURCH         ;use OURCH
CCA7:C4 21            278 GETCUR1   CPY   WNDWDTH       ;is the value too big
CCA9:90 02   CCAD     279           BCC   GETCUR2       ;=>no, fits just fine
CCAB:A0 00            280           LDY   #0            ;else force CH to 0
CCAD:                 281 *
CCAD:                 282 * GETCUR2 is commonly used to set the current cursor
CCAD:                 283 * position when Y can be used.
CCAD:                 284 *
CCAD:8C 7B 05         285 GETCUR2   STY   OURCH         ;update real cursor
CCB0:2C 1F C0         286           BIT   RD80VID       ;80 columns?
CCB3:10 02   CCB7     287           BPL   GETCUR3       ;=>no, set all cursors
CCB5:A0 00            288           LDY   #0            ;yes, peg CH to 0
CCB7:84 24            289 GETCUR3   STY   CH
CCB9:8C 7B 04         290           STY   OLDCH
CCBC:AC 7B 05         291           LDY   OURCH         ;get cursor
CCBF:60               292 GETCURX   RTS                 ;and fly...
CCC0:                  43           INCLUDE ESCAPE
```

```
CCC0:                 2 * START AN ESCAPE SEQUENCE:
CCC0:                 3 *  WE HANDLE THE FOLLOWING ONES:
CCC0:                 4 *     @ - HOME & CLEAR
CCC0:                 5 *     A - Cursor right
CCC0:                 6 *     B - Cursor left
CCC0:                 7 *     C - Cursor down
CCC0:                 8 *     D - Cursor up
CCC0:                 9 *     E - CLR TO EOL
CCC0:                10 *     F - CLR TO EOS
CCC0:                11 *     I, Up Arrow - CURSOR UP (stay escape)
CCC0:                12 *     J, Lft Arrow - CURSOR LEFT (stay escape)
CCC0:                13 *     K, Rt Arrow - CURSOR RIGHT (stay escape)
CCC0:                14 *     M, Dn Arrow - CURSOR DOWN  (stay escape)
CCC0:                15 *     4 - GOTO 40 COLUMN MODE
CCC0:                16 *     8 - GOTO 80 COLUMN MODE
CCC0:                17 * CTL-D- Disable the printing of control chars
CCC0:                18 * CTL-E- Enable the printing of control chars
CCC0:                19 * CTL-Q- QUIT (PR#0/IN#0)
CCC0:                20 *
CCC0:B9 0C CD        21 ESC3      LDA   ESCCHAR,Y      ;GET CHAR TO "PRINT"
CCC3:5A              22           PHY                  ;save index
CCC4:20 58 CD        23           JSR   CTLCHAR        ;execute character
CCC7:7A              24           PLY                  ;restore index
CCC8:C0 08           25           CPY   #YHI           ;If Y<YHI, stay escape
CCCA:B0 21   CCED    26           BCS   ESCRDKEY       ;=>exit escape mode
CCCC:                27 *
CCCC:                28 * This is the entry point called by RDKEY iff escapes
CCCC:                29 * are enabled and an escape is encountered.  The next
CCCC:                30 * keypress is read and processed.  If it is a key that
CCCC:                31 * terminates escape mode, a new key is read by ESCRDKEY.
CCCC:                32 * If escape mode should not be terminated, NEWESC is
CCCC:                33 * called again.
CCCC:                34 *
CCCC:20 1D CC        35 NEWESC    JSR   PICKY          ;get current character
CCCF:48              36           PHA                  ;and save it
CCD0:29 80           37           AND   #$80           ;save invert bit
CCD2:49 AB           38           EOR   #$AB           ;make it inverted "+"
CCD4:20 B3 C3        39           JSR   STORY          ;and pop it on the screen
CCD7:20 E6 C8        40 ESC0      JSR   XBITKBD        ;check for keystroke
CCDA:10 FB   CCD7    41           BPL   ESC0
CCDC:68              42           PLA                  ;get old char
CCDD:20 99 CC        43           JSR   CLRKBD         ;restore char, get key
CCE0:20 9B C3        44           JSR   UPSHIFT        ;upshift esc char
CCE3:A0 13           45 ESC1      LDY   #ESCNUM        ;COUNT/INDEX
CCE5:D9 F8 CC        46 ESC2      CMP   ESCTAB,Y       ;IS IT A VALID ESCAPE?
CCE8:F0 D6   CCC0    47           BEQ   ESC3           =>yes
CCEA:88              48           DEY
CCEB:10 F8   CCE5    49           BPL   ESC2           ;TRY 'EM ALL...
CCED:                50 *
CCED:                51 * End of escape sequence, read next character.
CCED:                52 * This is initially called by RDCHAR which is usually called
CCED:                53 * by GETLN to read characters with escapes enabled.
CCED:                54 *
CCED:A9 08           55 ESCRDKEY  LDA   #M.CTL         ;enable escape sequences
CCEF:1C FB 04        56           TRB   VMODE
CCF2:20 0C FD        57           JSR   RDKEY          ;read char with escapes
CCF5:4C 44 FD        58           JMP   NOESCAPE       ;got the key, disable escapes
CCF8:                59 *
```

```
CCF8:                60 * When in escape mode, the characters in ESCTAB (high)
CCF8:                61 * bits set), are mapped into the characters in ESCCHAR.
CCF8:                62 * These characters are then executed by a call to CTLCHAR.
CCF8:                63 *
CCF8:                64 * CTLCHAR looks up a character in the table starting at
CCF8:                65 * CTLTAB.  It uses the current index as an index into the
CCF8:                66 * table of routine addresses, CTLADR.  If the character is
CCF8:                67 * not in the table, a call to VIDOUT1 is done in case the
CCF8:                68 * character is BS, LF, CR, or BEL.
CCF8:                69 *
CCF8:                70 * NOTE: CTLON and CTLOFF are not accessible except through
CCF8:                71 * and escape sequence
CCF8:                72 *
CCF8:                73           MSB   ON             ;high bit on
CCF8:        CCF8    74 ESCTAB    EQU   *
CCF8:CA              75           ASC   'J'            ;left (stay esc)
CCF9:88              76           DFB   $88            ;left arrow (stay esc)
CCFA:CD              77           ASC   'M'            ;down (stay esc)
CCFB:8B              78           DFB   $8B            ;up arrow (stay esc)
CCFC:95              79           DFB   $95            ;right arrow (stay esc)
CCFD:8A              80           DFB   $8A            ;down arrow (stay esc)
CCFE:C9              81           ASC   'I'            ;up (stay esc)
CCFF:CB              82           ASC   'K'            ;right (stay esc)
CD00:        0008    83 YHI       EQU   *-ESCTAB
CD00:C2              84           ASC   'B'            ;left
CD01:C3              85           ASC   'C'            ;down
CD02:C4              86           ASC   'D'            ;up
CD03:C1              87           ASC   'A'            ;right
CD04:C0              88           ASC   '@'            ;formfeed
CD05:C5              89           ASC   'E'            ;clear EOL
CD06:C6              90           ASC   'F'            ;clear EOS
CD07:B4              91           ASC   '4'            ;40 column mode
CD08:B8              92           ASC   '8'            ;80 column mode
CD09:91              93           DFB   $91            ;CTL-Q = QUIT
CD0A:84              94           DFB   $84            ;CTL-D ;ctl char disable
CD0B:85              95           DFB   $85            ;CTL-E ;ctl char enable
CD0C:                96 *
CD0C:        0013    97 ESCNUM    EQU   *-ESCTAB-1
CD0C:                98 *
CD0C:        CD0C    99 ESCCHAR   EQU   *              ;list of escape chars
CD0C:88             100           DFB   $88            ;J: BS (stay esc)
CD0D:88             101           DFB   $88            ;<-:BS (stay esc)
CD0E:8A             102           DFB   $8A            ;M: LF (stay esc)
CD0F:9F             103           DFB   $9F            ;UP:US (stay esc)
CD10:9C             104           DFB   $9C            ;->:FS (stay esc)
CD11:8A             105           DFB   $8A            ;DN: LF (stay esc)
CD12:9F             106           DFB   $9F            ;I: UP (stay esc)
CD13:9C             107           DFB   $9C            ;K: RT (stay esc)
CD14:88             108           DFB   $88            ;ESC-B = BS
CD15:        CD15   109 CTLTAB    EQU   *              ;list of control characters
CD15:8A             110           DFB   $8A            ;ESC-C = DN
CD16:9F             111           DFB   $9F            ;ESC-D = UP
CD17:9C             112           DFB   $9C            ;ESC-A = RT
CD18:8C             113           DFB   $8C            ;@: Formfeed
CD19:9D             114           DFB   $9D            ;E: CLREOL
CD1A:8B             115           DFB   $8B            ;F: CLREOP
CD1B:91             116           DFB   $91            ;SET40
CD1C:92             117           DFB   $92            ;SET80
```

```
CD1D:95                 118             DFB   $95             ;QUIT
CD1E:04                 119             DFB   $04             ;Disable controls (escape only)
CD1F:05                 120             DFB   $05             ;Enable controls (escape only)
CD20:                   121 * escape chars end here
CD20:85                 122             DFB   $85             ;X.CUR.ON
CD21:86                 123             DFB   $86             ;X.CUR.OFF
CD22:8E                 124             DFB   $8E             ;Normal
CD23:8F                 125             DFB   $8F             ;Inverse
CD24:96                 126             DFB   $96             ;Scroll down
CD25:97                 127             DFB   $97             ;Scroll up
CD26:98                 128             DFB   $98             ;mouse chars off
CD27:99                 129             DFB   $99             ;home cursor
CD28:9A                 130             DFB   $9A             ;clear line
CD29:9B                 131             DFB   $9B             ;mouse chars on
CD2A:                   132 *
CD2A:          0014     133 CTLNUM      EQU   *-CTLTAB-1
CD2A:                   134 *
CD2A:          CD2A     135 CTLADR      EQU   *
CD2A:66 FC              136             DW    LF              ;move cursor down
CD2C:1A FC              137             DW    UP              ;move cursor up
CD2E:A0 FB              138             DW    NEWADV          ;forward a space
CD30:58 FC              139             DW    HOME            ;home cursor, clear screen
CD32:9C FC              140             DW    CLREOL          ;clear to end of line
CD34:42 FC              141             DW    CLREOP          ;clear to end of page
CD36:C0 CD              142             DW    SET40           ;set 40 column mode
CD38:BE CD              143             DW    SET80           ;set 80 column mode
CD3A:45 CE              144             DW    QUIT            ;Quit video firmware
CD3C:91 CD              145             DW    CTLOFF          ;disable //e control chars
CD3E:95 CD              146             DW    CTLON           ;enable //e control chars
CD40:89 CD              147             DW    X.CUR.ON        ;turn on cursor (pascal)
CD42:8D CD              148             DW    X.CUR.OFF       ;turn off cursor (pascal)
CD44:B0 CD              149             DW    X.SO            ;normal video
CD46:B7 CD              150             DW    X.SI            ;inverse video
CD48:30 CB              151             DW    SCROLLDN        ;scroll down a line
CD4A:35 CB              152             DW    SCROLLUP        ;scroll up a line
CD4C:9F CD              153             DW    MOUSOFF         ;disable mouse characters
CD4E:A5 CD              154             DW    HOMECUR         ;move cursor home
CD50:A0 FC              155             DW    CLRLIN          ;clear current line
CD52:99 CD              156             DW    MOUSON          ;enable mouse characters
CD54:                   157 *
CD54:                   158             MSB   ON
CD54:                   159 *
CD54:                   160 * CTLCHAR executes the control character in the
CD54:                   161 * accumulator.  If it is called by Pascal, the character
CD54:                   162 * is always executed.  If it is called by the video
CD54:                   163 * firmware, the character is executed if M.CTL is set
CD54:                   164 * and M.CTL2 is clear.
CD54:                   165 *
CD54:                   166 * Note: This routine is only called if the video firmware
CD54:                   167 * is active. The Monitor ROM calls VIDOUT1 if the video
CD54:                   168 * firmware is inactive.
CD54:                   169 *
CD54:2C C1 CB           170 CTLCHAR0    BIT   SEV1            ;set V (use M.CTL)
CD57:50                 171             DFB   $50             ;BVC opcode (never taken)
CD58:                   172 *
CD58:B8                 173 CTLCHAR     CLV                   ;Always do control character
CD59:DA                 174             PHX                   ;save X
CD5A:8D F8 04           175             STA   TEMP1           ;temp save of A
```

```
CD5D:20 04 FC      176           JSR   VIDOUT1       ;try to execute CR, LF, BS, or BEL
CD60:CD F8 04      177           CMP   TEMP1         ;if acc has changed
CD63:D0 0A   CD6F  178           BNE   CTLDONE       ;then function done
CD65:A2 14         179           LDX   #CTLNUM       ;number of CTL chars
CD67:DD 15 CD      180 FNDCTL    CMP   CTLTAB,X      ;is it in table
CD6A:F0 05   CD71  181           BEQ   CTLGO         ;=>yes, should we execute?
CD6C:CA            182           DEX                 ;else check next
CD6D:10 F8   CD67  183           BPL   FNDCTL        ;=>try next one
CD6F:FA            184 CTLDONE   PLX                 ;restore X
CD70:60            185           RTS                 ;and return
CD71:              186 *
CD71:48            187 CTLGO     PHA                 ;save A
CD72:50 0C   CD80  188           BVC   CTLGO1        ;V clear, always do (pascal,escape)
CD74:AD FB 04      189           LDA   VMODE         ;controls are enabled iff
CD77:29 28         190           AND   #M.CTL+M.CTL2 ; M.CTL = 1 and
CD79:49 08         191           EOR   #M.CTL        ; M.CTL2 = 0
CD7B:F0 03   CD80  192           BEQ   CTLGO1        ;=>they're enabled!!
CD7D:68            193 CGO       PLA                 ;restore A
CD7E:FA            194           PLX                 ;restore X
CD7F:60            195           RTS                 ;and return
CD80:              196 *
CD80:8A            197 CTLGO1    TXA                 ;double X as index
CD81:0A            198           ASL   A             ;into address table
CD82:AA            199           TAX
CD83:68            200           PLA                 ;restore A
CD84:20 A4 FC      201           JSR   CTLDO         ;execute the char
CD87:FA            202           PLX                 ;restore X
CD88:60            203           RTS                 ;and return
CD89:              204 *
CD89:              205 * X.CUR.ON = Allow Pascal cursor display
CD89:              206 * X.CUR.OFF = Disable Pascal cursor display
CD89:              207 * Cursor is not displayed during call, so it will
CD89:              208 * be right when "redisplayed".
CD89:              209 * Note: Though these commands are executed from BASIC,
CD89:              210 * they have no effect on firmware operation.
CD89:              211 *
CD89:A9 10         212 X.CUR.ON  LDA   #M.CURSOR     ;clear cursor bit
CD8B:80 0E   CD9B  213           BRA   CLRIT
CD8D:              214 *
CD8D:A9 10         215 X.CUR.OFF LDA   #M.CURSOR     ;set cursor bit
CD8F:80 10   CDA1  216           BRA   SETIT
CD91:              217 *
CD91:              218 * The control characters other than CR,LF,BEL,BS
CD91:              219 * are normally enabled when video firmware is active.
CD91:              220 * They can be disabled and enabled using the ESC-D
CD91:              221 * and ESC-E escape sequences.
CD91:              222 *
CD91:A9 20         223 CTLOFF    LDA   #M.CTL2       ;disable control characters
CD93:80 0C   CDA1  224           BRA   SETIT         ;by setting M.CTL2
CD95:              225 *
CD95:A9 20         226 CTLON     LDA   #M.CTL2       ;enable control characters
CD97:80 02   CD9B  227           BRA   CLRIT         ;by clearing M.CTL2
CD99:              228 *
CD99:              229 * Enable mouse text by clearing M.MOUSE
CD99:              230 *
CD99:A9 01         231 MOUSON    LDA   #M.MOUSE
CD9B:1C FB 04      232 CLRIT     TRB   VMODE
CD9E:60            233           RTS
```

```
CD9F:                   234 *
CD9F:                   235 * Disable mouse text by setting M.MOUSE
CD9F:                   236 *
CD9F:A9 01              237 MOUSOFF   LDA   #M.MOUSE
CDA1:0C FB 04           238 SETIT     TSB   VMODE
CDA4:60                 239           RTS
CDA5:                   240 *
CDA5:                   241 * EXECUTE HOME:
CDA5:                   242 *
CDA5:20 E9 FE           243 HOMECUR   JSR   CLRCH         ;move cursors to far left
CDA8:A8                 244           TAY                 ;(probably not needed)
CDA9:A5 22              245           LDA   WNDTOP        ;and to top of window
CDAB:85 25              246           STA   CV
CDAD:4C 88 FC           247           JMP   NEWVTABZ      ;then set base address, OURCV
CDB0:                   248 *
CDB0:                   249 * EXECUTE "NORMAL VIDEO"
CDB0:                   250 *
CDB0:20 84 FE           251 X.SO      JSR   SETNORM       ;set INVFLG to $FF
CDB3:A9 04              252           LDA   #M.VMODE      ;then clear inverse mode bit
CDB5:80 E4    CD9B      253           BRA   CLRIT
CDB7:                   254 *
CDB7:                   255 * EXECUTE "INVERSE VIDEO"
CDB7:                   256 *
CDB7:20 80 FE           257 X.SI      JSR   SETINV        ;set INVFLG to $3F
CDBA:A9 04              258           LDA   #M.VMODE      ;then set inverse mode bit
CDBC:80 E3    CDA1      259           BRA   SETIT
CDBE:                   260 *
CDBE:                   261 * EXECUTE '40COL MODE' or '80COL MODE':
CDBE:                   262 *
CDBE:38                 263 SET80     SEC                 ;flag an 80 column window
CDBF:90                 264           DFB   $90           ;BCC opcode (never taken)
CDC0:18                 265 SET40     CLC                 ;flag a 40 column window
CDC1:2C FB 04           266           BIT   VMODE         ;but...is it pascal?
CDC4:10 54    CE1A      267           BPL   SETX          ;=>yes, don't execute
CDC6:08                 268           PHP                 ;save window size
CDC7:20 1B CE           269           JSR   HOOKITUP      ;COPYROM if needed, set I/O hooks
CDCA:28                 270           PLP                 ;and get 40/80
CDCB:80 08    CDD5      271           BRA   WIN0          ;=>set window
CDCD:                   272 *
CDCD:                   273 * CHK80 is called by PR#0 to convert to 40 if it was
CDCD:                   274 * 80. Otherwise the window is left ajar.
CDCD:                   275 *
CDCD:2C 1F C0           276 CHK80     BIT   RD80VID       ;don't set 40 if
CDD0:10 48    CE1A      277           BPL   SETX          ;already 40
CDD2:                   278 *
CDD2:18                 279 WIN40     CLC                 ;flag 40 column window
CDD3:B0                 280           DFB   $B0           ;BCS opcode (never taken)
CDD4:38                 281 WIN80     SEC                 ;flag 80 column window
CDD5:64 22              282 WIN0      STZ   WNDTOP        ;set window top now
CDD7:2C 1A C0           283           BIT   RDTEXT        ;for text or mixed
CDDA:30 04    CDE0      284           BMI   WIN1          ;=>text
CDDC:A9 14              285           LDA   #20
CDDE:85 22              286           STA   WNDTOP        ;used by 80<->40 conversion
CDE0:2C 1F C0           287 WIN1      BIT   RD80VID       ;80 columns now?
CDE3:08                 288           PHP                 ;save 80 or 40
CDE4:B0 07    CDED      289           BCS   WIN2          ;=>80: convert if 40
CDE6:10 0A    CDF2      290           BPL   WIN3          ;=>40: no convert
CDE8:20 53 CE           291           JSR   SCRN84        ;80: convert to 40
```

```
CDEB:80 05    CDF2  292           BRA   WIN3          ;done converting
CDED:30 03    CDF2  293 WIN2      BMI   WIN3          ;=>80: no convert
CDEF:20 80 CE       294           JSR   SCRN48        ;40: convert to 80
CDF2:20 9D CC       295 WIN3      JSR   GETCUR        ;determine absolute CH
CDF5:98             296           TYA                 ;in case the window setting
CDF6:18             297           CLC                 ;was different
CDF7:65 20          298           ADC   WNDLFT
CDF9:28             299           PLP                 ;pin to right edge if
CDFA:B0 06    CE02  300           BCS   WIN4          ;80 to 40 leaves cursor
CDFC:C9 28          301           CMP   #40           ;off the screen
CDFE:90 02    CE02  302           BCC   WIN4
CE00:A9 27          303           LDA   #39
CE02:20 EC FE       304 WIN4      JSR   SETCUR        ;set new cursor
CE05:A5 25          305           LDA   CV            ;set new base address
CE07:20 C1 FB       306           JSR   BASCALC       ;for left = 0 (always)
CE0A:               307 *
CE0A:64 20          308 WNDREST   STZ   WNDLFT        ;Called by INIT and Pascal
CE0C:A9 18          309           LDA   #$18          ;and bottom
CE0E:85 23          310           STA   WNDBTM
CE10:A9 28          311           LDA   #$28          ;set left,width,bottom
CE12:2C 1F C0       312           BIT   RD80VID       ;set width to 80 if 80 columns
CE15:10 01    CE18  313           BPL   WIN5
CE17:0A             314           ASL   A
CE18:85 21          315 WIN5      STA   WNDWDTH       ;set width
CE1A:60             316 SETX      RTS                 ;exit used by SET40/80
CE1B:               317 *
CE1B:               318 * Turn on video firmware:
CE1B:               319 *
CE1B:               320 * This routine is used by BASIC init, ESC-4, ESC-8
CE1B:               321 * It copies the Monitor ROM to the language card
CE1B:               322 * if necessary; it sets the input and output hooks to
CE1B:               323 * $C30x; it sets all switches for video firmware operation
CE1B:               324 *
CE1B:2C 7B 06       325 HOOKITUP  BIT   VFACTV        ;don't touch hooks
CE1E:10 11    CE31  326           BPL   VIDMODE       ;if video firmware already active
CE20:20 38 C3       327 HOOKUP    JSR   COPYROM       ;Copy ROM to LC?
CE23:A9 05          328 SETHOOKS  LDA   #>C3KEYIN     ;set up $C300 hooks
CE25:85 38          329           STA   KSWL
CE27:A9 07          330           LDA   #>C3COUT1
CE29:85 36          331           STA   CSWL
CE2B:A9 C3          332           LDA   #<C3COUT1
CE2D:85 39          333           STA   KSWH
CE2F:85 37          334           STA   CSWH
CE31:               335 *
CE31:               336 * Now set the video firmware active
CE31:               337 *
CE31:9C FB 07       338 VIDMODE   STZ   CURSOR        ;set a solid inverse cursor
CE34:A9 08          339           LDA   #M.CTL        ;preserve M.CTL bit
CE36:2D FB 04       340           AND   VMODE
CE39:09 81          341           ORA   #M.PASCAL+M.MOUSE ;no pascal,mouse
CE3B:               342 *
CE3B:               343 * Pascal calls here to set its mode
CE3B:               344 *
CE3B:8D FB 04       345 PVMODE    STA   VMODE         ;set mode bits
CE3E:9C 7B 06       346           STZ   VFACTV        ;say video firmware active
CE41:8D 0F C0       347           STA   SETALTCHAR    ;and set alternate char set
CE44:60             348 QX        RTS
CE45:               349 *
```

```
CE45:                   350 * QUIT converts the screen from 80 to 40 if necessary,
CE45:                   351 * sets a 40 column window, and restores the normal I/O
CE45:                   352 * hooks (COUT1 and KEYIN).
CE45:                   353 *
CE45:2C FB 04           354 QUIT        BIT   VMODE          ;no quitting from pascal
CE48:10 FA    CE44      355             BPL   QX
CE4A:20 D2 CD           356             JSR   WIN40          ;first, do an escape 4
CE4D:20 89 FE           357 ZZQUIT      JSR   SETKBD         ;do a IN#0  (used by COMM)
CE50:4C 93 FE           358             JMP   SETVID         ;and a PR#0
```

```
CE53:                 360 *
CE53:                 361 * SCRN84 and SCRN48 convert screens between 40 & 80 cols.
CE53:                 362 * WNDTOP must be set up to indicate the last line to
CE53:                 363 * be done.  All registers are trashed.
CE53:                 364 *
CE53:A2 17            365 SCRN84    LDX   #23           ;start at bottom of screen
CE55:8D 01 C0         366           STA   SET80COL      ;allow page 2 access
CE58:8A               367 SCR1      TXA                 ;calc base for line
CE59:20 C1 FB         368           JSR   BASCALC
CE5C:A0 27            369           LDY   #39           ;start at right of screen
CE5E:5A               370 SCR2      PHY                 ;save 40 index
CE5F:98               371           TYA                 ;div by 2 for 80 column index
CE60:4A               372           LSR   A
CE61:B0 03   CE66     373           BCS   SCR3
CE63:2C 55 C0         374           BIT   TXTPAGE2      ;even column, do page 2
CE66:A8               375 SCR3      TAY                 ;get 80 index
CE67:B1 28            376           LDA   (BASL),Y      ;get 80 char
CE69:2C 54 C0         377           BIT   TXTPAGE1      ;restore page1
CE6C:7A               378           PLY                 ;get 40 index
CE6D:91 28            379           STA   (BASL),Y
CE6F:88               380           DEY
CE70:10 EC   CE5E     381           BPL   SCR2          ;do next 40 byte
CE72:CA               382           DEX                 ;do next line
CE73:30 04   CE79     383           BMI   SCR4          ;=>done with setup
CE75:E4 22            384           CPX   WNDTOP        ;at top yet?
CE77:B0 DF   CE58     385           BCS   SCR1
CE79:8D 00 C0         386 SCR4      STA   CLR80COL      ;clear 80STORE for 40 columns
CE7C:8D 0C C0         387           STA   CLR80VID      ;clear 80VID for 40 columns
CE7F:60               388           RTS
CE80:                 389 *
CE80:A2 17            390 SCRN48    LDX   #23           ;start at bottom of screen
CE82:8A               391 SCR5      TXA                 ;set base for current line
CE83:20 C1 FB         392           JSR   BASCALC
CE86:A0 00            393           LDY   #0            ;start at left of screen
CE88:8D 01 C0         394           STA   SET80COL      ;enable page2 store
CE8B:B1 28            395 SCR6      LDA   (BASL),Y      ;get 40 column char
CE8D:5A               396 SCR8      PHY                 ;save 40 column index
CE8E:48               397           PHA                 ;save char
CE8F:98               398           TYA                 ;div 2 for 80 column index
CE90:4A               399           LSR   A
CE91:B0 03   CE96     400           BCS   SCR7          ;save on page1
CE93:8D 55 C0         401           STA   TXTPAGE2
CE96:A8               402 SCR7      TAY                 ;get 80 column index
CE97:68               403           PLA                 ;now save character
CE98:91 28            404           STA   (BASL),Y
CE9A:8D 54 C0         405           STA   TXTPAGE1      ;flip page1
CE9D:7A               406           PLY                 ;restore 40 column index
CE9E:C8               407           INY                 ;move to the right
CE9F:C0 28            408           CPY   #40           ;at right yet?
CEA1:90 E8   CE8B     409           BCC   SCR6          ;=>no, do next column
CEA3:20 CF CB         410           JSR   CLRHALF       ;clear half of screen
CEA6:CA               411           DEX                 ;else do next line of screen
CEA7:30 04   CEAD     412           BMI   SCR9          ;=>done with top line
CEA9:E4 22            413           CPX   WNDTOP        ;at top yet?
CEAB:B0 D5   CE82     414           BCS   SCR5
CEAD:8D 0D C0         415 SCR9      STA   SET80VID      ;convert to 80 columns
CEB0:60               416           RTS
CEB1:                  44           INCLUDE PASCAL      ;Pascal support stuff
```

```
CEB1:AA              3 PSTATUS   TAX                 ;is request code = 0?
CEB2:F0 08   CEBC    4           BEQ   PIORDY        ;=>yes, ready for output
CEB4:CA              5           DEX                 ;check for any input
CEB5:D0 07   CEBE    6           BNE   PSTERR        ;=>bad request, return error
CEB7:20 E6 C8        7           JSR   XBITKBD       ;test keyboard
CEBA:10 04   CEC0    8           BPL   PNOTRDY       ;=>no keystroked
CEBC:38              9 PIORDY    SEC                 ;good return
CEBD:60             10           RTS
CEBE:A2 03          11 PSTERR    LDX   #3            ;else flag error
CEC0:18             12 PNOTRDY   CLC
CEC1:60             13           RTS
CEC2:               14 *
CEC2:               15 * PASCAL OUTPUT:
CEC2:               16 *
CEC2:        CEC2   17 PWRITE    EQU   *
CEC2:09 80          18           ORA   #$80          ;turn on high bit
CEC4:AA             19           TAX                 ;save character
CEC5:20 54 CF       20           JSR   PSETUP2       ;SETUP ZP STUFF, don't set ROM
CEC8:A9 08          21           LDA   #M.GOXY       ;ARE WE DOING GOTOXY?
CECA:2C FB 04       22           BIT   VMODE
CECD:D0 2B   CEFA   23           BNE   GETX          ;=>Doing X or Y?
CECF:8A             24           TXA                 ;now check for control char
CED0:89 60          25           BIT   #$60          ;is it control?
CED2:F0 45   CF19   26           BEQ   PCTL          ;=>yes, do control
CED4:AC 7B 05       27           LDY   OURCH         ;get horizontal position
CED7:24 32          28           BIT   INVFLG        ;check for inverse
CED9:30 02   CEDD   29           BMI   PWR1          ;normal, go store it
CEDB:29 7F          30           AND   #$7F
CEDD:20 C1 C3       31 PWR1      JSR   STORE         ;now store it (erasing cursor)
CEE0:C8             32           INY                 ;INC CH
CEE1:8C 7B 05       33           STY   OURCH
CEE4:C4 21          34           CPY   WNDWDTH
CEE6:90 0C   CEF4   35           BCC   PWRET
CEE8:20 60 C3       36           JSR   SETROM
CEEB:20 E9 FE       37           JSR   CLRCH         ;set cursor position to 0
CEEE:20 66 FC       38           JSR   LF
CEF1:20 54 C3       39 PWRITERET JSR   RESETLC
CEF4:20 0B CC       40 PWRET     JSR   PASINVERT     ;display new cursor
CEF7:A2 00          41 PRET      LDX   #$0           ;return with no error
CEF9:60             42           RTS
CEFA:               43 *
CEFA:               44 *  HANDLE GOTOXY STUFF:
CEFA:               45 *
CEFA:        CEFA   46 GETX      EQU   *
CEFA:20 0B CC       47           JSR   PASINVERT     ;turn off cursor
CEFD:8A             48           TXA                 ;get character
CEFE:38             49           SEC
CEFF:E9 A0          50           SBC   #160          ;MAKE BINARY
CF01:2C FB 06       51           BIT   XCOORD        ;doing X?
CF04:30 2A   CF30   52           BMI   PSETX         ;=>yes, set it
CF06:               53 *
CF06:               54 * Set Y and do the GOTOXY
CF06:               55 *
CF06:        CF06   56 GETY      EQU   *
CF06:8D FB 05       57           STA   OURCV
CF09:20 71 CF       58           JSR   PASCALC       ;calc base addr
CF0C:AC FB 06       59           LDY   XCOORD
CF0F:20 AD CC       60           JSR   GETCUR2       ;set proper cursors
```

```
CF12:A9 08          61            LDA   #M.GOXY       ;turn off gotoxy
CF14:1C FB 04       62            TRB   VMODE
CF17:80 DB   CEF4   63            BRA   PWRET         ;=>DONE (ALWAYS TAKEN)
CF19:               64 *
CF19:20 0B CC       65 PCTL       JSR   PASINVERT     ;turn off cursor
CF1C:8A             66            TXA                 ;get char
CF1D:C9 9E          67            CMP   #$9E          ;is it gotoXY?
CF1F:F0 08   CF29   68            BEQ   STARTXY       ;=>yes, start it up
CF21:20 60 C3       69            JSR   SETROM        ;must switch in ROM for controls
CF24:20 58 CD       70            JSR   CTLCHAR       ;EXECUTE IT IF POSSIBLE
CF27:80 C8   CEF1   71            BRA   PWRITERET     ;=>display new cursor, exit
CF29:               72 *
CF29:               73 * START THE GOTOXY SEQUENCE:
CF29:               74 *
CF29:        CF29   75 STARTXY    EQU   *
CF29:A9 08          76            LDA   #M.GOXY
CF2B:0C FB 04       77            TSB   VMODE         ;turn on gotoxy
CF2E:A9 FF          78            LDA   #$FF          ;set XCOORD to -1
CF30:8D FB 06       79 PSETX      STA   XCOORD        ;set X
CF33:80 BF   CEF4   80            BRA   PWRET         ;=>display cursor and exit
CF35:               81 *
CF35:               82 * PASCAL INPUT:
CF35:               83 *
CF35:20 54 CF       84 PASREAD    JSR   PSETUP2       ;SETUP ZP STUFF
CF38:20 D5 C8       85 GKEY       JSR   XRDKBD        ;key pressed?
CF3B:10 FB   CF38   86            BPL   GKEY          ;=>not yet
CF3D:29 7F          87            AND   #$7F          ;DROP HI BIT
CF3F:80 B6   CEF7   88            BRA   PRET          ;good exit
CF41:               89 *
CF41:               90 * PASCAL INITIALIZATION:
CF41:               91 *
CF41:        CF41   92 PINIT      EQU   *
CF41:A9 01          93            LDA   #M.MOUSE      ;Set mode to pascal
CF43:20 3B CE       94            JSR   PVMODE        ;without mouse characters
CF46:20 51 CF       95            JSR   PSETUP        ;setup zero page for pascal
CF49:20 D4 CD       96            JSR   WIN80         ;do 40->80 convert
CF4C:20 58 FC       97            JSR   HOME          ;home and clear screen
CF4F:80 A0   CEF1   98            BRA   PWRITERET     ;display cursor, set OURCH,OURCV...
CF51:               99 *
CF51:        CF51  100 PSETUP     EQU   *
CF51:20 60 C3      101            JSR   SETROM        ;save LC state, set ROM read
CF54:64 22         102 PSETUP2    STZ   WNDTOP        ;set top to 0
CF56:20 0A CE      103            JSR   WNDREST       ;init either 40 or 80 window
CF59:A9 FF         104            LDA   #$FF          ;assume normal text
CF5B:85 32         105            STA   INVFLG
CF5D:A9 04         106            LDA   #M.VMODE      ;is it
CF5F:2C FB 04      107            BIT   VMODE
CF62:F0 02   CF66  108            BEQ   PS1           ;=>yes
CF64:46 32         109            LSR   INVFLG        ;no, make flag inverse
CF66:AC 7B 05      110 PS1        LDY   OURCH
CF69:20 AD CC      111            JSR   GETCUR2       ;set all cursors
CF6C:AD FB 05      112            LDA   OURCV
CF6F:85 25         113            STA   CV
CF71:              114 *
CF71:              115 * Put BASCALC here so we don't have to switch
CF71:              116 * in the ROMs for each character output.
CF71:              117 *
CF71:0A            118 PASCALC    ASL   A
```

```
CF72:A8          119           TAY               ;calc base addr in BASL,H
CF73:4A          120           LSR   A           ;for given line no.
CF74:4A          121           LSR   A
CF75:29 03       122           AND   #$03        ; 0<=line no.<=$17
CF77:09 04       123           ORA   #$4         ; arg=000ABCDE, generate
CF79:85 29       124           STA   BASH        ; BASH=000001CD
CF7B:98          125           TYA               ;and
CF7C:6A          126           ROR   A           ; BASL=EABAB000
CF7D:29 98       127           AND   #$98
CF7F:85 28       128 PASCLC2   STA   BASL
CF81:0A          129           ASL   A
CF82:0A          130           ASL   A
CF83:04 28       131           TSB   BASL
CF85:60          132           RTS
CF86:             45           include moremisc  ;More random junk
```

```
CF86:                 2 ****************************************
CF86:                 3 *
CF86:                 4 * Here are more miscellaneous routines
CF86:                 5 * stuffed here in a valiant effort to make other code align
CF86:                 6 * properly
CF86:                 7 *
CF86:                 8 ****************************************


CF86:                10 * Various tables
CF86:83 8B 8B        11 irqtble   dfb   >lcbank2,>lcbank1,>lcbank1
CF89:05 03 55        12           dfb   >wrcardram,>rdcardram,>txtpage2

CF8C:9E 0B 40 50     14 comtbl    dfb   $9E,$0B,$40,$50,$16,$0B,$01,$00

CF94:CD C1 D8 D9     16 rtbl      asc   'MAXYPS'


CF9A:                18 ****************************************
CF9A:                19 *
CF9A:                20 * MOVEIRQ - This routine transfers the roms interrupt vector into
CF9A:                21 *  both language cards
CF9A:                22 *
CF9A:                23 ****************************************
CF9A:         CF9A   24 moveirq   equ   *
CF9A:20 60 C3        25           JSR   SETROM         ;Read ROM and Write to RAM
CF9D:AD 16 C0        26           LDA   RDALTZP        ;Which language card?
CFA0:0A              27           ASL   A              ;C=1 if alternate card
CFA1:A0 01           28           LDY   #1             ;Move two bytes
CFA3:B9 FE FF        29 MIRQLP    LDA   IRQVECT,Y      ;Get byte from ROM
CFA6:8D 09 C0        30           STA   SETALTZP       ;Set alternate card
CFA9:99 FE FF        31           STA   IRQVECT,Y      ;Store it in the RAM card
CFAC:8D 08 C0        32           STA   SETSTDZP       ;Set main card
CFAF:99 FE FF        33           STA   IRQVECT,Y
CFB2:88              34           DEY
CFB3:10 EE    CFA3   35           BPL   MIRQLP         ;Go do the second byte
CFB5:90 03    CFBA   36           BCC   MIRQSTD        ;Is the card set right?
CFB7:8D 09 C0        37           STA   SETALTZP       ;No, it wasn't
CFBA:4C 54 C3        38 MIRQSTD   JMP   RESETLC        ;Clean up & go home


CFBD:                40 ****************************************
CFBD:                41 * CLRKBD2 - Moved here from scrolling routines
CFBD:                42 ****************************************
CFBD:         CFBD   43 clrkbd2   equ   *
CFBD:5A              44           phy                  ;Now preserves Y
CFBE:20 B3 C3        45           jsr   story
CFC1:7A              46           ply
CFC2:4C D5 C8        47           jmp   xrdkbd


CFC5:                49 ****************************************
CFC5:                50 *
CFC5:                51 * LOOKASC - addition to monitor input routine
```

```
CFC5:                52 * if a quote (') in input, the ascii of the next is input
CFC5:                53 * like a hex number
CFC5:                54 *
CFC5:                55 ****************************************
CFC5:         CFC5   56 lookasc   equ   *
CFC5:B0 11    CFD8   57           bcs   ladig          ;Was char a hex digit?
CFC7:C9 A0           58           cmp   #$A0           ;Is it a quote
CFC9:D0 13    CFDE   59           bne   ladone         ;Done if not
CFCB:B9 00 02        60           lda   inbuf,y        ;Get next char
CFCE:A2 07           61           ldx   #7             ;for shifting asc into A2L and A2H
CFD0:C9 8D           62           cmp   #$8D           ;Was it a cr?
CFD2:F0 07    CFDB   63           beq   lacr           ;Go handle cr
CFD4:C8              64           iny                   ;Advance index into inbuf
CFD5:4C 90 FF        65           jmp   nxtbit         ;Go shift it in
CFD8:4C 8A FF        66 ladig     jmp   dig
CFDB:4C A7 FF        67 lacr      jmp   getnum
CFDE:60              68 ladone    rts
CFDF:         0021   46           ds    $D000-*,0
---- NEXT OBJECT FILE NAME IS /BUILD/FIRM.1
F800:         F800   47           ORG   F8ORG
F800:                48           INCLUDE AUTOST1      ;F8 monitor rom
```

```
F800:4A              3 PLOT      LSR   A              ;Y-COORD/2
F801:08              4           PHP                  ;SAVE LSB IN CARRY
F802:20 47 F8        5           JSR   GBASCALC       ;CALC BASE ADR IN GBASL,H
F805:28              6           PLP                  ;RESTORE LSB FROM CARRY
F806:A9 0F           7           LDA   #$0F           ;MASK $0F IF EVEN
F808:90 02   F80C    8           BCC   RTMASK
F80A:69 E0           9           ADC   #$E0           ;MASK $F0 IF ODD
F80C:85 2E          10 RTMASK    STA   MASK
F80E:B1 26          11 PLOT1     LDA   (GBASL),Y      ;DATA
F810:45 30          12           EOR   COLOR          ; XOR COLOR
F812:25 2E          13           AND   MASK           ;  AND MASK
F814:51 26          14           EOR   (GBASL),Y      ;   XOR DATA
F816:91 26          15           STA   (GBASL),Y      ;    TO DATA
F818:60             16           RTS
F819:               17 *
F819:20 00 F8       18 HLINE     JSR   PLOT           ;PLOT SQUARE
F81C:C4 2C          19 HLINE1    CPY   H2             ;DONE?
F81E:B0 11   F831   20           BCS   RTS1           ; YES, RETURN
F820:C8             21           INY                  ; NO, INCR INDEX (X-COORD)
F821:20 0E F8       22           JSR   PLOT1          ;PLOT NEXT SQUARE
F824:90 F6   F81C   23           BCC   HLINE1         ;ALWAYS TAKEN
F826:69 01          24 VLINEZ    ADC   #$01           ;NEXT Y-COORD
F828:48             25 VLINE     PHA                  ; SAVE ON STACK
F829:20 00 F8       26           JSR   PLOT           ; PLOT SQUARE
F82C:68             27           PLA
F82D:C5 2D          28           CMP   V2             ;DONE?
F82F:90 F5   F826   29           BCC   VLINEZ         ; NO, LOOP.
F831:60             30 RTS1      RTS
F832:               31 *
F832:A0 2F          32 CLRSCR    LDY   #$2F           ;MAX Y, FULL SCRN CLR
F834:D0 02   F838   33           BNE   CLRSC2         ;ALWAYS TAKEN
F836:A0 27          34 CLRTOP    LDY   #$27           ;MAX Y, TOP SCRN CLR
F838:84 2D          35 CLRSC2    STY   V2             ;STORE AS BOTTOM COORD
F83A:               36 ;                             FOR VLINE CALLS
F83A:A0 27          37           LDY   #$27           ;RIGHTMOST X-COORD (COLUMN)
F83C:A9 00          38 CLRSC3    LDA   #$00           ;TOP COORD FOR VLINE CALLS
F83E:85 30          39           STA   COLOR          ;CLEAR COLOR (BLACK)
F840:20 28 F8       40           JSR   VLINE          ;DRAW VLINE
F843:88             41           DEY                  ;NEXT LEFTMOST X-COORD
F844:10 F6   F83C   42           BPL   CLRSC3         ;LOOP UNTIL DONE.
F846:60             43           RTS
F847:               44 *
F847:48             45 GBASCALC  PHA                  ;FOR INPUT  00DEFGH
F848:4A             46           LSR   A
F849:29 03          47           AND   #$03
F84B:09 04          48           ORA   #$04           ;GENERATE GBASH=000001FG
F84D:85 27          49           STA   GBASH
F84F:68             50           PLA                  ;AND GBASL=HDEDE000
F850:29 18          51           AND   #$18
F852:90 02   F856   52           BCC   GBCALC
F854:69 7F          53           ADC   #$7F
F856:85 26          54 GBCALC    STA   GBASL
F858:0A             55           ASL   A
F859:0A             56           ASL   A
F85A:05 26          57           ORA   GBASL
F85C:85 26          58           STA   GBASL
F85E:60             59           RTS
F85F:               60 *
```

```
F85F:A5 30          61 NXTCOL   LDA   COLOR          ;INCREMENT COLOR BY 3
F861:18             62          CLC
F862:69 03          63          ADC   #$03
F864:29 0F          64 SETCOL   AND   #$0F          ;SETS COLOR=17*A MOD 16
F866:85 30          65          STA   COLOR
F868:0A             66          ASL   A             ;BOTH HALF BYTES OF COLOR EQUAL
F869:0A             67          ASL   A
F86A:0A             68          ASL   A
F86B:0A             69          ASL   A
F86C:05 30          70          ORA   COLOR
F86E:85 30          71          STA   COLOR
F870:60             72          RTS
F871:               73 *
F871:4A             74 SCRN     LSR   A             ;READ SCREEN Y-COORD/2
F872:08             75          PHP                 ;SAVE LSB (CARRY)
F873:20 47 F8       76          JSR   GBASCALC      ;CALC BASE ADDRESS
F876:B1 26          77          LDA   (GBASL),Y     ;GET BYTE
F878:28             78          PLP                 ;RESTORE LSB FROM CARRY
F879:90 04   F87F   79 SCRN2    BCC   RTMSKZ        ;IF EVEN, USE LO H
F87B:4A             80          LSR   A
F87C:4A             81          LSR   A
F87D:4A             82          LSR   A             ;SHIFT HIGH HALF BYTE DOWN
F87E:4A             83          LSR   A
F87F:29 0F          84 RTMSKZ   AND   #$0F          ;MASK 4-BITS
F881:60             85          RTS
F882:               86 *
F882:A6 3A          87 INSDS1   LDX   PCL           ;PRINT PCL,H
F884:A4 3B          88          LDY   PCH
F886:20 96 FD       89          JSR   PRYX2
F889:20 48 F9       90          JSR   PRBLNK        ;FOLLOWED BY A BLANK
F88C:A1 3A          91          LDA   (PCL,X)       ;GET OPCODE
F88E:A8             92 INSDS2   TAY                  ;Lable moved down 1
F88F:4A             93          LSR   A             ;EVEN/ODD TEST
F890:90 05   F897   94          BCC   IEVEN
F892:6A             95          ROR   A             ;BIT 1 TEST
F893:B0 0C   F8A1   96          BCS   ERR           ;XXXXXX11 INVALID OP
F895:29 87          97          AND   #$87          ;MASK BITS
F897:4A             98 IEVEN    LSR   A             ;LSB INTO CARRY FOR L/R TEST
F898:AA             99          TAX
F899:BD 62 F9      100          LDA   FMT1,X        ;GET FORMAT INDEX BYTE
F89C:20 79 F8      101          JSR   SCRN2         ;R/L H-BYTE ON CARRY
F89F:D0 04   F8A5  102          BNE   GETFMT
F8A1:A0 FC         103 ERR      LDY   #$FC          ;SUBSTITUTE $FC FOR INVALID OPS
F8A3:A9 00         104          LDA   #$00          ;SET PRINT FORMAT INDEX TO 0
F8A5:AA            105 GETFMT   TAX
F8A6:BD A6 F9      106          LDA   FMT2,X        ;INDEX INTO PRINT FORMAT TABLE
F8A9:85 2E         107          STA   FORMAT        ;SAVE FOR ADR FIELD FORMATTING
F8AB:29 03         108          AND   #$03          ;MASK FOR 2-BIT LENGTH
F8AD:              109 ; (0=1 BYTE, 1=2 BYTE, 2=3 BYTE)
F8AD:85 2F         110          STA   LENGTH
F8AF:20 35 FC      111          JSR   NEWOPS       ;get index for new opcodes
F8B2:F0 18   F8CC  112          BEQ   GOTONE       ;found a new op (or no op)
F8B4:29 8F         113          AND   #$8F          ;MASK FOR 1XXX1010 TEST
F8B6:AA            114          TAX                 ; SAVE IT
F8B7:98            115          TYA                 ;OPCODE TO A AGAIN
F8B8:A0 03         116          LDY   #$03
F8BA:E0 8A         117          CPX   #$8A
F8BC:F0 0B   F8C9  118          BEQ   MNNDX3
```

```
F8BE:4A             119 MNNDX1    LSR   A
F8BF:90 08   F8C9   120           BCC   MNNDX3          ;FORM INDEX INTO MNEMONIC TABLE
F8C1:4A             121           LSR   A
F8C2:4A             122 MNNDX2    LSR   A             ;  1) 1XXX1010 => 00101XXX
F8C3:09 20          123           ORA   #$20            ;  2) XXXYYY01 => 00111XXX
F8C5:88             124           DEY                 ;  3) XXXYYY10 => 00110XXX
F8C6:D0 FA   F8C2   125           BNE   MNNDX2          ;  4) XXXYY100 => 00100XXX
F8C8:C8             126           INY                 ;  5) XXXXX000 => 000XXXXX
F8C9:88             127 MNNDX3    DEY
F8CA:D0 F2   F8BE   128           BNE   MNNDX1
F8CC:60             129 GOTONE    RTS
F8CD:               130 *
F8CD:FF FF FF       131           DFB   $FF,$FF,$FF
F8D0:               132 *
F8D0:20 82 F8       133 INSTDSP   JSR   INSDS1          ;GEN FMT, LEN BYTES
F8D3:48             134           PHA                   ;SAVE MNEMONIC TABLE INDEX
F8D4:B1 3A          135 PRNTOP    LDA   (PCL),Y
F8D6:20 DA FD       136           JSR   PRBYTE
F8D9:A2 01          137           LDX   #$01            ;PRINT 2 BLANKS
F8DB:20 4A F9       138 PRNTBL    JSR   PRBL2
F8DE:C4 2F          139           CPY   LENGTH          ;PRINT INST (1-3 BYTES)
F8E0:C8             140           INY                   ;IN A 12 CHR FIELD
F8E1:90 F1   F8D4   141           BCC   PRNTOP
F8E3:A2 03          142           LDX   #$03            ;CHAR COUNT FOR MNEMONIC INDEX
F8E5:C0 04          143           CPY   #$04
F8E7:90 F2   F8DB   144           BCC   PRNTBL
F8E9:68             145           PLA                   ;RECOVER MNEMONIC INDEX
F8EA:A8             146           TAY
F8EB:B9 C0 F9       147           LDA   MNEML,Y
F8EE:85 2C          148           STA   LMNEM           ;FETCH 3-CHAR MNEMONIC
F8F0:B9 00 FA       149           LDA   MNEMR,Y         ;  (PACKED INTO 2-BYTES)
F8F3:85 2D          150           STA   RMNEM
F8F5:A9 00          151 PRMN1     LDA   #$00
F8F7:A0 05          152           LDY   #$05
F8F9:06 2D          153 PRMN2     ASL   RMNEM           ;SHIFT 5 BITS OF CHARACTER INTO A
F8FB:26 2C          154           ROL   LMNEM
F8FD:2A             155           ROL   A               ;  (CLEARS CARRY)
F8FE:88             156           DEY
F8FF:D0 F8   F8F9   157           BNE   PRMN2
F901:69 BF          158           ADC   #$BF            ;ADD "?" OFFSET
F903:20 ED FD       159           JSR   COUT            ;OUTPUT A CHAR OF MNEM
F906:CA             160           DEX
F907:D0 EC   F8F5   161           BNE   PRMN1
F909:20 48 F9       162           JSR   PRBLNK          ;OUTPUT 3 BLANKS
F90C:A4 2F          163           LDY   LENGTH
F90E:A2 06          164           LDX   #$06            ;CNT FOR 6 FORMAT BITS
F910:E0 03          165 PRADR1    CPX   #$03
F912:F0 1C   F930   166           BEQ   PRADR5          ;IF X=3 THEN ADDR.
F914:06 2E          167 PRADR2    ASL   FORMAT
F916:90 0E   F926   168           BCC   PRADR3
F918:BD B9 F9       169           LDA   CHAR1-1,X
F91B:20 ED FD       170           JSR   COUT
F91E:BD B3 F9       171           LDA   CHAR2-1,X
F921:F0 03   F926   172           BEQ   PRADR3
F923:20 ED FD       173           JSR   COUT
F926:CA             174 PRADR3    DEX
F927:D0 E7   F910   175           BNE   PRADR1
F929:60             176           RTS
```

```
F92A:                 177 *
F92A:88               178 PRADR4    DEY
F92B:30 E7   F914     179           BMI    PRADR2
F92D:20 DA FD         180           JSR    PRBYTE
F930:A5 2E            181 PRADR5    LDA    FORMAT
F932:C9 E8            182           CMP    #$E8            ;HANDLE REL ADR MODE
F934:B1 3A            183           LDA    (PCL),Y         ;SPECIAL (PRINT TARGET,
F936:90 F2   F92A     184           BCC    PRADR4          ;  NOT OFFSET)
F938:20 56 F9         185 RELADR    JSR    PCADJ3
F93B:AA               186           TAX                    ;PCL,PCH+OFFSET+1 TO A,Y
F93C:E8               187           INX
F93D:D0 01   F940     188           BNE    PRNTYX          ;+1 TO Y,X
F93F:C8               189           INY
F940:98               190 PRNTYX    TYA
F941:20 DA FD         191 PRNTAX    JSR    PRBYTE          ;OUTPUT TARGET ADR
F944:8A               192 PRNTX     TXA                    ;  OF BRANCH AND RETURN
F945:4C DA FD         193           JMP    PRBYTE
F948:                 194 *
F948:A2 03            195 PRBLNK    LDX    #$03            ;BLANK COUNT
F94A:A9 A0            196 PRBL2     LDA    #$A0            ;LOAD A SPACE
F94C:20 ED FD         197 PRBL3     JSR    COUT            ;OUTPUT A BLANK
F94F:CA               198           DEX
F950:D0 F8   F94A     199           BNE    PRBL2           ;LOOP UNTIL COUNT=0
F952:60               200           RTS
F953:                 201 *
F953:38               202 PCADJ     SEC                    ;0=1 BYTE, 1=2 BYTE,
F954:A5 2F            203 PCADJ2    LDA    LENGTH          ;  2=3 BYTE
F956:A4 3B            204 PCADJ3    LDY    PCH
F958:AA               205           TAX                    ;TEST DISPLACEMENT SIGN
F959:10 01   F95C     206           BPL    PCADJ4          ;  (FOR REL BRANCH)
F95B:88               207           DEY                    ;EXTEND NEG BY DECR PCH
F95C:65 3A            208 PCADJ4    ADC    PCL
F95E:90 01   F961     209           BCC    RTS2            ;PCL+LENGTH(OR DISPL)+1 TO A
F960:C8               210           INY                    ;  CARRY INTO Y (PCH)
F961:60               211 RTS2      RTS
F962:                 212 *
F962:                 213 ; FMT1 BYTES:    XXXXXXY0 INSTRS
F962:                 214 ; IF Y=0         THEN RIGHT HALF BYTE
F962:                 215 ; IF Y=1         THEN LEFT HALF BYTE
F962:                 216 ;                  (X=INDEX)
F962:                 217 *
F962:0F               218 FMT1      DFB    $0F
F963:22               219           DFB    $22
F964:FF               220           DFB    $FF
F965:33               221           DFB    $33
F966:CB               222           DFB    $CB
F967:62               223           DFB    $62
F968:FF               224           DFB    $FF
F969:73               225           DFB    $73
F96A:03               226           DFB    $03
F96B:22               227           DFB    $22
F96C:FF               228           DFB    $FF
F96D:33               229           DFB    $33
F96E:CB               230           DFB    $CB
F96F:66               231           DFB    $66
F970:FF               232           DFB    $FF
F971:77               233           DFB    $77
F972:0F               234           DFB    $0F
```

```
F973:20          235             DFB   $20
F974:FF          236             DFB   $FF
F975:33          237             DFB   $33
F976:CB          238             DFB   $CB
F977:60          239             DFB   $60
F978:FF          240             DFB   $FF
F979:70          241             DFB   $70
F97A:0F          242             DFB   $0F
F97B:22          243             DFB   $22
F97C:FF          244             DFB   $FF
F97D:39          245             DFB   $39
F97E:CB          246             DFB   $CB
F97F:66          247             DFB   $66
F980:FF          248             DFB   $FF
F981:7D          249             DFB   $7D
F982:0B          250             DFB   $0B
F983:22          251             DFB   $22
F984:FF          252             DFB   $FF
F985:33          253             DFB   $33
F986:CB          254             DFB   $CB
F987:A6          255             DFB   $A6
F988:FF          256             DFB   $FF
F989:73          257             DFB   $73
F98A:11          258             DFB   $11
F98B:22          259             DFB   $22
F98C:FF          260             DFB   $FF
F98D:33          261             DFB   $33
F98E:CB          262             DFB   $CB
F98F:A6          263             DFB   $A6
F990:FF          264             DFB   $FF
F991:87          265             DFB   $87
F992:01          266             DFB   $01
F993:22          267             DFB   $22
F994:FF          268             DFB   $FF
F995:33          269             DFB   $33
F996:CB          270             DFB   $CB
F997:60          271             DFB   $60
F998:FF          272             DFB   $FF
F999:70          273             DFB   $70
F99A:01          274             DFB   $01
F99B:22          275             DFB   $22
F99C:FF          276             DFB   $FF
F99D:33          277             DFB   $33
F99E:CB          278             DFB   $CB
F99F:60          279             DFB   $60
F9A0:FF          280             DFB   $FF
F9A1:70          281             DFB   $70
F9A2:24          282             DFB   $24
F9A3:31          283             DFB   $31
F9A4:65          284             DFB   $65
F9A5:78          285             DFB   $78
F9A6:            286 ; ZZXXXY01 INSTR'S
F9A6:00          287 FMT2        DFB   $00       ;ERR
F9A7:21          288             DFB   $21       ;IMM
F9A8:81          289             DFB   $81       ;Z-PAGE
F9A9:82          290             DFB   $82       ;ABS
F9AA:59          291             DFB   $59       ;(ZPAG,X)
F9AB:4D          292             DFB   $4D       ;(ZPAG),Y
```

```
F9AC:91          293             DFB   $91          ;ZPAG,X
F9AD:92          294             DFB   $92           ;ABS,X
F9AE:86          295             DFB   $86           ;ABS,Y
F9AF:4A          296             DFB   $4A           ;(ABS)
F9B0:85          297             DFB   $85           ;ZPAG,Y
F9B1:9D          298             DFB   $9D           ;RELATIVE
F9B2:49          299             DFB   $49          ;(ZPAG)     (new)
F9B3:5A          300             DFB   $5A          ;(ABS,X)    (new)
F9B4:            301 *
F9B4:D9          302 CHAR2       DFB   $D9           ;'Y'
F9B5:00          303             DFB   $00          ; (byte F of FMT2)
F9B6:D8          304             DFB   $D8           ;'Y'
F9B7:A4          305             DFB   $A4           ;'$'
F9B8:A4          306             DFB   $A4           ;'$'
F9B9:00          307             DFB   $00
F9BA:            308 *
F9BA:AC          309 CHAR1       DFB   $AC           ;','
F9BB:A9          310             DFB   $A9           ;')'
F9BC:AC          311             DFB   $AC           ;','
F9BD:A3          312             DFB   $A3           ;'#'
F9BE:A8          313             DFB   $A8           ;'('
F9BF:A4          314             DFB   $A4           ;'$'
F9C0:1C          315 MNEML       DFB   $1C
F9C1:8A          316             DFB   $8A
F9C2:1C          317             DFB   $1C
F9C3:23          318             DFB   $23
F9C4:5D          319             DFB   $5D
F9C5:8B          320             DFB   $8B
F9C6:1B          321             DFB   $1B
F9C7:A1          322             DFB   $A1
F9C8:9D          323             DFB   $9D
F9C9:8A          324             DFB   $8A
F9CA:1D          325             DFB   $1D
F9CB:23          326             DFB   $23
F9CC:9D          327             DFB   $9D
F9CD:8B          328             DFB   $8B
F9CE:1D          329             DFB   $1D
F9CF:A1          330             DFB   $A1
F9D0:1C          331             DFB   $1C          ;BRA
F9D1:29          332             DFB   $29
F9D2:19          333             DFB   $19
F9D3:AE          334             DFB   $AE
F9D4:69          335             DFB   $69
F9D5:A8          336             DFB   $A8
F9D6:19          337             DFB   $19
F9D7:23          338             DFB   $23
F9D8:24          339             DFB   $24
F9D9:53          340             DFB   $53
F9DA:1B          341             DFB   $1B
F9DB:23          342             DFB   $23
F9DC:24          343             DFB   $24
F9DD:53          344             DFB   $53
F9DE:19          345             DFB   $19
F9DF:A1          346             DFB   $A1          ; (A) FORMAT ABOVE
F9E0:AD          347             DFB   $AD          ; TSB
F9E1:1A          348             DFB   $1A
F9E2:5B          349             DFB   $5B
F9E3:5B          350             DFB   $5B
```

```
F9E4:A5          351           DFB   $A5
F9E5:69          352           DFB   $69
F9E6:24          353           DFB   $24
F9E7:24          354           DFB   $24          ; (B) FORMAT
F9E8:AE          355           DFB   $AE
F9E9:AE          356           DFB   $AE
F9EA:A8          357           DFB   $A8
F9EB:AD          358           DFB   $AD
F9EC:29          359           DFB   $29
F9ED:8A          360           DFB   $8A
F9EE:7C          361           DFB   $7C
F9EF:8B          362           DFB   $8B          ; (C) FORMAT
F9F0:15          363           DFB   $15
F9F1:9C          364           DFB   $9C
F9F2:6D          365           DFB   $6D
F9F3:9C          366           DFB   $9C
F9F4:A5          367           DFB   $A5
F9F5:69          368           DFB   $69
F9F6:29          369           DFB   $29
F9F7:53          370           DFB   $53          ; (D) FORMAT
F9F8:84          371           DFB   $84
F9F9:13          372           DFB   $13
F9FA:34          373           DFB   $34
F9FB:11          374           DFB   $11
F9FC:A5          375           DFB   $A5
F9FD:69          376           DFB   $69
F9FE:23          377           DFB   $23          ; (E) FORMAT
F9FF:A0          378           DFB   $A0
FA00:            379 *
FA00:D8          380 MNEMR     DFB   $D8
FA01:62          381           DFB   $62
FA02:5A          382           DFB   $5A
FA03:48          383           DFB   $48
FA04:26          384           DFB   $26
FA05:62          385           DFB   $62
FA06:94          386           DFB   $94
FA07:88          387           DFB   $88
FA08:54          388           DFB   $54
FA09:44          389           DFB   $44
FA0A:C8          390           DFB   $C8
FA0B:54          391           DFB   $54
FA0C:68          392           DFB   $68
FA0D:44          393           DFB   $44
FA0E:E8          394           DFB   $E8
FA0F:94          395           DFB   $94
FA10:C4          396           DFB   $C4          ;BRA
FA11:B4          397           DFB   $B4
FA12:08          398           DFB   $08
FA13:84          399           DFB   $84
FA14:74          400           DFB   $74
FA15:B4          401           DFB   $B4
FA16:28          402           DFB   $28
FA17:6E          403           DFB   $6E
FA18:74          404           DFB   $74
FA19:F4          405           DFB   $F4
FA1A:CC          406           DFB   $CC
FA1B:4A          407           DFB   $4A
FA1C:72          408           DFB   $72
```

```
FA1D:F2             409           DFB   $F2
FA1E:A4             410           DFB   $A4
FA1F:8A             411           DFB   $8A          ; (A) FORMAT
FA20:06             412           DFB   $06          ; TSB
FA21:AA             413           DFB   $AA
FA22:A2             414           DFB   $A2
FA23:A2             415           DFB   $A2
FA24:74             416           DFB   $74
FA25:74             417           DFB   $74
FA26:74             418           DFB   $74
FA27:72             419           DFB   $72          ; (B) FORMAT
FA28:44             420           DFB   $44
FA29:68             421           DFB   $68
FA2A:B2             422           DFB   $B2
FA2B:32             423           DFB   $32
FA2C:B2             424           DFB   $B2
FA2D:72             425           DFB   $72
FA2E:22             426           DFB   $22
FA2F:72             427           DFB   $72          ; (C) FORMAT
FA30:1A             428           DFB   $1A
FA31:1A             429           DFB   $1A
FA32:26             430           DFB   $26
FA33:26             431           DFB   $26
FA34:72             432           DFB   $72
FA35:72             433           DFB   $72
FA36:88             434           DFB   $88
FA37:C8             435           DFB   $C8          ; (D) FORMAT
FA38:C4             436           DFB   $C4
FA39:CA             437           DFB   $CA
FA3A:26             438           DFB   $26
FA3B:48             439           DFB   $48
FA3C:44             440           DFB   $44
FA3D:44             441           DFB   $44
FA3E:A2             442           DFB   $A2
FA3F:C8             443           DFB   $C8          ; (E) FORMAT
FA40:               444 *
FA40:85 45          445 IRQ       STA   $45          ;+ Trash $45 for those who want it
FA42:A5 45          446           LDA   $45          ;+
FA44:4C 03 C8       447           JMP   NEWIRQ       ;+
FA47:               448 *
FA47:               449 *
FA47:               450 * NEWBRK is called by the interrupt handler which has
FA47:               451 * set the hardware to its default state and encoded
FA47:               452 * the state in the accumulator.  Software that wants
FA47:               453 * to do break processing using full system resources
FA47:               454 * can restore the machine state from this value.
FA47:               455 *
FA47:85 44          456 NEWBRK     STA   MACSTAT      ;save state of machine
FA49:7A             457           PLY                ;restore registers for save
FA4A:FA             458           PLX
FA4B:68             459           PLA
FA4C:               460 *
FA4C:28             461 BREAK     PLP                ;Note: same as old BREAK routine!!
FA4D:20 4A FF       462           JSR   SAVE         ;save reg's on BRK
FA50:68             463           PLA                ;including PC
FA51:85 3A          464           STA   PCL
FA53:68             465           PLA
FA54:85 3B          466           STA   PCH
```

```
FA56:6C F0 03      467             JMP   (BRKV)          ;call BRK HANDLER
FA59:              468 *
FA59:20 82 F8      469 OLDBRK      JSR   INSDS1          ;PRINT USER PC
FA5C:20 DA FA      470             JSR   RGDSP1          ;  AND REGS
FA5F:4C 65 FF      471             JMP   MON             ;GO TO MONITOR (NO PASS GO, NO $200!)
FA62:              472 *
FA62:D8            473 RESET       CLD                   ;DO THIS FIRST THIS TIME
FA63:20 84 FE      474             JSR   SETNORM
FA66:20 2F FB      475             JSR   INIT
FA69:20 4D CE      476             JSR   ZZQUIT          ;+ Setvid & Setkbd
FA6C:20 1A C4      477             JSR   INITMOUSE       ;initialize the mouse
FA6F:20 04 CC      478             JSR   CLRPORT         ;clear port setup bytes
FA72:9C FF 04      479             STZ   ACIABUF         ;and the commahead buffer
FA75:AD 5F C0      480             LDA   SETAN3          ; AN3 = TTL HI
FA78:20 BD FA      481             JSR   RESET.X         ; initialize other devices
FA7B:2C 10 C0      482             BIT   KBDSTRB         ; CLEAR KEYBOARD
FA7E:80 05   FA85  483             BRA   BEEPSKIP        ;+ Bell already beeped
FA80:EA            484             NOP                   ;+ align code
FA81:D8            485 NEWMON      CLD
FA82:20 3A FF      486             JSR   BELL            ; CAUSES DELAY IF KEY BOUNCES
FA85:AD F3 03      487 BEEPSKIP    LDA   SOFTEV+1        ;IS RESET HI
FA88:49 A5         488             EOR   #$A5            ;A FUNNY COMPLEMENT OF THE
FA8A:CD F4 03      489             CMP   PWREDUP         ; PWR UP BYTE ???
FA8D:D0 17   FAA6  490             BNE   PWRUP           ; NO SO PWRUP
FA8F:AD F2 03      491             LDA   SOFTEV          ; YES SEE IF COLD START
FA92:D0 0F   FAA3  492             BNE   NOFIX           ; HAS BEEN DONE YET?
FA94:A9 E0         493             LDA   #$E0            ; DOES SEV POINT AT BASIC?
FA96:CD F3 03      494             CMP   SOFTEV+1
FA99:D0 08   FAA3  495             BNE   NOFIX           ; YES SO REENTER SYSTEM
FA9B:A0 03         496 FIXSEV      LDY   #3              ; NO SO POINT AT WARM START
FA9D:8C F2 03      497             STY   SOFTEV          ; FOR NEXT RESET
FAA0:4C 00 E0      498             JMP   BASIC           ; AND DO THE COLD START
FAA3:              499 *
FAA3:6C F2 03      500 NOFIX       JMP   (SOFTEV)
FAA6:              501 *
FAA6:20 CA FC      502 PWRUP       JSR   COLDSTART       ;Trash memory, init ports
FAA9:        FAA9  503 SETPG3      EQU   *               ; SET PAGE 3 VECTORS
FAA9:A2 05         504             LDX   #5
FAAB:BD FC FA      505 SETPLP      LDA   PWRCON-1,X      ; WITH CNTRL B ADRS
FAAE:9D EF 03      506             STA   BRKV-1,X        ; OF CURRENT BASIC
FAB1:CA            507             DEX
FAB2:D0 F7   FAAB  508             BNE   SETPLP
FAB4:A9 C6         509             LDA   #$C6            ; LOAD HI SLOT +1
FAB6:80 5A   FB12  510             BRA   PWRUP2          ;branch around mnemonics
FAB8:              511 *
FAB8:              512 * Extension to MNEML (left mnemonics)
FAB8:              513 *
FAB8:8A            514             DFB   $8A             ;PHY
FAB9:8B            515             DFB   $8B             ;PLY
FABA:A5            516             DFB   $A5             ;STZ
FABB:AC            517             DFB   $AC             ;TRB
FABC:00            518             DFB   $00             ;???
FABD:              519 *
FABD:              520 * This extension to the monitor reset routine ($FA62)
FABD:              521 * checks for apple keys.  If both are pressed, it goes
FABD:              522 * into an exerciser mode.  If the open apple key only is
FABD:              523 * pressed, memory is selectively trashed and a cold start
FABD:              524 * is done.
```

```
FABD:                   525 *
FABD:A9 FF              526 RESET.X   LDA   #$FF
FABF:8D FB 04           527           STA   VMODE          ;initialize mode
FAC2:20 3A FF           528           JSR   BELL           ;+ Need bell delay for 3.5" drive
FAC5:20 F8 C5           529           JSR   PCNVRST        ;+ Reset protocol converter
FAC8:0E 62 C0           530           ASL   BUTN1
FACB:2C 61 C0           531           BIT   BUTN0
FACE:10 5E   FB2E       532           BPL   RTS2D
FAD0:90 D4   FAA6       533           BCC   PWRUP          ;open apple only, reboot
FAD2:4C C1 C7           534           JMP   BANGER         ;both apples, exercise 'er
FAD5:EA                 535           NOP                   ;+ align code
FAD6:EA                 536           NOP                   ;+
FAD7:20 8E FD           537 REGDSP    JSR   CROUT           ;DISPLAY USER REG CONTENTS
FADA:A9 44              538 RGDSP1    LDA   #$44            ;WITH LABELS
FADC:85 40              539           STA   A3L             ;Memory state now printed
FADE:A9 00              540           LDA   #$00
FAE0:85 41              541           STA   A3H
FAE2:A2 FA              542           LDX   #$FA
FAE4:A9 A0              543 RDSP1     LDA   #$A0
FAE6:20 ED FD           544           JSR   COUT
FAE9:BD 9A CE           545           LDA   RTBL-$FA,X
FAEC:20 ED FD           546           JSR   COUT
FAEF:A9 BD              547           LDA   #$BD
FAF1:20 ED FD           548           JSR   COUT
FAF4:B5 4A              549           LDA   ACC+5,X
FAF6:80 0A   FB02       550           BRA   RGDSP2         ;make room for mnemonics
FAF8:                   551 *
FAF8:                   552 * Right half of new mnemonics, indexed from MNEMR
FAF8:                   553 *
FAF8:74                 554           DFB   $74            ;PHY
FAF9:74                 555           DFB   $74            ;PLY
FAFA:76                 556           DFB   $76            ;STZ
FAFB:C6                 557           DFB   $C6            ;TRB
FAFC:00                 558           DFB   $00            ;???
FAFD:                   559 *
FAFD:59 FA              560 PWRCON    DW    OLDBRK
FAFF:00 E0 45           561           DFB   $00,$E0,$45
FB02:                   562 *
FB02:20 DA FD           563 RGDSP2    JSR   PRBYTE
FB05:E8                 564           INX
FB06:30 DC   FAE4       565           BMI   RDSP1
FB08:60                 566           RTS
FB09:                   567 *
FB09:C1 F0 F0 EC        568 TITLE     ASC   'Apple         ]['
FB11:C4                 569           DFB   $C4            ;optional filler
FB12:                   570 *
FB12:86 00              571 PWRUP2    STX   LOC0            ; SETPG3 MUST RETURN X=0
FB14:85 01              572           STA   LOC1            ; SET PTR H
FB16:20 60 FB           573           JSR   APPLEII        ;Display our banner...
FB19:6C 00 00           574           JMP   (LOC0)         ;JUMP $C600
FB1C:00                 575           BRK
FB1D:00                 576           BRK
FB1E:                   577 *
FB1E:4C 00 C9           578 PREAD     JMP   MPADDLE        ;read mouse paddle
FB21:A0 00              579           LDY   #$00            ;INIT COUNT
FB23:EA                 580           NOP                   ;COMPENSATE FOR 1ST COUNT
FB24:EA                 581           NOP
FB25:BD 64 C0           582 PREAD2    LDA   PADDL0,X        ;COUNT Y-REG EVERY 12 USEC.
```

```
FB28:10 04    FB2E   583            BPL    RTS2D
FB2A:C8              584            INY
FB2B:D0 F8    FB25   585            BNE    PREAD2          ;EXIT AT 255 MAX
FB2D:88              586            DEY
FB2E:60              587 RTS2D      RTS
FB2F:                 49            INCLUDE AUTOST2
```

```
FB2F:                    2 *
FB2F:A9 00               3 INIT      LDA   #$00           ;CLR STATUS FOR DEBUG SOFTWARE
FB31:85 48               4           STA   STATUS
FB33:AD 56 C0            5           LDA   LORES
FB36:AD 54 C0            6           LDA   TXTPAGE1       ;INIT VIDEO MODE
FB39:AD 51 C0            7 SETTXT    LDA   TXTSET         ;SET FOR TEXT MODE
FB3C:A9 00               8           LDA   #$00           ;FULL SCREEN WINDOW
FB3E:F0 0B   FB4B        9           BEQ   SETWND
FB40:AD 50 C0           10 SETGR     LDA   TXTCLR         ;SET FOR GRAPHICS MODE
FB43:AD 53 C0           11           LDA   MIXSET         ;LOWER 4 LINES AS TEXT WINDOW
FB46:20 36 F8           12           JSR   CLRTOP
FB49:A9 14              13           LDA   #$14
FB4B:85 22              14 SETWND    STA   WNDTOP        ;SET WINDOW
FB4D:EA                 15           NOP
FB4E:EA                 16           NOP
FB4F:20 0A CE           17           JSR   WNDREST        ;40/80 column width
FB52:80 05   FB59       18           BRA   VTAB23
FB54:                   19 *
FB54:09 80              20 DOCTL     ORA   #$80           ;controls need high bit
FB56:4C 54 CD           21           JMP   CTLCHAR0       ;execute control char
FB59:                   22 *
FB59:A9 17              23 VTAB23    LDA   #$17           ;VTAB TO ROW 23
FB5B:85 25              24 TABV      STA   CV             ;VTABS TO ROW IN A-REG
FB5D:4C 22 FC           25           JMP   VTAB           ;don't set OURCV!!
FB60:                   26 *
FB60:20 58 FC           27 APPLEII   JSR   HOME           ;CLEAR THE SCRN
FB63:A0 09              28           LDY   #9
FB65:B9 BA C5           29 STITLE    LDA   APPLE2C-1,Y    ;GET A CHAR
FB68:99 0D 04           30           STA   LINE1+13,Y     ;PUT IT AT TOP CENTER OF SCREEN
FB6B:88                 31           DEY
FB6C:D0 F7   FB65       32           BNE   STITLE
FB6E:60                 33           RTS
FB6F:                   34 *
FB6F:AD F3 03           35 SETPWRC   LDA   SOFTEV+1       ;ROUTINE TO CALCULATE THE 'FUNNY
FB72:49 A5              36           EOR   #$A5           ;COMPLEMENT' FOR THE RESET VECTOR
FB74:8D F4 03           37           STA   PWREDUP
FB77:60                 38           RTS
FB78:                   39 *
FB78:        FB78       40 VIDWAIT   EQU   *              ;CHECK FOR A PAUSE (CONTROL-S).
FB78:C9 8D              41           CMP   #$8D           ;ONLY WHEN I HAVE A CR
FB7A:D0 18   FB94       42           BNE   NOWAIT         ;NOT SO, DO REGULAR
FB7C:AC 00 C0           43           LDY   KBD            ;IS KEY PRESSED?
FB7F:10 13   FB94       44           BPL   NOWAIT         ;NO.
FB81:C0 93              45           CPY   #$93           ;YES -- IS IT CTRL-S?
FB83:D0 0F   FB94       46           BNE   NOWAIT         ;NOPE - IGNORE
FB85:2C 10 C0           47           BIT   KBDSTRB        ;CLEAR STROBE
FB88:AC 00 C0           48 KBDWAIT   LDY   KBD            ;WAIT TILL NEXT KEY TO RESUME
FB8B:10 FB   FB88       49           BPL   KBDWAIT        ;WAIT FOR KEYPRESS
FB8D:C0 83              50           CPY   #$83           ;IS IT CONTROL-C?
FB8F:F0 03   FB94       51           BEQ   NOWAIT         ;YES, SO LEAVE IT
FB91:2C 10 C0           52           BIT   KBDSTRB        ;CLR STROBE
FB94:2C 7B 06           53 NOWAIT    BIT   VFACTV        ;is video firmware active?
FB97:30 64   FBFD       54           BMI   VIDOUT         ;=>no, do normal 40 column
FB99:89 60              55           BIT   #$60           ;is it a control?
FB9B:F0 B7   FB54       56           BEQ   DOCTL          ;=>yes, do it
FB9D:20 B8 C3           57           JSR   STORCH         ;print w/inverse mask
FBA0:EE 7B 05           58 NEWADV    INC   OURCH          ;advance cursor
FBA3:AD 7B 05           59           LDA   OURCH          ;and update others
```

```
FBA6:2C 1F C0        60            BIT   RD80VID       ;but only if not 80 columns
FBA9:30 05   FBB0    61            BMI   NEWADV1       ;=>80 columns, leav'em
FBAB:8D 7B 04        62            STA   OLDCH
FBAE:85 24           63            STA   CH
FBB0:80 46   FBF8    64 NEWADV1    BRA   ADV2          ;check for CR
FBB2:                65 *
FBB2:EA              66            NOP
FBB3:                67 *
FBB3:06              68 F8VERSION  DFB   GOODF8        ;//e, chels ID byte
FBB4:                69 *
FBB4:10 06   FBBC    70 DOCOUT1    BPL   DCX           ;=>video firmware active, no mask
FBB6:C9 A0           71            CMP   #$A0          ;is it control char?
FBB8:90 02   FBBC    72            BCC   DCX           ;=>yes, no mask
FBBA:25 32           73            AND   INVFLG        ;else apply inverse mask
FBBC:4C F6 FD        74 DCX        JMP   COUTZ         ;and print character
FBBF:00              75            BRK
FBC0:                76 *
FBC0:00              77            DFB   $00           ;chels ID byte
FBC1:                78 *
FBC1:48              79 BASCALC    PHA                 ;CALC BASE ADDR IN BASL,H
FBC2:4A              80            LSR   A             ;FOR GIVEN LINE NO.
FBC3:29 03           81            AND   #$03           ; 0<=LINE NO.<=$17
FBC5:09 04           82            ORA   #$04           ;ARG=000ABCDE, GENERATE
FBC7:85 29           83            STA   BASH           ; BASH=000001CD
FBC9:68              84            PLA                  ; AND
FBCA:29 18           85            AND   #$18           ; BASL=EABAB000
FBCC:90 02   FBD0    86            BCC   BASCLC2
FBCE:69 7F           87            ADC   #$7F
FBD0:85 28           88 BASCLC2    STA   BASL
FBD2:0A              89            ASL   A
FBD3:0A              90            ASL   A
FBD4:05 28           91            ORA   BASL
FBD6:85 28           92            STA   BASL
FBD8:60              93            RTS
FBD9:                94 *
FBD9:C9 87           95 CHKBELL    CMP   #$87           ;BELL CHAR? (CONTROL-G)
FBDB:D0 12   FBEF    96            BNE   RTS2B          ; NO, RETURN.
FBDD:A9 40           97 BELL1      LDA   #$40          ; YES...
FBDF:20 A8 FC        98            JSR   WAIT          ;DELAY .01 SECONDS
FBE2:A0 C0           99            LDY   #$C0
FBE4:A9 0C          100 BELL2      LDA   #$0C          ;TOGGLE SPEAKER AT 1 KHZ
FBE6:20 A8 FC       101            JSR   WAIT          ; FOR .1 SEC.
FBE9:AD 30 C0       102            LDA   SPKR
FBEC:88             103            DEY
FBED:D0 F5   FBE4   104            BNE   BELL2
FBEF:60             105 RTS2B      RTS
FBF0:               106 *
FBF0:A4 24          107 STORADV    LDY   CH            ;get 40 column position
FBF2:91 28          108            STA   (BASL),Y      ;and store
FBF4:E6 24          109 ADVANCE    INC   CH            ;increment cursor
FBF6:A5 24          110            LDA   CH
FBF8:C5 21          111 ADV2       CMP   WNDWDTH        ;BEYOND WINDOW WIDTH?
FBFA:B0 66   FC62   112            BCS   CR             ; YES, CR TO NEXT LINE.
FBFC:60             113 RTS3       RTS                  ; NO, RETURN.
FBFD:               114 *
FBFD:C9 A0          115 VIDOUT     CMP   #$A0           ;CONTROL CHAR?
FBFF:B0 EF   FBF0   116            BCS   STORADV        ; NO, OUTPUT IT.
FC01:A8             117            TAY                  ;INVERSE VIDEO?
```

```
FC02:10 EC    FBF0  118           BPL   STORADV        ; YES, OUTPUT IT.
FC04:C9 8D          119 VIDOUT1   CMP   #$8D          ;CR?
FC06:F0 6B    FC73  120           BEQ   NEWCR         ;Yes, use new routine
FC08:C9 8A          121           CMP   #$8A          ;LINE FEED?
FC0A:F0 5A    FC66  122           BEQ   LF            ; IF SO, DO IT.
FC0C:C9 88          123           CMP   #$88          ;BACK SPACE? (CONTROL-H)
FC0E:D0 C9    FBD9  124           BNE   CHKBELL       ; NO, CHECK FOR BELL.
FC10:20 E2 FE       125 BS        JSR   DECCH         ;decrement all cursor H indices
FC13:10 E7    FBFC  126           BPL   RTS3          ;IF POSITIVE, OK; ELSE MOVE UP.
FC15:A5 21          127           LDA   WNDWDTH       ;get window width,
FC17:20 EB FE       128           JSR   WDTHCH        ;and set CH's to WNDWDTH-1
FC1A:A5 22          129 UP        LDA   WNDTOP        ;CURSOR V INDEX
FC1C:C5 25          130           CMP   CV
FC1E:B0 DC    FBFC  131           BCS   RTS3          ;top line, exit
FC20:C6 25          132           DEC   CV            ;not top, go up one
FC22:               133 *
FC22:80 62    FC86  134 VTAB      BRA   NEWVTAB       ;go update OURCV
FC24:20 C1 FB       135 VTABZ     JSR   BASCALC       ;calculate the base address
FC27:A5 20          136           LDA   WNDLFT        ;get the left window edge
FC29:2C 1F C0       137           BIT   RD80VID       ;80 columns?
FC2C:10 02    FC30  138           BPL   VTAB40        ;=>no, left edge ok
FC2E:4A             139           LSR   A             ;divide width by 2
FC2F:18             140           CLC                 ;prepare to add
FC30:65 28          141 VTAB40    ADC   BASL          ;add width to base
FC32:85 28          142           STA   BASL
FC34:60             143 RTS4      RTS
FC35:               144 *
FC35:               145 * NEWOPS translates the opcode in the Y register
FC35:               146 * to a mnemonic table index and returns with Z=1.
FC35:               147 * If Y is not a new opcode, Z=0.
FC35:               148 *
FC35:98             149 NEWOPS    TYA                 ;get the opcode
FC36:A2 16          150           LDX   #NUMOPS       ;check through new opcodes
FC38:DD FE FE       151 NEWOP1    CMP   OPTBL,X       ;does it match?
FC3B:F0 43    FC80  152           BEQ   GETINDX       ;=>yes, get new index
FC3D:CA             153           DEX
FC3E:10 F8    FC38  154           BPL   NEWOP1        ;else check next one
FC40:60             155           RTS                 ;not found, exit with BNE
FC41:               156 *
FC41:00             157           BRK
FC42:               158 *
FC42:80 19    FC5D  159 CLREOP    BRA   CLREOP1        ;ESC F IS CLR TO END OF PAGE
FC44:A5 25          160 CLREOP2   LDA   CV
FC46:48             161 CLEOP1    PHA                  ;SAVE CURRENT LINE NO. ON STACK
FC47:20 24 FC       162           JSR   VTABZ          ;CALC BASE ADDRESS
FC4A:20 9E FC       163           JSR   CLEOLZ         ;CLEAR TO EOL. (SETS CARRY)
FC4D:A0 00          164           LDY   #$00           ;CLEAR FROM H INDEX=0 FOR REST
FC4F:68             165           PLA                  ;INCREMENT CURRENT LINE NO.
FC50:1A             166           INC   A
FC51:C5 23          167           CMP   WNDBTM         ;DONE TO BOTTOM OF WINDOW?
FC53:90 F1    FC46  168           BCC   CLEOP1         ; NO, KEEP CLEARING LINES.
FC55:B0 CB    FC22  169           BCS   VTAB           ; YES, TAB TO CURRENT LINE
FC57:00             170           BRK
FC58:               171 *
FC58:20 A5 CD       172 HOME      JSR   HOMECUR       ;move cursor home
FC5B:80 E7    FC44  173           BRA   CLREOP2       ;then clear to end of page
FC5D:               174 *
FC5D:20 9D CC       175 CLREOP1   JSR   GETCUR        ;load Y with proper CH
```

```
FC60:80 E2   FC44  176           BRA   CLREOP2       ;before clearing page
FC62:              177 *
FC62:80 0F   FC73  178 CR        BRA   NEWCR         ;only LF if not Pascal
FC64:00            179           BRK
FC65:00            180           BRK
FC66:              181 *
FC66:E6 25         182 LF        INC   CV             ;INCR CURSOR V. (DOWN 1 LINE)
FC68:A5 25         183           LDA   CV
FC6A:C5 23         184           CMP   WNDBTM         ;OFF SCREEN?
FC6C:90 1A   FC88  185           BCC   NEWVTABZ       ;set base+WNDLFT
FC6E:C6 25         186           DEC   CV             ;DECR CURSOR V. (BACK TO BOTTOM)
FC70:              187 *
FC70:4C 35 CB      188 SCROLL    JMP   SCROLLUP       ;scroll the screen
FC73:              189 *
FC73:20 E9 FE      190 NEWCR     JSR   CLRCH          ;set CH's to 0
FC76:2C FB 04      191           BIT   VMODE          ;is it Pascal?
FC79:10 0A   FC85  192           BPL   CRRTS          ;pascal, no LF
FC7B:20 44 FD      193           JSR   NOESCAPE       ;else clear escape mode
FC7E:80 E6   FC66  194           BRA   LF             ;then do LF
FC80:              195 *
FC80:BD 15 FF      196 GETINDX   LDA   INDX,X         ;lookup index for mnemonic
FC83:A0 00         197           LDY   #0             ;exit with BEQ
FC85:60            198 CRRTS     RTS
FC86:              199 *
FC86:A5 25         200 NEWVTAB   LDA   CV             ;update //e CV
FC88:8D FB 05      201 NEWVTABZ  STA   OURCV
FC8B:80 97   FC24  202           BRA   VTABZ          ;and calc base+WNDLFT
FC8D:              203 *
FC8D:20 9D CC      204 NEWCLREOL JSR   GETCUR         ;get current cursor
FC90:A9 A0         205 NEWCLEOLZ LDA   #$A0           ;get a blank
FC92:2C 7B 06      206           BIT   VFACTV         ;if video firmware active,
FC95:30 02   FC99  207           BMI   NEWC1          ;=>don't use inverse mask
FC97:25 32         208           AND   INVFLG
FC99:4C C2 CB      209 NEWC1     JMP   DOCLR          ;go do clear
FC9C:              210 *
FC9C:80 EF   FC8D  211 CLREOL    BRA   NEWCLREOL      ;get cursor and clear
FC9E:80 F0   FC90  212 CLEOLZ    BRA   NEWCLEOLZ      ;clear from Y
FCA0:              213 *
FCA0:A0 00         214 CLRLIN    LDY   #0             ;clear entire line
FCA2:80 EC   FC90  215           BRA   NEWCLEOLZ
FCA4:              216 *
FCA4:7C 2A CD      217 CTLDO     JMP   (CTLADR,X)     ;jump to proper routine
FCA7:              218 *
FCA7:EA            219           NOP
FCA8:              220 *
FCA8:38            221 WAIT      SEC
FCA9:48            222 WAIT2     PHA
FCAA:E9 01         223 WAIT3     SBC   #$01
FCAC:D0 FC   FCAA  224           BNE   WAIT3          ;1.0204 USEC
FCAE:68            225           PLA                  ;(13+2712*A+512*A*A)
FCAF:E9 01         226           SBC   #$01
FCB1:D0 F6   FCA9  227           BNE   WAIT2
FCB3:60            228 RTS6      RTS
FCB4:              229 *
FCB4:E6 42         230 NXTA4     INC   A4L            ;INCR 2-BYTE A4
FCB6:D0 02   FCBA  231           BNE   NXTA1          ; AND A1
FCB8:E6 43         232           INC   A4H
FCBA:A5 3C         233 NXTA1     LDA   A1L            ;INCR 2-BYTE A1.
```

```
FCBC:C5 3E          234             CMP   A2L             ; AND COMPARE TO A2
FCBE:A5 3D          235             LDA   A1H             ; (CARRY SET IF >=)
FCC0:E5 3F          236             SBC   A2H
FCC2:E6 3C          237             INC   A1L
FCC4:D0 02   FCC8   238             BNE   RTS4B
FCC6:E6 3D          239             INC   A1H
FCC8:60             240 RTS4B       RTS
FCC9:               241 *
FCC9:60             242 HEADR       RTS                   ;don't do it
FCCA:               243 *
FCCA:A0 B0          244 COLDSTART   LDY   #$B0            ;let it precess down
FCCC:64 3C          245             STZ   A1L
FCCE:A2 BF          246             LDX   #$BF            ;start from BFXX down
FCD0:86 3D          247 BLAST       STX   A1H
FCD2:A9 A0          248             LDA   #$A0            ;store blanks
FCD4:91 3C          249             STA   (A1L),Y
FCD6:88             250             DEY
FCD7:91 3C          251             STA   (A1L),Y
FCD9:CA             252             DEX                   ;back down to next page
FCDA:E0 01          253             CPX   #1              ;stay away from stack
FCDC:D0 F2   FCD0   254             BNE   BLAST           ;fall into COMINIT
FCDE:               255 *
FCDE:8D 01 C0       256             STA   SET80COL        ;init ALT screen holes
FCE1:AD 55 C0       257             LDA   TXTPAGE2        ;for serial and comm ports
FCE4:A2 88          258             LDX   #$88            ;C = 1 from CPX #1
FCE6:BD 8B CF       259 COM1        LDA   COMTBL-1,X      ;XFER from rom
FCE9:90 0A   FCF5   260             BCC   COM2            ;branch if defaults ok
FCEB:DD 77 04       261             CMP   $477,X          ;test for prior setup
FCEE:18             262             CLC                   ;branch if not valid
FCEF:D0 04   FCF5   263             BNE   COM2            ;If $4F8 & $4FF = TBL values
FCF1:E0 82          264             CPX   #$82
FCF3:90 06   FCFB   265             BCC   COM3
FCF5:9D 77 04       266 COM2        STA   $477,X
FCF8:CA             267             DEX                   ;move all 8...
FCF9:D0 EB   FCE6   268             BNE   COM1
FCFB:AD 54 C0       269 COM3        LDA   TXTPAGE1        ;restore switches
FCFE:8D 00 C0       270             STA   CLR80COL        ;to default states
FD01:60             271             RTS
FD02:EA             272             NOP                    ;+
FD03:EA             273             NOP
FD04:EA             274             NOP
FD05:EA             275             NOP
FD06:EA             276             NOP
FD07:EA             277             NOP
FD08:EA             278             NOP
FD09:EA             279             NOP
FD0A:EA             280             NOP
FD0B:EA             281             NOP
FD0C:               282 *
FD0C:A4 24          283 RDKEY       LDY   CH              ;get char at current position
FD0E:B1 28          284             LDA   (BASL),Y        ;for those who restore it
FD10:EA             285             NOP                   ;if a program controls input
FD11:EA             286             NOP                   ;hooks, no cursor may be displayed
FD12:EA             287             NOP
FD13:EA             288             NOP
FD14:EA             289             NOP
FD15:EA             290             NOP
FD16:EA             291             NOP
```

```
FD17:EA            292            NOP
FD18:              293 *
FD18:6C 38 00      294 KEYIN0     JMP   (KSWL)        ;GO TO USER KEY-IN
FD1B:              295 *
FD1B:91 28         296 KEYIN      STA   (BASL),Y      ;erase false images
FD1D:20 4C CC      297            JSR   SHOWCUR       ;display true cursor
FD20:20 70 CC      298 DONXTCUR   JSR   UPDATE        ;look for key, blink II cursor
FD23:10 FB   FD20  299            BPL   DONXTCUR      ;loop until keypress
FD25:48            300 GOTKEY     PHA                 ;save character
FD26:A9 08         301            LDA   #M.CTL        ;were escapes enabled?
FD28:2C FB 04      302            BIT   VMODE
FD2B:D0 1D   FD4A  303            BNE   NOESC2        ;=>no, there is no escape
FD2D:68            304            PLA                 ;yes, there may be a way out!!
FD2E:C9 9B         305            CMP   #ESC          ;escape?
FD30:D0 06   FD38  306            BNE   LOOKPICK      ;=>no escape
FD32:4C CC CC      307            JMP   NEWESC        ;=>go do escape sequence
FD35:              308 *
FD35:4C ED CC      309 RDCHAR     JMP   ESCRDKEY      ;do RDKEY with escapes
FD38:              310 *
FD38:2C 7B 06      311 LOOKPICK   BIT   VFACTV        ;only process f.arrow
FD3B:30 07   FD44  312            BMI   NOESCAPE      ;if video firmware is active
FD3D:C9 95         313            CMP   #PICK         ;was it PICK? (->,CTL-U)
FD3F:D0 03   FD44  314            BNE   NOESCAPE      ;no, just return
FD41:20 1D CC      315            JSR   PICKY         ;yes, pick the character
FD44:              316 *
FD44:              317 * NOESCAPE is used by GETCOUT too.
FD44:              318 *
FD44:48            319 NOESCAPE   PHA                 ;save it
FD45:A9 08         320 NOESC1     LDA   #M.CTL        ;disable escape sequences
FD47:0C FB 04      321            TSB   VMODE         ;and enable controls
FD4A:68            322 NOESC2     PLA                 ;by setting M.CTL
FD4B:60            323            RTS
FD4C:              324 *
FD4C:EA            325            NOP
FD4D:              326 *
FD4D:20 A6 C3      327 NOTCR      JSR   GETCOUT       ;disable controls and print
FD50:C9 88         328            CMP   #$88          ;CHECK FOR EDIT KEYS
FD52:F0 1D   FD71  329            BEQ   BCKSPC        ;  - BACKSPACE
FD54:C9 98         330            CMP   #$98
FD56:F0 0A   FD62  331            BEQ   CANCEL        ;  - CONTROL-X
FD58:E0 F8         332            CPX   #$F8
FD5A:90 03   FD5F  333            BCC   NOTCR1        ;MARGIN?
FD5C:20 3A FF      334            JSR   BELL          ; YES, SOUND BELL
FD5F:E8            335 NOTCR1     INX                 ;ADVANCE INPUT INDEX
FD60:D0 13   FD75  336            BNE   NXTCHAR
FD62:A9 DC         337 CANCEL     LDA   #$DC          ;BACKSLASH AFTER CANCELLED LINE
FD64:20 A6 C3      338            JSR   GETCOUT
FD67:20 8E FD      339 GETLNZ     JSR   CROUT         ;OUTPUT 'CR'
FD6A:A5 33         340 GETLN      LDA   PROMPT        ;OUTPUT PROMPT CHAR
FD6C:20 ED FD      341            JSR   COUT
FD6F:A2 01         342 GETLN1     LDX   #$01          ;INIT INPUT INDEX
FD71:8A            343 BCKSPC     TXA
FD72:F0 F3   FD67  344            BEQ   GETLNZ        ;WILL BACKSPACE TO 0
FD74:CA            345            DEX
FD75:20 ED CC      346 NXTCHAR    JSR   ESCRDKEY      ;do new RDCHAR (allow escapes)
FD78:C9 95         347            CMP   #PICK         ;USE SCREEN CHAR
FD7A:D0 08   FD84  348            BNE   ADDINP        ; FOR CONTROL-U
FD7C:20 1D CC      349            JSR   PICKY         ;lift char from screen
```

```
FD7F:EA          350          NOP
FD80:EA          351          NOP
FD81:EA          352          NOP                ;no upshifting needed
FD82:EA          353          NOP
FD83:EA          354          NOP
FD84:9D 00 02    355 ADDINP   STA   IN,X         ;ADD TO INPUT BUFFER
FD87:C9 8D       356          CMP   #$8D
FD89:D0 C2  FD4D 357          BNE   NOTCR
FD8B:20 9C FC    358 CROUT1   JSR   CLREOL       ;CLR TO EOL IF CR
FD8E:A9 8D       359 CROUT    LDA   #$8D
FD90:D0 5B  FDED 360          BNE   COUT         ;(ALWAYS)
FD92:            361 *
FD92:A4 3D       362 PRA1     LDY   A1H          ;PRINT CR,A1 IN HEX
FD94:A6 3C       363          LDX   A1L
FD96:20 8E FD    364 PRYX2    JSR   CROUT
FD99:20 40 F9    365          JSR   PRNTYX
FD9C:A0 00       366          LDY   #$00
FD9E:A9 AD       367          LDA   #$AD         ;PRINT '-'
FDA0:4C ED FD    368          JMP   COUT
FDA3:            369 *
FDA3:A5 3C       370 XAM8     LDA   A1L
FDA5:09 07       371          ORA   #$07         ;SET TO FINISH AT
FDA7:85 3E       372          STA   A2L          ; MOD 8=7
FDA9:A5 3D       373          LDA   A1H
FDAB:85 3F       374          STA   A2H
FDAD:A5 3C       375 MOD8CHK  LDA   A1L
FDAF:29 07       376          AND   #$07
FDB1:D0 03  FDB6 377          BNE   DATAOUT
FDB3:20 92 FD    378 XAM      JSR   PRA1
FDB6:A9 A0       379 DATAOUT  LDA   #$A0
FDB8:20 ED FD    380          JSR   COUT         ;OUTPUT BLANK
FDBB:B1 3C       381          LDA   (A1L),Y
FDBD:20 DA FD    382          JSR   PRBYTE       ;OUTPUT BYTE IN HEX
FDC0:20 BA FC    383          JSR   NXTA1
FDC3:90 E8  FDAD 384          BCC   MOD8CHK      ;NOT DONE YET. GO CHECK MOD 8
FDC5:60          385 RTS4C    RTS                ;DONE.
FDC6:            386 *
FDC6:4A          387 XAMPM    LSR   A            ;DETERMINE IF MONITOR MODE IS
FDC7:90 EA  FDB3 388          BCC   XAM          ; EXAMINE, ADD OR SUBTRACT
FDC9:4A          389          LSR   A
FDCA:4A          390          LSR   A
FDCB:A5 3E       391          LDA   A2L
FDCD:90 02  FDD1 392          BCC   ADD
FDCF:49 FF       393          EOR   #$FF         ;FORM 2'S COMPLEMENT FOR SUBTRACT.
FDD1:65 3C       394 ADD      ADC   A1L
FDD3:48          395          PHA
FDD4:A9 BD       396          LDA   #$BD         ;PRINT '=', THEN RESULT
FDD6:20 ED FD    397          JSR   COUT
FDD9:68          398          PLA
FDDA:            399 *
FDDA:48          400 PRBYTE   PHA                ;PRINT BYTE AS 2 HEX DIGITS
FDDB:4A          401          LSR   A            ; (DESTROYS A-REG)
FDDC:4A          402          LSR   A
FDDD:4A          403          LSR   A
FDDE:4A          404          LSR   A
FDDF:20 E5 FD    405          JSR   PRHEXZ
FDE2:68          406          PLA
FDE3:            407 *
```

```
FDE3:29 0F          408 PRHEX     AND   #$0F          ;PRINT HEX DIGIT IN A-REG
FDE5:09 B0          409 PRHEXZ    ORA   #$B0          ;LSBITS ONLY.
FDE7:C9 BA          410           CMP   #$BA
FDE9:90 02   FDED   411           BCC   COUT
FDEB:69 06          412           ADC   #$06
FDED:               413 *
FDED:6C 36 00       414 COUT      JMP   (CSWL)        ;VECTOR TO USER OUTPUT ROUTINE
FDF0:               415 *
FDF0:2C 7B 06       416 COUT1     BIT   VFACTV        ;video firmware active?
FDF3:4C B4 FB       417           JMP   DOCOUT1       ;mask II mode characters
FDF6:84 35          418 COUTZ     STY   YSAV1         ;SAVE Y-REG
FDF8:48             419           PHA                 ;SAVE A -REG
FDF9:20 78 FB       420           JSR   VIDWAIT       ;OUTPUT CHR AND CHECK FOR CTRL-S
FDFC:68             421           PLA                 ;RESTORE A-REG
FDFD:A4 35          422           LDY   YSAV1         ;AND Y-REG
FDFF:60             423           RTS                 ;RETURN TO SENDER...
FE00:               424 *
FE00:C6 34          425 BL1       DEC   YSAV
FE02:F0 9F   FDA3   426           BEQ   XAM8
FE04:               427 *
FE04:CA             428 BLANK     DEX                  ;BLANK TO MON
FE05:D0 16   FE1D   429           BNE   SETMDZ         ;AFTER BLANK
FE07:C9 BA          430           CMP   #$BA           ;DATA STORE MODE?
FE09:D0 BB   FDC6   431           BNE   XAMPM          ; NO; XAM, ADD, OR SUBTRACT.
FE0B:               432 *
FE0B:85 31          433 STOR      STA   MODE           ;KEEP IN STORE MODE
FE0D:A5 3E          434           LDA   A2L
FE0F:91 40          435           STA   (A3L),Y        ;STORE AS LOW BYTE AT (A3)
FE11:E6 40          436           INC   A3L
FE13:D0 02   FE17   437           BNE   RTS5           ;INCR A3, RETURN.
FE15:E6 41          438           INC   A3H
FE17:60             439 RTS5      RTS
FE18:               440 *
FE18:A4 34          441 SETMODE   LDY   YSAV           ;SAVE CONVERTED ':', '+',
FE1A:B9 FF 01       442           LDA   IN-1,Y         ; '-', '.' AS MODE
FE1D:85 31          443 SETMDZ    STA   MODE
FE1F:60             444           RTS
FE20:               445 *
FE20:A2 01          446 LT        LDX   #$01
FE22:B5 3E          447 LT2       LDA   A2L,X          ;COPY A2 (2 BYTES) TO
FE24:95 42          448           STA   A4L,X          ; A4 AND A5
FE26:95 44          449           STA   A5L,X
FE28:CA             450           DEX
FE29:10 F7   FE22   451           BPL   LT2
FE2B:60             452           RTS
FE2C:               453 *
FE2C:B1 3C          454 MOVE      LDA   (A1L),Y        ;MOVE (A1) THRU (A2) TO (A4)
FE2E:91 42          455           STA   (A4L),Y
FE30:20 B4 FC       456           JSR   NXTA4
FE33:90 F7   FE2C   457           BCC   MOVE
FE35:60             458           RTS
FE36:               459 *
FE36:B1 3C          460 VERIFY    LDA   (A1L),Y        ;VERIFY (A1) THRU (A2)
FE38:D1 42          461           CMP   (A4L),Y        ; WITH (A4)
FE3A:F0 1C   FE58   462           BEQ   VFYOK
FE3C:20 92 FD       463           JSR   PRA1
FE3F:B1 3C          464           LDA   (A1L),Y
FE41:20 DA FD       465           JSR   PRBYTE
```

```
FE44:A9 A0          466            LDA   #$A0
FE46:20 ED FD       467            JSR   COUT
FE49:A9 A8          468            LDA   #$A8
FE4B:20 ED FD       469            JSR   COUT
FE4E:B1 42          470            LDA   (A4L),Y
FE50:20 DA FD       471            JSR   PRBYTE
FE53:A9 A9          472            LDA   #$A9
FE55:20 ED FD       473            JSR   COUT
FE58:20 B4 FC       474 VFYOK      JSR   NXTA4
FE5B:90 D9   FE36   475            BCC   VERIFY
FE5D:60             476            RTS
FE5E:               477 *
FE5E:20 75 FE       478 LIST       JSR   A1PC            ;MOVE A1 (2 BYTES) TO
FE61:A9 14          479            LDA   #$14            ; PC IF SPEC'D AND
FE63:48             480 LIST2      PHA                    ;+DISASSEMBLE 20 INSTRUCTIONS.
FE64:20 C4 C5       481            JSR   SHOWINST        ;+Display a line
FE67:68             482            PLA
FE68:3A             483            DEC   A               ;+Count down
FE69:D0 F8   FE63   484            BNE   LIST2
FE6B:60             485            RTS
FE6C:               486 *
FE6C:4C 86 C9       487 MINI       JMP   GETINST1        ;+Go to the mini assembler
FE6F:C6 34          488 TRACE      DEC   YSAV            ;+Stay on T for trace
FE71:4C 43 CA       489 STEPZ      JMP   STEP            ;+Off to the step routine
FE74:         0001  490            ds    $FE75-*,0       ;+Extra bytes
FE75:               491 *
FE75:8A             492 A1PC       TXA                   ;IF USER SPECIFIED AN ADDRESS,
FE76:F0 07   FE7F   493            BEQ   A1PCRTS         ; COPY IT FROM A1 TO PC.
FE78:B5 3C          494 A1PCLP     LDA   A1L,X           ;YEP, SO COPY IT.
FE7A:95 3A          495            STA   PCL,X
FE7C:CA             496            DEX
FE7D:10 F9   FE78   497            BPL   A1PCLP
FE7F:60             498 A1PCRTS    RTS
FE80:               499 *
FE80:A0 3F          500 SETINV     LDY   #$3F            ;SET FOR INVERSE VID
FE82:D0 02   FE86   501            BNE   SETIFLG         ; VIA COUT1
FE84:A0 FF          502 SETNORM    LDY   #$FF            ;SET FOR NORMAL VID
FE86:84 32          503 SETIFLG    STY   INVFLG
FE88:60             504            RTS
FE89:               505 *
FE89:A9 00          506 SETKBD     LDA   #$00            ;DO 'IN#0'
FE8B:85 3E          507 INPORT     STA   A2L             ;DO 'IN#AREG'
FE8D:A2 38          508 INPRT      LDX   #KSWL
FE8F:A0 1B          509            LDY   #KEYIN
FE91:D0 08   FE9B   510            BNE   IOPRT
FE93:               511 *
FE93:A9 00          512 SETVID     LDA   #$0             ;DO 'PR#0'
FE95:85 3E          513 OUTPORT    STA   A2L             ;DO 'PR#AREG'
FE97:A2 36          514 OUTPRT     LDX   #CSWL
FE99:A0 F0          515            LDY   #COUT1
FE9B:A5 3E          516 IOPRT      LDA   A2L
FE9D:29 0F          517            AND   #$0F
FE9F:D0 06   FEA7   518            BNE   NOTPRT0         ;not slot 0
FEA1:C0 1B          519            CPY   #KEYIN          ;Continue if KEYIN
FEA3:F0 39   FEDE   520            BEQ   IOPRT1
FEA5:80 1B   FEC2   521            BRA   OPRT0           ;=>do PR#0
FEA7:09 C0          522 NOTPRT0    ORA   #<IOADR
FEA9:A0 00          523            LDY   #$00
```

```
FEAB:94 00          524 IOPRT2    STY   LOC0,X
FEAD:95 01          525           STA   LOC1,X
FEAF:60             526           RTS
FEB0:               527 *
FEB0:4C 00 E0       528 XBASIC    JMP   BASIC          ;TO BASIC, COLD START
FEB3:               529 *
FEB3:4C 03 E0       530 BASCONT   JMP   BASIC2         ;TO BASIC, WARM START
FEB6:               531 *
FEB6:20 75 FE       532 GO        JSR   A1PC           ;ADDR TO PC IF SPECIFIED
FEB9:20 3F FF       533           JSR   RESTORE        ;RESTORE FAKE REGISTERS
FEBC:6C 3A 00       534           JMP   (PCL)          ; AND GO!
FEBF:               535 *
FEBF:4C D7 FA       536 REGZ      JMP   REGDSP         ;GO DISPLAY REGISTERS
FEC2:               537 *
FEC2:3A             538 OPRT0     DEC   A              ;Need $FF
FEC3:8D FB 07       539           STA   CURSOR         ;set checkerboard cursor
FEC6:A9 F7          540           LDA   #$FF-M.CTL     ;reset mode
FEC8:80 04   FECE   541           BRA   DOPR0
FECA:               542 *
FECA:4C F8 03       543 USR       JMP   USRADR         ;JUMP TO CONTROL-Y VECTOR IN RAM
FECD:               544 *
FECD:60             545 WRITE     RTS                  ;Tape write not needed
FECE:               546 *
FECE:8D 7B 06       547 DOPR0     STA   VFACTV         ;say video firmware inactive
FED1:8D 0E C0       548           STA   CLRALTCHAR     ;switch in normal char set
FED4:0C FB 04       549           TSB   VMODE          ;don't change M.CTL
FED7:DA             550           PHX                  ;save X and Y
FED8:5A             551           PHY                  ;for rest of PR#0
FED9:20 CD CD       552           JSR   CHK80          ;convert to 40 if needed
FEDC:7A             553           PLY
FEDD:FA             554           PLX
FEDE:A9 FD          555 IOPRT1    LDA   #<COUT1        ;set I/O page
FEE0:80 C9   FEAB   556           BRA   IOPRT2         ;=>go set output hook
FEE2:               557 *
FEE2:               558 * DECCH decrements the current cursor
FEE2:               559 * CLRCH sets all cursors to 0
FEE2:               560 * SETCUR sets cursors to value in Acc.
FEE2:               561 * See explanatory note with GETCUR
FEE2:               562 *
FEE2:5A             563 DECCH     PHY                  ;(from $FC10)
FEE3:20 9D CC       564           JSR   GETCUR         ;get current CH
FEE6:88             565           DEY                  ;decrement it
FEE7:80 05   FEEE   566           BRA   SETCUR1        ;go update cursors
FEE9:               567 *
FEE9:A9 01          568 CLRCH     LDA   #1             ;set all cursors to 0
FEEB:3A             569 WDTHCH    DEC   A              ;dec window width (from $FC17)
FEEC:5A             570 SETCUR    PHY                  ;save Y
FEED:A8             571           TAY                  ;need value in Y
FEEE:20 AD CC       572 SETCUR1   JSR   GETCUR2        ;save new CH
FEF1:7A             573           PLY                  ;restore Y
FEF2:AD 7B 05       574           LDA   OURCH          ;and get new CH into acc
FEF5:60             575           RTS                  ;(Need LDA to set flags)
FEF6:               576 *
FEF6:20 00 FE       577 CRMON     JSR   BL1            ;HANDLE CR AS BLANK
FEF9:68             578           PLA                  ; THEN POP STACK
FEFA:68             579           PLA                  ; AND RETURN TO MON
FEFB:D0 6C   FF69   580           BNE   MONZ           ;(ALWAYS)
FEFD:               581 *
```

```
FEFD:60                582 READ      RTS                     ;Tape read not needed
FEFE:                  583 *
FEFE:                  584 * OPTBL is a table containing the new opcodes that
FEFE:                  585 * wouldn't fit into the existing lookup table.
FEFE:                  586 *
FEFE:12                587 OPTBL     DFB   $12               ;ORA (ZPAG)
FEFF:14                588           DFB   $14               ;TRB ZPAG
FF00:1A                589           DFB   $1A               ;INC A
FF01:1C                590           DFB   $1C               ;TRB ABS
FF02:32                591           DFB   $32               ;AND (ZPAG)
FF03:34                592           DFB   $34               ;BIT ZPAG,X
FF04:3A                593           DFB   $3A               ;DEC A
FF05:3C                594           DFB   $3C               ;BIT ABS,X
FF06:52                595           DFB   $52               ;EOR (ZPAG)
FF07:5A                596           DFB   $5A               ;PHY
FF08:64                597           DFB   $64               ;STZ ZPAG
FF09:72                598           DFB   $72               ;ADC (ZPAG)
FF0A:74                599           DFB   $74               ;STZ ZPAG,X
FF0B:7A                600           DFB   $7A               ;PLY
FF0C:7C                601           DFB   $7C               ;JMP (ABS,X)
FF0D:89                602           DFB   $89               ;BIT IMM
FF0E:92                603           DFB   $92               ;STA (ZPAG)
FF0F:9C                604           DFB   $9C               ;STZ ABS
FF10:9E                605           DFB   $9E               ;STZ ABS,X
FF11:B2                606           DFB   $B2               ;LDA (ZPAG)
FF12:D2                607           DFB   $D2               ;CMP (ZPAG)
FF13:F2                608           DFB   $F2               ;SBC (ZPAG)
FF14:FC                609           DFB   $FC               ;??? (the unknown opcode)
FF15:          0016    610 NUMOPS    EQU   *-OPTBL-1         ;number of bytes to check
FF15:                  611 *
FF15:                  612 * INDX contains pointers to the mnemonics for each of
FF15:                  613 * the opcodes in OPTBL.  Pointers with BIT 7
FF15:                  614 * set indicate extensions to MNEML or MNEMR.
FF15:                  615 *
FF15:38                616 INDX      DFB   $38
FF16:FB                617           DFB   $FB
FF17:37                618           DFB   $37
FF18:FB                619           DFB   $FB
FF19:39                620           DFB   $39
FF1A:21                621           DFB   $21
FF1B:36                622           DFB   $36
FF1C:21                623           DFB   $21
FF1D:3A                624           DFB   $3A
FF1E:F8                625           DFB   $F8
FF1F:FA                626           DFB   $FA
FF20:3B                627           DFB   $3B
FF21:FA                628           DFB   $FA
FF22:F9                629           DFB   $F9
FF23:22                630           DFB   $22
FF24:21                631           DFB   $21
FF25:3C                632           DFB   $3C
FF26:FA                633           DFB   $FA
FF27:FA                634           DFB   $FA
FF28:3D                635           DFB   $3D
FF29:3E                636           DFB   $3E
FF2A:3F                637           DFB   $3F
FF2B:FC                638           DFB   $FC               ;???
FF2C:00                639           BRK
```

```
                        640 *
FF2D:                   641 PRERR     LDA   #$C5            ;PRINT 'ERR', THEN FALL INTO
FF2F:20 ED FD           642           JSR   COUT            ; FWEEPER.
FF32:A9 D2              643           LDA   #$D2
FF34:20 ED FD           644           JSR   COUT
FF37:20 ED FD           645           JSR   COUT
FF3A:                   646 *
FF3A:A9 87              647 BELL      LDA   #$87            ;MAKE A JOYFUL NOISE, THEN RETURN.
FF3C:4C ED FD           648           JMP   COUT
FF3F:                   649 *
FF3F:A5 48              650 RESTORE   LDA   STATUS          ;RESTORE 6502 REGISTER CONTENTS
FF41:48                 651           PHA                   ; USED BY DEBUG SOFTWARE
FF42:A5 45              652           LDA   A5H
FF44:A6 46              653 RESTR1    LDX   XREG
FF46:A4 47              654           LDY   YREG
FF48:28                 655           PLP
FF49:60                 656           RTS
FF4A:                   657 *
FF4A:85 45              658 SAVE      STA   A5H             ;SAVE 6502 REGISTER CONTENTS
FF4C:86 46              659 SAV1      STX   XREG            ; FOR DEBUG SOFTWARE
FF4E:84 47              660           STY   YREG
FF50:08                 661           PHP
FF51:68                 662           PLA
FF52:85 48              663           STA   STATUS
FF54:BA                 664           TSX
FF55:86 49              665           STX   SPNT
FF57:D8                 666           CLD
FF58:60                 667           RTS
FF59:                   668 *
FF59:20 84 FE           669 OLDRST    JSR   SETNORM         ;SET SCREEN MODE
FF5C:20 2F FB           670           JSR   INIT            ; AND INIT KBD/SCREEN
FF5F:20 93 FE           671           JSR   SETVID          ; AS I/O DEVS.
FF62:20 89 FE           672           JSR   SETKBD
FF65:                   673 *
FF65:D8                 674 MON       CLD                 ;MUST SET HEX MODE!
FF66:20 3A FF           675           JSR   BELL            ;FWEEPER.
FF69:A9 AA              676 MONZ      LDA   #$AA            ;'*' PROMPT FOR MONITOR
FF6B:85 33              677           STA   PROMPT
FF6D:20 67 FD           678           JSR   GETLNZ          ;READ A LINE OF INPUT
FF70:20 C7 FF           679           JSR   ZMODE           ;CLEAR MONITOR MODE, SCAN IDX
FF73:20 A7 FF           680 NXTITM    JSR   GETNUM          ;GET ITEM, NON-HEX
FF76:84 34              681           STY   YSAV            ; CHAR IN A-REG.
FF78:A0 17              682           LDY   #SUBTBL-CHRTBL  ; X-REG=0 IF NO HEX INPUT
FF7A:88                 683 CHRSRCH   DEY
FF7B:30 E8   FF65       684           BMI   MON             ;COMMAND NOT FOUND, BEEP & TRY AGAIN.
FF7D:D9 CC FF           685           CMP   CHRTBL,Y        ;FIND COMMAND CHAR IN TABLE
FF80:D0 F8   FF7A       686           BNE   CHRSRCH         ;NOT THIS TIME
FF82:20 BE FF           687           JSR   TOSUB           ;GOT IT! CALL CORRESPONDING SUBROUTINE
FF85:A4 34              688           LDY   YSAV            ;PROCESS NEXT ENTRY ON HIS LINE
FF87:4C 73 FF           689           JMP   NXTITM
FF8A:                   690 *
FF8A:A2 03              691 DIG       LDX   #$03
FF8C:0A                 692           ASL   A
FF8D:0A                 693           ASL   A               ;GOT HEX DIGIT,
FF8E:0A                 694           ASL   A               ; SHIFT INTO A2
FF8F:0A                 695           ASL   A
FF90:0A                 696 NXTBIT    ASL   A
FF91:26 3E              697           ROL   A2L
```

```
FF93:26 3F          698            ROL   A2H
FF95:CA             699            DEX                    ;LEAVE X=$FF IF DIG
FF96:10 F8   FF90   700            BPL   NXTBIT
FF98:A5 31          701 NXTBAS     LDA   MODE
FF9A:D0 06   FFA2   702            BNE   NXTBS2           ;IF MODE IS ZERO,
FF9C:B5 3F          703            LDA   A2H,X            ; THEN COPY A2 TO A1 AND A3
FF9E:95 3D          704            STA   A1H,X
FFA0:95 41          705            STA   A3H,X
FFA2:E8             706 NXTBS2     INX
FFA3:F0 F3   FF98   707            BEQ   NXTBAS
FFA5:D0 06   FFAD   708            BNE   NXTCHR
FFA7:A2 00          709 GETNUM     LDX   #$00             ;CLEAR A2
FFA9:86 3E          710            STX   A2L
FFAB:86 3F          711            STX   A2H
FFAD:20 B4 C5       712 NXTCHR     JSR   GETUP            ;Get char, iny, upshift
FFB0:49 B0          713            EOR   #$B0
FFB2:C9 0A          714            CMP   #$0A
FFB4:90 D4   FF8A   715            BCC   DIG              ;it's a digit
FFB6:69 88          716            ADC   #$88
FFB8:C9 FA          717            CMP   #$FA
FFBA:4C C5 CF       718            JMP   LOOKASC          ;+ Check for quote
FFBD:00             719            BRK
FFBE:               720 *
FFBE:A9 FE          721 TOSUB      LDA   #<GO-1           ;DISPATCH TO SUBROUTINE, BY
FFC0:48             722            PHA                    ; PUSHING THE HI-ORDER SUBR ADDR,
FFC1:B9 E3 FF       723            LDA   SUBTBL,Y         ; THEN THE LO-ORDER SUBR ADDR
FFC4:48             724            PHA                    ; ONTO THE STACK,
FFC5:A5 31          725            LDA   MODE             ; (CLEARING THE MODE, SAVE THE OLD
FFC7:A0 00          726 ZMODE      LDY   #$00             ; MODE IN A-REG),
FFC9:84 31          727            STY   MODE
FFCB:60             728            RTS                    ; AND 'RTS' TO THE SUBROUTINE!
FFCC:               729 *
FFCC:BC             730 CHRTBL     DFB   $BC              ;^C  (BASIC WARM START)
FFCD:B2             731            DFB   $B2              ;^Y  (USER VECTOR)
FFCE:BE             732            DFB   $BE              ;^E  (OPEN AND DISPLAY REGISTERS)
FFCF:9A             733            DFB   $9A              ;+!  (Mini assembler)
FFD0:EF             734            DFB   $EF              ;V   (MEMORY VERIFY)
FFD1:C4             735            DFB   $C4              ;^K  (IN#SLOT)
FFD2:A9             736            DFB   $A9              ;^P  (PR#SLOT)
FFD3:BB             737            DFB   $BB              ;^B  (BASIC COLD START)
FFD4:A6             738            DFB   $A6              ;'-' (SUBTRACTION)
FFD5:A4             739            DFB   $A4              ;'+' (ADDITION)
FFD6:06             740            DFB   $06              ;M   (MEMORY MOVE)
FFD7:95             741            DFB   $95              ;'<' (DELIMITER FOR MOVE, VFY)
FFD8:07             742            DFB   $07              ;N   (SET NORMAL VIDEO)
FFD9:02             743            DFB   $02              ;I   (SET INVERSE VIDEO)
FFDA:05             744            DFB   $05              ;L   (DISASSEMBLE 20 INSTRS)
FFDB:00             745            DFB   $00              ;G   (EXECUTE PROGRAM)
FFDC:93             746            DFB   $93              ;':' (MEMORY FILL)
FFDD:A7             747            DFB   $A7              ;'.' (ADDRESS DELIMITER)
FFDE:C6             748            DFB   $C6              ;'CR' (END OF INPUT)
FFDF:99             749            DFB   $99              ;BLANK
FFE0:EC             750            DFB   $EC              ;+S  (Step)
FFE1:ED             751            DFB   $ED              ;+T  (Trace)
FFE2:EA             752            NOP                    ;+
FFE3:               753 *
FFE3:               754 * Table of low order monitor routine
FFE3:               755 * dispatch addresses.
```

```
FFE3:                 756 *
FFE3:B2               757 SUBTBL    DFB   >BASCONT-1
FFE4:C9               758           DFB   >USR-1
FFE5:BE               759           DFB   >REGZ-1
FFE6:6B               760           DFB   >MINI-1          ;+
FFE7:35               761           DFB   >VERIFY-1
FFE8:8C               762           DFB   >INPRT-1
FFE9:96               763           DFB   >OUTPRT-1
FFEA:AF               764           DFB   >XBASIC-1
FFEB:17               765           DFB   >SETMODE-1
FFEC:17               766           DFB   >SETMODE-1
FFED:2B               767           DFB   >MOVE-1
FFEE:1F               768           DFB   >LT-1
FFEF:83               769           DFB   >SETNORM-1
FFF0:7F               770           DFB   >SETINV-1
FFF1:5D               771           DFB   >LIST-1
FFF2:B5               772           DFB   >GO-1
FFF3:17               773           DFB   >SETMODE-1
FFF4:17               774           DFB   >SETMODE-1
FFF5:F5               775           DFB   >CRMON-1
FFF6:03               776           DFB   >BLANK-1
FFF7:70               777           DFB   >STEPZ-1          ;+
FFF8:6E               778           DFB   >TRACE-1          ;+
FFF9:                 779 *
FFF9:       0001      780           ds    $FFFA-*,0
FFFA:                 781 *
FFFA:FB 03            782           DW    NMI               ;NON-MASKABLE INTERRUPT VECTOR
FFFC:62 FA            783           DW    RESET             ;RESET VECTOR
FFFE:03 C8            784 IRQVECT   DW    NEWIRQ            ;INTERRUPT REQUEST VECTOR
0000:                  50           include bank2
```

```
0000:                  2 ****************************************
0000:                  3 *
0000:                  4 * Bank 2 of the roms
0000:                  5 *
0000:                  6 ****************************************
---- NEXT OBJECT FILE NAME IS /BUILD/FIRM.2
C000:          C000    7            org   $C000
C000:                 51            include mint          ;Mouse & acia interrupt handler
C000:          0100    1            ds    $C100-*,0
```

```
C100:                  4 ****************************************
C100:                  5 *
C100:                  6 * Mouse interrupt handler
C100:                  7 *
C100:                  8 * MOUSEINT - Monitor's interrupt handler
C100:                  9 *
C100:                 10 * Returns C = 0 if interrupt handled
C100:                 11 * If not mouse interrupt, Goes to aciaint
C100:                 12 * New in this rom:
C100:                 13 * If D7 of moumode = 1, mouse X and Y interrupts are not processed
C100:                 14 * and are passed on to the user.
C100:                 15 *
C100:                 16 ****************************************
C100:          C100   17 mouseint  equ   *              ;Entry point if X & Y set up
C100:A9 0E            18          lda   #$0E           ;Clear status bits
C102:1C 7C 07         19          trb   moustat


C105:38               21          sec                   ;Assume interrupt not handled
C106:                 22 * Check for vertical blanking interrupt
C106:AD 19 C0         23          lda   vblint         ;VBL interrupt?
C109:10 2B    C136    24          bpl   chkmou
C10B:8D 79 C0         25          sta   iouenbl        ;Enable iou access & clear VBL interrupt
C10E:A9 0C            26          lda   #vblmode       ;Should we leave vbl active?
C110:2C FC 07         27          bit   moumode
C113:D0 03    C118    28          bne   cvnovbl
C115:8D 5A C0         29          sta   iou+2          ;Disable VBL
C118:09 02            30 cvnovbl  ora   #movmode
C11A:8D 78 C0         31          sta   ioudsbl
C11D:2C 7C 06         32          bit   mouarm         ;VBL bit in arm isn't used
C120:D0 02    C124    33          bne   cvmoved
C122:A9 0C            34          lda   #vblmode       ;Didn't move
C124:2C 63 C0         35 cvmoved  bit   moubut         ;Button pressed?
C127:10 02    C12B    36          bpl   cvbut
C129:49 04            37          eor   #butmode       ;Clear the button bit
C12B:2D FC 07         38 cvbut    and   moumode        ;Which bits were set in the mode
C12E:0C 7C 07         39          tsb   moustat
C131:1C 7C 06         40          trb   mouarm
C134:69 FE            41          adc   #$FE           ;C=1 if int passes to user
C136:                 42 * Check & update mouse movement
C136:          C136   43 chkmou    equ   *
C136:AD FC 07         44          lda   moumode        ;If D7 = 1, user better handle it
C139:30 72    C1AD    45          bmi   xmdone
C13B:AD 15 C0         46          lda   mouxint        ;Mouse interrupt?
C13E:0D 17 C0         47          ora   mouyint
C141:10 6A    C1AD    48          bpl   xmdone         ;If not return with C from vbl
C143:8A               49          txa                   ;Get X1 in A
C144:A2 00            50          ldx   #0
C146:2C 15 C0         51          bit   mouxint        ;X movement?
C149:30 0A    C155    52          bmi   cmxmov
C14B:98               53 cmloop   tya                   ;Get Y1 into A
C14C:49 80            54          eor   #$80           ;Complement direction
C14E:A2 80            55          ldx   #$80
C150:2C 17 C0         56          bit   mouyint
C153:10 39    C18E    57          bpl   cmnoy
C155:0A               58 cmxmov   asl   A
C156:BD 7C 04         59          lda   mouxl,x        ;A = current low byte
```

```
C159:B0 1A   C175   60            bcs   cmrght        ;Which way?
C15B:DD 7D 04       61            cmp   minxl,x       ;Move left
C15E:D0 08   C168   62            bne   cmlok
C160:BD 7C 05       63            lda   mouxh,x
C163:DD 7D 05       64            cmp   minxh,x
C166:F0 22   C18A   65            beq   cmnoint
C168:BD 7C 04       66 cmlok      lda   mouxl,x
C16B:D0 03   C170   67            bne   cmnt0         ;Borrow from high byte?
C16D:DE 7C 05       68            dec   mouxh,x
C170:DE 7C 04       69 cmnt0      dec   mouxl,x
C173:80 15   C18A   70            bra   cmnoint
C175:DD 7D 06       71 cmrght     cmp   maxxl,x       ;At high bound?
C178:D0 08   C182   72            bne   cmrok
C17A:BD 7C 05       73            lda   mouxh,x
C17D:DD 7D 07       74            cmp   maxxh,x
C180:F0 08   C18A   75            beq   cmnoint
C182:FE 7C 04       76 cmrok      inc   mouxl,x       ;Move right
C185:D0 03   C18A   77            bne   cmnoint
C187:FE 7C 05       78            inc   mouxh,x
C18A:E0 00          79 cmnoint    cpx   #0
C18C:F0 BD   C14B   80            beq   cmloop
C18E:8D 48 C0       81 cmnoy      sta   mouclr
C191:A9 02          82            lda   #movmode      ;Should we enable VBL?
C193:2D FC 07       83            and   moumode
C196:F0 09   C1A1   84            beq   cmnovbl       ;Branch if not
C198:8D 79 C0       85            sta   iouenbl
C19B:8D 5B C0       86            sta   iou+3         ;Enable VBL int
C19E:8D 78 C0       87            sta   ioudsbl
C1A1:09 20          88 cmnovbl    ora   #movarm       ;Mark that we moved
C1A3:0C 7C 06       89            tsb   mouarm
C1A6:A9 0E          90            lda   #$0E
C1A8:2D 7C 07       91            and   moustat
C1AB:69 FE          92            adc   #$FE          ;C=1 iff any bits were 1
C1AD:B0 05   C1B4   93 xmdone     bcs   aciaint       ;If not handled, try acia
C1AF:4C 84 C7       94            jmp   swrts2        ;Back we go
```

```
C1B2:                 96 *   This routine will determine if the source of
C1B2:                 97 * is either of the built in ACIAs.  If neither port
C1B2:                 98 * generated the interrupt, or the interrupt was due
C1B2:                 99 * to a transmit buffer empty, protocol converter, or
C1B2:                100 * 'unbuffered' receiver full, the carry is set indi-
C1B2:                101 * cating an externally serviced interrupt.
C1B2:                102 *   If the interrupt source was keyboard, 'buffered'
C1B2:                103 * serial input, or the DCD, the interrupt is serviced
C1B2:                104 * and the carry is cleared indicating interrupt was
C1B2:                105 * serviced. (DCD handshake replaces CTS.)
C1B2:                106 *   Location "ACIABUF" specifies which (if either) re-
C1B2:                107 * ceiver data is buffered.  For port 1 it must contain
C1B2:                108 * $C1, for port 2 a $C2.  Any other values are cause
C1B2:                109 * interrupts to pass to external (RAM based) routines.
C1B2:                110 *   Location "TYPHED" specifies whether Keyboard in-
C1B2:                111 * put should be buffered, ignored, or processed by
C1B2:                112 * RAM based routines.  If bit 7=1 and bit 6=0, key-
C1B2:                113 * board data is placed in the type-ahead buffer. If
C1B2:                114 * bit 6 is set the interrupt is cleared, but must
C1B2:                115 * be recognized and serviced by a RAM routine.  If
C1B2:                116 * both bits = 0, the interrupt is serviced, but the
C1B2:                117 * keyboard data is ignored.
C1B2:                118 *   While using type-ahead, Open-Apple CTRL-X will
C1B2:                119 * flush the buffer.  No other code is recognized.
C1B2:                120 *   If the source was an ACIA that has the transmit
C1B2:                121 * interrupt enabled, the original value of the ACIAs
C1B2:                122 * status registers is preserved. Automatic serial input
C1B2:                123 * buffering is not serviced from a port so configured.
C1B2:                124 * Interrupts originating from the protocol converter or
C1B2:                125 * keyboard (RAM serviced) do not inhibit serial buffering
C1B2:                126 * and are passed thru.  The RAM service routine can rec-
C1B2:                127 * ognize the interrupt source by a 1 state in bit 6 of
C1B2:                128 * the ACIAs status register.  The RAM service routine must
C1B2:                129 * cause the clearing of DSR (bit 6) AND make a second ac-
C1B2:                130 * cess to the status register before returning.
C1B2:                131 *
C1B2:                132 *
C1B2:38              133 notacia   sec                   ;Not acia int
C1B3:60              134 acdone    rts
C1B4:          C1B4  135 aciaint   equ   *
C1B4:20 BA C1        136           jsr   aciaint2        ;Extra jsr since rest needs RTS
C1B7:4C 84 C7        137           jmp   swrts2
C1BA:          C1BA  138 aciaint2  equ   *
C1BA:A2 C2           139           ldx   #<comslot       ;Test port 2 first
C1BC:20 C2 C1        140           jsr   aciatst         ;Check for interrupt
C1BF:90 F2     C1B3  141           bcc   acdone          ;Return if interrupt done
C1C1:CA              142           dex                    ;Try port 1
C1C2:BC 42 C1        143 aciatst   ldy   devno2,x        ;Get index for acia
C1C5:A9 04           144           lda   #$4             ;If xmit ints enabled pass to user
C1C7:59 FA BF        145           eor   scomd,y         ;Check if D<3>, D<2> = 01
C1CA:29 0C           146           and   #$0C            ;
C1CC:F0 E4     C1B2  147           beq   notacia         ;User better take it!
C1CE:B9 F9 BF        148           lda   sstat,y         ;Get status
C1D1:9D 38 04        149           sta   astat,x         ;Save it away
C1D4:10 DC     C1B2  150           bpl   notacia         ;No interrupt
C1D6:E0 C2           151 aitst2    cpx   #<comslot       ;C=1 if com port. Called from serout3
C1D8:B0 02     C1DC  152           bcs   aiport2         ;Invert DSR if port1
C1DA:49 40           153           eor   #$40
```

```
C1DC:3C 38 05      154 aiport2   bit   extint,x      ;Is DSR enabled?
C1DF:70 29   C20A  155           bvs   aipass        ;Yes, user wants it
C1E1:10 25   C208  156           bpl   aieatit       ;No, eat it
C1E3:90 23   C208  157           bcc   aieatit       ;Yes but I don't want it for port 1
C1E5:89 40         158           bit   #$40          ;Is DSR 1?
C1E7:F0 21   C20A  159           beq   aipass        ;If not, skip it
C1E9:              160 * It's a keyboard interrupt
C1E9:AD 00 C0      161           lda   kbd           ;Get the key
C1EC:A0 80         162           ldy   #$80
C1EE:20 28 C2      163           jsr   putbuf        ;Put it in the buffer
C1F1:C9 98         164           cmp   #$98          ;Is it a ^x?
C1F3:D0 0B   C200  165           bne   ainoflsh
C1F5:AD 62 C0      166           lda   butn1         ;And the closed apple?
C1F8:10 06   C200  167           bpl   ainoflsh
C1FA:8E FF 05      168           stx   twkey         ;Flush the type ahead buffer
C1FD:8E FF 06      169           stx   trkey
C200:AD 10 C0      170 ainoflsh  lda   kbdstrb       ;Clear the keyboard
C203:              171 * $A0 $B0 table needed by serial firmware
C203:        C142  172 devno2    equ   *-sltdmy
C203:A0 B0         173           ldy   #$B0          ;Restore y
C205:B9 F9 BF      174           lda   sstat,y       ;Read status to clear int
C208:29 BF         175 aieatit   and   #$BF          ;Clear the DSR bit
C20A:0A            176 aipass    asl   A             ;Shift DSR into C
C20B:0A            177           asl   A
C20C:29 20         178           and   #$20          ;Is the receiver full?
C20E:F0 3E   C24E  179           beq   aciadone      ;If not, we're done
C210:B9 FA BF      180           lda   scomd,y       ;Are receive interrupts enabled?
C213:49 01         181           eor   #1            ;Check for D<1>,D<0> = 01
C215:29 03         182           and   #3
C217:D0 35   C24E  183           bne   aciadone      ;If not, were done
C219:8A            184           txa                  ;Is this acia buffered?
C21A:4D FF 04      185           eor   aciabuf
C21D:D0 93   C1B2  186           bne   notacia       ;The user better handle it!
C21F:08            187           php                  ;Save DSR status
C220:20 22 C3      188           jsr   getdata       ;Get char & check xon, etc
C223:90 28   C24D  189           bcc   aieat         ;Don't put in buffer if eaten
C225:A0 00         190           ldy   #0
C227:D0            191           dfb   $D0           ;BNE opcode to skip PHP
C228:        C228  192 putbuf    equ   *
C228:08            193           php
C229:DA            194           phx
C22A:48            195           pha
C22B:B9 7F 05      196           lda   twser,y       ;Get buffer pointer
C22E:AA            197           tax                  ;Save it for later
C22F:1A            198           inc   A             ;Bump it to next free byte
C230:89 7F         199           bit   #$7F          ;Overflow?
C232:D0 01   C235  200           bne   pbok
C234:98            201           tya                  ;Wrap pointer
C235:D9 7F 06      202 pbok      cmp   trser,y       ;Buffer full?
C238:F0 03   C23D  203           beq   pbfull
C23A:99 7F 05      204           sta   twser,y       ;Save the new pointer
C23D:68            205 pbfull    pla                  ;Get the data
C23E:2C 14 C0      206           bit   rdramwrt
C241:8D 05 C0      207           sta   wrcardram     ;It goes to aux ram
C244:9D 00 08      208           sta   thbuf,x
C247:30 03   C24C  209           bmi   aiaux         ;Branch if we want aux
C249:8D 04 C0      210           sta   wrmainram
C24C:FA            211 aiaux     plx
```

```
C24D:28          212 aieat      plp                ;Get DSR status back
C24E:60          213 aciadone   rts
```

```
C24F:                 215 ****************************************
C24F:                 216 *
C24F:                 217 * SEROUT3 - Outputs a character to a acia
C24F:                 218 * Inputs: A = char, X = Cn
C24F:                 219 *
C24F:                 220 ****************************************
C24F:          C24F   221 serout3   equ   *
C24F:20 55 C2         222          jsr   serout4
C252:4C 84 C7         223          jmp   swrts2
C255:          C255   224 serout4   equ   *              ;Entry point with rts
C255:48               225          pha                   ;Save the char
C256:2C AB C2         226          bit   sorts           ;Control char?
C259:F0 03    C25E    227          beq   sordy           ;Don't inc column if so
C25B:FE 38 07         228          inc   col,x
C25E:20 B2 C2         229 sordy    jsr   getstat2        ;Get acia status
C261:29 30            230          and   #$30            ;Y set by getstat
C263:C9 10            231          cmp   #$10
C265:D0 F7    C25E    232          bne   sordy
C267:BD B8 06         233          lda   flags,x         ;Is XON/XOFF enabled?
C26A:89 20            234          bit   #$20
C26C:F0 1F    C28D    235          beq   sook            ;Branch if not
C26E:EC FF 04         236          cpx   aciabuf         ;Is port interrupt driven?
C271:F0 13    C286    237          beq   sotst
C273:20 E9 C2         238          jsr   xrdnobuf        ;Get a char from the acia
C276:90 0E    C286    239          bcc   sotst           ;Branch if no char
C278:BC 34 C2         240          ldy   charptr,x       ;Get pointer to charbuf
C27B:99 FE 05         241          sta   charbuf,y       ;Save the character
C27E:BD B8 06         242          lda   flags,x         ;Set bit for char in buffer
C281:09 04            243          ora   #$04
C283:9D B8 06         244          sta   flags,x
C286:BD B8 06         245 sotst    lda   flags,x         ;Check if in xoff
C289:29 02            246          and   #$02
C28B:D0 D1    C25E    247          bne   sordy           ;Loop if not ready
C28D:BC 42 C1         248 sook     ldy   devno2,x
C290:68               249          pla
C291:48               250          pha                   ;Get char to XMIT
C292:99 F8 BF         251          sta   sdata,y         ;Out it goes
C295:3C B8 06         252          bit   flags,x         ;V=1 if LF after CR
C298:49 0D            253          eor   #$0D            ;check for CR.
C29A:0A               254          asl   A               ;preserve bit 7
C29B:D0 0D    C2AA    255          bne   sodone          ;branch if not CR.
C29D:50 06    C2A5    256          bvc   clrcol          ;branch if no LF after CR
C29F:A9 14            257          lda   #$14            ;Get LF*2
C2A1:6A               258          ror   A               ;no shift in high bit
C2A2:20 55 C2         259          jsr   serout4         ;Output the LF but don't echo it
C2A5:64 24            260 clrcol   stz   ch              ;0 position & column
C2A7:9E 38 07         261          stz   col,x
C2AA:68               262 sodone   pla                   ;Get the char back
C2AB:60               263 sorts    rts
```

```
C2AC:                  265 ****************************************
C2AC:                  266 *
C2AC:                  267 * GETSTAT - Gets the status from a acia
C2AC:                  268 * GETSTAT2 - Call from this side
C2AC:                  269 * If interrupt, aciatst is called
C2AC:                  270 * note: external interrupts are lost
C2AC:                  271 * inputs: X = Cn
C2AC:                  272 * outputs: A = status, X = Cn, Y = devno
C2AC:                  273 *
C2AC:                  274 ****************************************
C2AC:          C2AC    275 getstat   equ   *
C2AC:20 B2 C2          276           jsr   getstat2
C2AF:4C 84 C7          277           jmp   swrts2        ;Return to other side
C2B2:          C2B2    278 getstat2  equ   *
C2B2:08                279           php                  ;Save interrupt status
C2B3:78                280           sei
C2B4:BC 42 C1          281 gsttst    ldy   devno2,x       ;Get index into hardware
C2B7:B9 F9 BF          282           lda   sstat,y        ;Get the status
C2BA:10 05    C2C1     283           bpl   gstnoint       ;D7 = 1 if interrupt
C2BC:20 D6 C1          284           jsr   aitst2         ;Go service the interrupt
C2BF:80 F3    C2B4     285           bra   gsttst         ;Interrupt may have changed status
C2C1:28                286 gstnoint  plp                  ;Restore interrupt status
C2C2:60                287           rts
```

```
C2C3:                 289 ****************************************
C2C3:                 290 *   This is the serial input routine.  Carry
C2C3:                 291 * flag set indicates that returned data is
C2C3:                 292 * valid.
C2C3:                 293 *
C2C3:                 294 ****************************************
C2C3:         C2C3    295 xrdser     equ   *
C2C3:20 C9 C2         296            jsr   xrdser2
C2C6:4C 84 C7         297            jmp   swrts
C2C9:         C2C9    298 xrdser2    equ   *
C2C9:EC FF 04         299            cpx   aciabuf        ;is serial input buffered?
C2CC:D0 07    C2D5    300            bne   xnosbuf        ;(in english "NO SERIAL BUFFER")
C2CE:A0 00            301            ldy   #0             ;Y=0 for serial buffer
C2D0:20 FD C2         302            jsr   getbuf2        ;Any data in buffer?
C2D3:B0 1F    C2F4    303            bcs   xrddone
C2D5:                 304 *
C2D5:BD B8 06         305 xnosbuf    lda   flags,x        ;Is there a char in the onr byte buffer?
C2D8:89 04            306            bit   #$04
C2DA:F0 0D    C2E9    307            beq   xrdnobuf       ;Branch if not
C2DC:29 FB            308            and   #$FB           ;Clear the bit
C2DE:9D B8 06         309            sta   flags,x
C2E1:BC 34 C2         310            ldy   charptr,x
C2E4:B9 FE 05         311            lda   charbuf,y
C2E7:38               312            sec
C2E8:60               313            rts
C2E9:                 314 *
C2E9:20 B2 C2         315 xrdnobuf   jsr   getstat2       ;Get ACIA status
C2EC:29 08            316            and   #$8
C2EE:18               317            clc                  ;indicate no data
C2EF:F0 03    C2F4    318            beq   xrddone        ;Branch if no data!
C2F1:20 22 C3         319            jsr   getdata        ;Get data and check xon, etc
C2F4:60               320 xrddone    rts

C2F5:         C234    322 charptr    equ   *-$C1          ;Pointer to character buffers
C2F5:00 80            323            dfb   $0,$80


C2F7:                 325 ****************************************
C2F7:                 326 *
C2F7:                 327 * GETBUF - Gets a byte from the input buffer
C2F7:                 328 * Inputs: Y=0 for Serial buffer 80 for Keyboard buffer
C2F7:                 329 * C = 0 if no data C = 1 if data valid A = Data
C2F7:                 330 *
C2F7:                 331 ****************************************
C2F7:         C2F7    332 getbuf     equ   *
C2F7:20 FD C2         333            jsr   getbuf2
C2FA:4C 84 C7         334            jmp   swrts
C2FD:         C2FD    335 getbuf2    equ   *
C2FD:B9 7F 06         336            lda   trser,Y        ;Test for data in buffer
C300:D9 7F 05         337            cmp   twser,Y        ;If = then no data
C303:18               338            clc
C304:F0 1B    C321    339            beq   gbdone         ;Branch if empty
C306:48               340            pha                  ;Save current value
C307:1A               341            inc   A              ;Update the pointer
C308:89 7F            342            bit   #$7F           ;Overflow
C30A:D0 01    C30D    343            bne   gbnoovr
C30C:98               344            tya
```

```
C30D:99 7F 06      345 gbnoovr   sta   trser,y       ;Store the updated pointer
C310:7A            346           ply                 ;Get the old value of the pointer
C311:AD 13 C0      347           lda   rdramrd       ;Are we in main ram
C314:0A            348           asl   A             ;C=1 for Aux ram
C315:8D 03 C0      349           sta   rdcardram     ;Force Aux ram
C318:B9 00 08      350           lda   thbuf,Y       ;Get byte from buffer
C31B:B0 04   C321  351           bcs   gbdone        ;Branch if we were in aux bank
C31D:8D 02 C0      352           sta   rdmainram     ;Set back to main
C320:38            353           sec                 ;Mark data there
C321:60            354 gbdone    rts


C322:              356 ****************************************
C322:              357 *
C322:              358 * GETDATA - Gets data from serial port
C322:              359 * and checks for LF, XON, XOFF
C322:              360 * inputs: Y = index to acia
C322:              361 * outputs: A = data, Y dest, C = 1 if data ok = 0 if eaten
C322:              362 *
C322:              363 ****************************************
C322:       C322   364 getdata   equ   *
C322:B9 F8 BF      365           lda   sdata,y
C325:48            366           pha                 ;Save the data
C326:09 80         367           ora   #$80          ;Set D7 for compares
C328:A8            368           tay
C329:BD B8 06      369           lda   flags,x       ;Get options byte
C32C:89 08         370           bit   #$08          ;Eat linefeeds?
C32E:D0 04   C334  371           bne   gdnolf
C330:C0 8A         372           cpy   #lfeed        ;Is it a LF?
C332:F0 12   C346  373           beq   gdeat         ;Eat it if it is
C334:89 20         374 gdnolf    bit   #$20          ;Xon/XOFF enabled?
C336:F0 10   C348  375           beq   gdok
C338:C0 91         376           cpy   #xon          ;Is it an XON?
C33A:D0 04   C340  377           bne   gdnxon
C33C:29 FD         378           and   #$FD          ;Clear xoff bit
C33E:80 06   C346  379           bra   gdeat         ;And eat it
C340:C0 93         380 gdnxon    cpy   #xoff
C342:D0 04   C348  381           bne   gdok
C344:09 02         382           ora   #$02          ;Set xoff bit
C346:18            383 gdeat     clc
C347:B0            384           dfb   $B0           ;BCS opcode
C348:38            385 gdok      sec
C349:9D B8 06      386           sta   flags,x
C34C:68            387           pla
C34D:60            388           rts
C34E:               52           include auxstuff    ;Auxillary move stuff
```

```
C34E:                  4 ****************************************
C34E:                  5 * NAME    : MOVEAUX
C34E:                  6 * FUNCTION: PERFORM CROSSBANK MEMORY MOVE
C34E:                  7 * INPUT   : A1=SOURCE ADDRESS
C34E:                  8 *         : A2=SOURCE END
C34E:                  9 *         : A4=DESTINATION START
C34E:                 10 *         : CARRY SET=MAIN->CARD
C34E:                 11 *                CLR=CARD->MAIN
C34E:                 12 * OUTPUT  : NONE
C34E:                 13 * VOLATILE: NOTHING
C34E:                 14 * CALLS   : NOTHING
C34E:                 15 ****************************************
C34E:          C34E   16 MOVEAUX   EQU   *
C34E:48               17           PHA                 ;SAVE AC
C34F:AD 13 C0         18           LDA   RDRAMRD       ;SAVE STATE OF
C352:48               19           PHA                 ; MEMORY FLAGS
C353:AD 14 C0         20           LDA   RDRAMWRT
C356:48               21           PHA
C357:                 22 *
C357:                 23 * SET FLAGS FOR CROSSBANK MOVE:
C357:                 24 *
C357:90 08    C361    25           BCC   MOVEC2M       ;=>CARD->MAIN
C359:8D 02 C0         26           STA   RDMAINRAM     ;SET FOR MAIN
C35C:8D 05 C0         27           STA   WRCARDRAM     ; TO CARD
C35F:B0 06    C367    28           BCS   MOVESTRT      ;=>(ALWAYS TAKEN)
C361:                 29 *
C361:          C361   30 MOVEC2M   EQU   *
C361:8D 04 C0         31           STA   WRMAINRAM     ;SET FOR CARD
C364:8D 03 C0         32           STA   RDCARDRAM     ; TO MAIN
C367:                 33 *
C367:          C367   34 MOVESTRT  EQU   *
C367:B2 3C            35 MOVELOOP  LDA   (A1L)         ;get a byte
C369:92 42            36           STA   (A4L)         ;move it
C36B:E6 42            37           INC   A4L
C36D:D0 02    C371    38           BNE   NEXTA1
C36F:E6 43            39           INC   A4H
C371:A5 3C            40 NEXTA1    LDA   A1L
C373:C5 3E            41           CMP   A2L
C375:A5 3D            42           LDA   A1H
C377:E5 3F            43           SBC   A2H
C379:E6 3C            44           INC   A1L
C37B:D0 02    C37F    45           BNE   C01
C37D:E6 3D            46           INC   A1H
C37F:90 E6    C367    47 C01       BCC   MOVELOOP      ;=>more to move
C381:                 48 *
C381:8D 04 C0         49           STA   WRMAINRAM     ;CLEAR FLAG2
C384:68               50           PLA                 ;GET ORIGINAL STATE
C385:10 03    C38A    51           BPL   C03           ;=>IT WAS OFF
C387:8D 05 C0         52           STA   WRCARDRAM
C38A:          C38A   53 C03       EQU   *
C38A:8D 02 C0         54           STA   RDMAINRAM     ;CLEAR FLAG1
C38D:68               55           PLA                 ;GET ORIGINAL STATE
C38E:10 03    C393    56           BPL   MOVERET       ;=>IT WAS OFF
C390:8D 03 C0         57           STA   RDCARDRAM
C393:          C393   58 MOVERET   EQU   *
C393:68               59           PLA                 ;Restore AC
C394:4C 84 C7         60           JMP   SWRTS2
```

```
C397:                  62 ****************************************
C397:                  63 * NAME     : XFER
C397:                  64 * FUNCTION: TRANSFER CONTROL CROSSBANK
C397:                  65 * INPUT    : $03ED=TRANSFER ADDR
C397:                  66 *           : CARRY SET=XFER TO CARD
C397:                  67 *                 CLR=XFER TO MAIN
C397:                  68 *           : VFLAG CLR=USE STD ZP/STK
C397:                  69 *           :       SET=USE ALT ZP/STK
C397:                  70 * OUTPUT   : NONE
C397:                  71 * VOLATILE: $03ED/03EE IN DEST BANK
C397:                  72 * CALLS    : NOTHING
C397:                  73 * NOTE     : ENTERED VIA JMP, NOT JSR
C397:                  74 ****************************************
C397:                  75 *
C397:          C397    76 XFER       EQU   *
C397:48                77            PHA                    ;SAVE AC ON CURRENT STACK
C398:                  78 *
C398:                  79 * COPY DESTINATION ADDRESS TO THE
C398:                  80 *  OTHER BANK SO THAT WE HAVE IT
C398:                  81 *  IN CASE WE DO A SWAP:
C398:                  82 *
C398:AD ED 03          83            LDA   $03ED            ;GET XFERADDR LO
C39B:48                84            PHA                    ;SAVE ON CURRENT STACK
C39C:AD EE 03          85            LDA   $03EE            ;GET XFERADDR HI
C39F:48                86            PHA                    ;SAVE IT TOO
C3A0:                  87 *
C3A0:                  88 * SWITCH TO APPROPRIATE BANK:
C3A0:                  89 *
C3A0:90 08    C3AA     90            BCC   XFERC2M          ;=>CARD->MAIN
C3A2:8D 03 C0          91            STA   RDCARDRAM        ;SET FOR RUNNING
C3A5:8D 05 C0          92            STA   WRCARDRAM        ; IN CARD RAM
C3A8:B0 06    C3B0     93            BCS   XFERZP           ;=> always taken
C3AA:         C3AA     94 XFERC2M    EQU   *
C3AA:8D 02 C0          95            STA   RDMAINRAM        ;SET FOR RUNNING
C3AD:8D 04 C0          96            STA   WRMAINRAM        ; IN MAIN RAM
C3B0:                  97 *
C3B0:         C3B0     98 XFERZP     EQU   *                ;SWITCH TO ALT ZP/STK
C3B0:68                99            PLA                    ;STUFF XFERADDR
C3B1:8D EE 03         100            STA   $03EE            ; HI AND
C3B4:68               101            PLA
C3B5:8D ED 03         102            STA   $03ED            ;  LO
C3B8:68               103            PLA                    ;RESTORE AC
C3B9:70 05    C3C0    104            BVS   XFERAZP          ;=>switch in alternate zp
C3BB:8D 08 C0         105            STA   SETSTDZP         ;else force standard zp
C3BE:50 03    C3C3    106            BVC   JMPDEST          ;=>always perform transfer
C3C0:8D 09 C0         107 XFERAZP   STA   SETALTZP         ;switch in alternate zp
C3C3:4C EB C7         108 JMPDEST   JMP   SWXFG02          ;Back we go
C3C6:                 109 ****************************************
C3C6:                  53            include banger2        ;Diagnostic routines
```

```
C3C6:                  3 ****************************************
C3C6:                  4 *
C3C6:                  5 * Here is the rest of the diagnostic stuff
C3C6:                  6 * the first part has been moved into the $D000 space
C3C6:                  7 * to make desperately needed room
C3C6:                  8 *
C3C6:                  9 ****************************************
C3C6:          C3C6   10 TSTMEM    equ   *
C3C6:86 01             11          stx   $01
C3C8:86 02             12          stx   $02
C3CA:86 03             13          stx   $03
C3CC:A2 04             14          ldx   #4            ;do RAM $100-$FFFF five times
C3CE:86 04             15          stx   $04
C3D0:85 05             16 MEM1     STA   $05           ;keep acc in a safe place
C3D2:A2 04             17          ldx   #4
C3D4:64 01             18          stz   $01
C3D6:E6 01             19          inc   1             ;point to page 1 first
C3D8:A8                20 mem2     tay                 ;save ACC in Y for now
C3D9:8D 83 C0          21          sta   lcbank2       ;anticipate not $C000 range...
C3DC:8D 83 C0          22          sta   lcbank2
C3DF:A5 01             23          lda   $01           ;get page address
C3E1:29 F0             24          and   #$F0          ;test for $C0-$CF range
C3E3:C9 C0             25          cmp   #$C0
C3E5:D0 0C   C3F3      26          bne   mem3          ;branch if not...
C3E7:AD 8B C0          27          lda   lcbank1
C3EA:AD 8B C0          28          lda   lcbank1       ;select primary $D000 space
C3ED:A5 01             29          lda   $01
C3EF:69 0F             30          adc   #$F           ;Plus carry =+$10
C3F1:D0 02   C3F5      31          bne   mem4          ;branch always taken
C3F3:A5 01             32 mem3     lda   $01
C3F5:85 03             33 mem4     sta   $03
C3F7:98                34          tya                 ;restore pattern to ACC
C3F8:A0 00             35          ldy   #$00          ;fill this page with the pattern
C3FA:18                36 mem5     clc
C3FB:7D 2A C8          37          adc   ntbl,x
C3FE:91 02             38          sta   ($02),y
C400:CA                39          dex                 ;keep x in the range 0-4
C401:10 02   C405      40          bpl   mem6
C403:A2 04             41          ldx   #4
C405:C8                42 mem6     iny                 ;all 256 filled yet?
C406:D0 F2   C3FA      43          bne   mem5          ;branch if not
C408:E6 01             44          inc   1             ;bump page #
C40A:D0 CC   C3D8      45          bne   mem2          ;loop through $0100 to $FF00

C40C:E6 01             47          inc   $01           ;point to page 1 again
C40E:A2 04             48          LDX   #4
C410:A5 05             49          LDA   $05
C412:A8                50 mem7     tay                 ;save ACC in Y for now
C413:AD 83 C0          51          lda   lcbank2       ;anticipate not $C000 range...
C416:AD 83 C0          52          lda   lcbank2
C419:A5 01             53          lda   $01           ;get page address
C41B:29 F0             54          and   #$F0          ;test for $C0-$CF range
C41D:C9 C0             55          cmp   #$C0
C41F:D0 09   C42A      56          bne   mem8          ;branch if not...
C421:AD 8B C0          57          lda   lcbank1       ;select primary $D000 space
C424:A5 01             58          lda   $01
C426:69 0F             59          adc   #$F           ;Plus carry =+$10
C428:D0 02   C42C      60          bne   mem9          ;branch always taken
```

```
C42A:A5 01          61 mem8     lda   $01
C42C:85 03          62 mem9     sta   $03
C42E:98             63          tya                 ;restore pattern to ACC
C42F:A0 00          64          ldy   #$00          ;fill this page with the pattern
C431:18             65 memA     clc
C432:7D 2A C8       66          adc   ntbl,x
C435:51 02          67          eor   ($02),y
C437:D0 39   C472   68          bne   MEMERROR      ;if any bits are different, give up!!!
C439:B1 02          69          lda   ($02),y       ;restore correct pattern
C43B:CA             70          dex                 ;keep x in the range 0-4
C43C:10 02   C440   71          bpl   memB
C43E:A2 04          72          ldx   #4
C440:C8             73 memB     iny                 ;all 256 filled yet?
C441:D0 EE   C431   74          bne   memA          ;branch if not
C443:E6 01          75          inc   1             ;bump page #
C445:D0 CB   C412   76          bne   mem7          ;loop through $0100 to $FF00
C447:6A             77          ror   a             ;change ACC for next pass
C448:2C 19 C0       78          bit   rdvblbar      ; use RDVBL for a little randomness...
C44B:10 02   C44F   79          bpl   memC
C44D:49 A5          80          eor   #$A5
C44F:C6 04          81 memC     dec   $04           ;have 5 passes been done yet?
C451:30 03   C456   82          bmi   memD          ;skip if yes
C453:4C D0 C3       83          jmp   mem1          ;start next pass


C456:AA             85 memD     TAX                 ;save acc
C457:2C 13 C0       86          BIT   rdramrd       ;main or aux ram ?
C45A:30 10   C46C   87          BMI   MEMF          ;skip if aux ram
C45C:8A             88          txa
C45D:8D 05 C0       89          STA   wrcardram     ;enable aux mem write
C460:8D 03 C0       90          STA   rdcardram     ;enable aux mem read
C463:8D 09 C0       91          STA   setaltzp      ;swap in alt zero page
C466:8D 81 C0       92          STA   ROMIN         ;Force rom enable
C469:4C B2 D4       93          jmp   TSTZPG        ; and test it!

C46C:8D 08 C0       95 MEMF     STA   setstdzp      ;swap in main zero page
C46F:4C EF C4       96          JMP   SWCHTST
```

```
C472:38              98 MEMERROR  sec                ;indicate main ram failure
C473:AA              99 BADBITS   tax                ;save bit pattern in x for now
C474:AD 13 C0       100           lda   rdramrd      ;main or aux mem?
C477:B8             101           clv                ;with V-FLG
C478:10 03   C47D   102           bpl   bbits1       ;branch if primary bank
C47A:2C 2A C8       103           bit   setv
C47D:A9 A0          104 bbits1    lda   #$A0         ;try to clear video screen
C47F:A0 06          105           ldy   #6
C481:99 FE BF       106 clrsts    sta   ioadr-2,y
C484:99 06 C0       107           sta   ioadr+6,y
C487:88             108           dey
C488:88             109           dey
C489:D0 F6   C481   110           bne   clrsts
C48B:8D 51 C0       111           sta   txtset
C48E:8D 54 C0       112           sta   txtpage1
C491:99 00 04       113 clrs      sta   $400,y
C494:99 00 05       114           sta   $500,y
C497:99 00 06       115           sta   $600,y
C49A:99 00 07       116           sta   $700,y
C49D:C8             117           iny
C49E:D0 F1   C491   118           bne   clrs
C4A0:8A             119           txa                ;test for switch test failure
C4A1:F0 27   C4CA   120           beq   BADSWTCH     ;branch if it was a switch
C4A3:A0 03          121           ldy   #3
C4A5:B0 02   C4A9   122           bcs   badmain      ;branch if ZP ok
C4A7:A0 05          123           ldy   #5
C4A9:A9 AA          124 badmain   lda   #$AA         ;mark aux report with an asterisks
C4AB:50 03   C4B0   125           bvc   badprim
C4AD:8D B0 05       126           sta   screen-8
C4B0:B9 66 C8       127 badprim   lda   rmess,y
C4B3:99 B1 05       128           sta   screen-7,y
C4B6:88             129           dey
C4B7:10 F7   C4B0   130           bpl   badprim      ;message is either "RAM" or "RAM ZP"
C4B9:A0 10          131           ldy   #$10         ;print bits
C4BB:8A             132 bbits2    txa
C4BC:4A             133           lsr   a
C4BD:AA             134           tax
C4BE:A9 58          135           lda   #$58         ;bits are printed as ascii 0 or 1
C4C0:2A             136           rol   a
C4C1:99 B6 05       137           sta   screen-2,y
C4C4:88             138           dey
C4C5:88             139           dey
C4C6:D0 F3   C4BB   140           bne   bbits2
C4C8:F0 FE   C4C8   141 hangx     beq   hangx        ;hang forever and ever
```

```
C4CA:A2 02          143 BADSWTCH  ldx   #2
C4CC:7A             144           ply
C4CD:08             145           php
C4CE:BD 6C C8       146 bswtch1   lda   smess,x       ;anticipate MMU error
C4D1:28             147           plp
C4D2:08             148           php
C4D3:90 03   C4D8   149           bcc   bswtch2       ;branch if not IOU error
C4D5:BD 6F C8       150           lda   smess+3,x     ;anticipate IOU error
C4D8:C0 06          151 bswtch2   cpy   #6            ;compare with where we left off
C4DA:90 0B   C4E7   152           bcc   bswtch3       ;skip if MMU
C4DC:C0 08          153           cpy   #8
C4DE:90 04   C4E4   154           bcc   bswtch2a      ;skip if GLU (ioudis or dhires failure)
C4E0:C0 11          155           cpy   #$11
C4E2:90 03   C4E7   156           bcc   bswtch3       ;skip if IOU
C4E4:BD 72 C8       157 bswtch2a  lda   smess+6,x     ;GLU error (ioudis failure)
C4E7:9D B8 05       158 bswtch3   sta   screen,x
C4EA:CA             159           dex
C4EB:10 E1   C4CE   160           bpl   bswtch1       ;print "MMU", "IOU" or "GLU"
C4ED:30 FE   C4ED   161 hangy     bmi   hangy         ;branch forever


C4EF:A0 01          163 SWCHTST   ldy   #MMUIDX
C4F1:A9 7F          164 swtst1    lda   #$7F
C4F3:6A             165 swtst2    ror   a             ;set switches of the IOU/MMU to match
                                                       Accumulator
C4F4:BE 2F C8       166           ldx   SWTBL0,y
C4F7:F0 0F   C508   167           beq   swtst4        ;branch if done setting switches
C4F9:90 03   C4FE   168           bcc   swtst3        ;branch if setting switch to 0-state
C4FB:BE 41 C8       169           ldx   SWTBL1,y      ;else get index to set switch to 1
C4FE:9D FF BF       170 swtst3    sta   ioadr-1,x     ;set switch
C501:C8             171           iny
C502:D0 EF   C4F3   172           bne   swtst2        ;branch always taken...
C504:               173 *
C504:AE 30 C0       174 click     ldx   spkr
C507:2A             175           rol   a
C508:88             176 swtst4    dey
C509:BE 53 C8       177           ldx   RSWTBL,y      ;now verify the settings just made
C50C:F0 13   C521   178           beq   swtst6        ;branch if done this pass
C50E:30 F4   C504   179           bmi   click         ;branch if this switch no to be
verified.
C510:2A             180           rol   a
C511:90 07   C51A   181           bcc   swtst5
C513:1E 00 C0       182           asl   ioadr,x
C516:90 1F   C537   183           bcc   swerr
C518:B0 EE   C508   184           bcs   swtst4        ;branch always
C51A:1E 00 C0       185 swtst5    asl   ioadr,x
C51D:B0 18   C537   186           bcs   swerr
C51F:90 E7   C508   187           bcc   swtst4        ;branch always
C521:               188 *
C521:2A             189 swtst6    rol   a             ;restore original value
C522:C8             190           iny                 ; and IOU/MMU index
C523:38             191           sec
C524:E9 01          192           sbc   #1            ;try next pattern
C526:B0 CB   C4F3   193           bcs   swtst2
C528:88             194           dey                 ;was MMU just tested?
C529:F0 08   C533   195           beq   swtst7        ;yes, go test IOU
C52B:C0 08          196           cpy   #IOUIDX-1     ;was IOU just tested?
C52D:D0 10   C53F   197           bne   BIGLOOP       ;no, go loop again
C52F:A0 11          198           ldy   #GLUIDX       ;yes, go test IOUDIS switch
C531:D0 BE   C4F1   199           bne   swtst1        ;branch always
```

```
C533:A0 09          200 swtst7    ldy   #IOUIDX
C535:D0 BA    C4F1  201           bne   swtst1        ;branch always
C537:               202 *
C537:5A             203 swerr     phy                 ;save y to distinguish from MMU or GLU
                                                       failure
C538:A2 00          204           ldx   #0            ;indicate switch error
C53A:C0 0A          205           cpy   #IOUIDX+1     ;set carry if IOU was cause
C53C:4C 7D C4       206           jmp   bbits1
```

```
C53F:46 80          208 BIGLOOP   lsr   $80
C541:D0 AC   C4EF   209           bne   SWCHTST
C543:A9 A0          210 blp2      lda   #$A0
C545:A0 00          211           ldy   #0
C547:99 00 04       212 blp3      sta   $400,y        ;clear screen for success message
C54A:99 00 05       213           sta   $500,y
C54D:99 00 06       214           sta   $600,y
C550:99 00 07       215           sta   $700,y
C553:C8             216           iny
C554:D0 F1   C547   217           bne   blp3
C556:AD 61 C0       218 blp4      LDA   butn0         ;test for both Open and Closed Apple
C559:2D 62 C0       219           AND   butn1         ; pressed
C55C:0A             220           asl   a             ;put result in carry
C55D:E6 FF          221           INC   $FF
C55F:A5 FF          222           LDA   $FF
C561:90 03   C566   223           bcc   dquit
C563:4C A9 D4       224           jmp   DIAGS
C566:               225 *
C566:AD 51 C0       226 dquit     lda   txtset        ;put success message on the screen
C569:A0 08          227           ldy   #8
C56B:B9 75 C8       228 suc2      lda   success,y
C56E:99 B8 05       229           sta   SCREEN,y
C571:88             230           dey
C572:10 F7   C56B   231           bpl   suc2
C574:30 E0   C556   232           bmi   blp4          ;loop forever
C576:        000A    54           ds    $C580-*,0     ;Appletalk stuff
C580:                55           include switcher2   ;Bank switch stuff @ 2:C780
```

```
C580:         0200     2           ds    $C780-*,$00
C780:                  3 ****************************************
C780:                  4 *
C780:                  5 * SWITCHING ROUTINES
C780:                  6 *
C780:                  7 ****************************************
C780:8D 28 C0          8 swrti2    sta   rombank
C783:40                9           rti
C784:8D 28 C0         10 swrts2    sta   rombank
C787:60               11           rts
C788:8D 28 C0         12 swreset2  sta   rombank
C78B:4C 62 FA         13           jmp   reset
C78E:8D 28 C0         14 swirq2    sta   rombank       ;Irq entry
C791:2C 87 C7         15           bit   swrtsop
C794:4C 04 C8         16           jmp   irqent
C797:8D 28 C0         17 swsthk2   sta   rombank
C79A:4C 80 C8         18           jmp   pcnv
C79D:8D 28 C0         19 swzzqt2   sta   rombank       ;Mouse basic routines
C7A0:4C 00 D4         20           jmp   basicin
C7A3:8D 28 C0         21           sta   rombank       ;Set terminal mode
C7A6:4C F1 C7         22           jmp   swsttm3
C7A9:8D 28 C0         23           sta   rombank       ;Jump to command routine
C7AC:4C 06 C8         24           jmp   swcmd3
C7AF:8D 28 C0         25           sta   rombank       ;Aux move
C7B2:4C 4E C3         26           jmp   moveaux
C7B5:8D 28 C0         27           sta   rombank       ;XFER
C7B8:4C 97 C3         28           jmp   xfer
C7BB:8D 28 C0         29           sta   rombank       ;Mouse interrupt handler
C7BE:4C 00 C1         30           jmp   mouseint
C7C1:8D 28 C0         31           sta   rombank       ;Diagnostics
C7C4:4C A9 D4         32           jmp   diags
C7C7:8D 28 C0         33           sta   rombank       ;Appletalk
C7CA:4C 80 C5         34           jmp   atalk
C7CD:8D 28 C0         35           sta   rombank       ;Serial output
C7D0:4C 4F C2         36           jmp   serout3
C7D3:8D 28 C0         37           sta   rombank       ;Get status
C7D6:4C AC C2         38           jmp   getstat
C7D9:8D 28 C0         39           sta   rombank       ;Read from serial port
C7DC:4C C3 C2         40           jmp   xrdser
C7DF:8D 28 C0         41           sta   rombank       ;Get char from buffer
C7E2:4C F7 C2         42           jmp   getbuf
C7E5:8D 28 C0         43           sta   rombank
C7E8:4C E0 D4         44           jmp   zznm
C7EB:8D 28 C0         45 swxfgo2   sta   rombank       ;Go to users xfer dest
C7EE:6C ED 03         46           jmp   ($3ED)
C7F1:DA               47 swsttm3   phx                  ;Save X
C7F2:20 16 C8         48           jsr   getlc
C7F5:5A               49           phy
C7F6:20 A0 D1         50           jsr   setterm
C7F9:80 13   C80E     51           bra   fixlc         ;Fix Language card and return

C7FB:         0008    53           ds    $C803-*,0     ;$C803 interrupt entry point
C803:4C 8E C7         54           jmp   swirq2

C806:DA               56 swcmd3    phx                  ;Go to the command routine
C807:20 16 C8         57           jsr   getlc         ;Get language card state
C80A:5A               58           phy                  ;Save it
C80B:20 00 D0         59           jsr   command
```

```
C80E:FA              60 fixlc     plx
C80F:FE 00 C0        61           inc   $C000,x       ;Restore LC
C812:FA              62           plx                  ;Restore real X
C813:4C 84 C7        63           jmp   swrts2

C816:                65 ****************************************
C816:                66 * GETLC - Gets language card state in Y
C816:                67 ****************************************
C816:        C816    68 getlc     equ   *
C816:A0 81           69           ldy   #$81
C818:2C 12 C0        70           bit   rdlcram       ;Language card enabled?
C81B:10 0C   C829    71           bpl   glcdone
C81D:A0 8B           72           ldy   #$8B
C81F:2C 11 C0        73           bit   rdlcbnk2      ;Bank 2?
C822:10 02   C826    74           bpl   glcbnk1
C824:A0 83           75           ldy   #$83          ;Bank 1!
C826:8D 81 C0        76 glcbnk1   sta   romin
C829:60              77 glcdone   rts


C82A:                79 * Diagnostic routine tables
C82A:        C82A    80 setv      equ   *
C82A:53 43 2B 29     81 ntbl      dfb   83,67,43,41,7
C82F:00 89 03 05     82 swtbl0    dfb   $00,$89,$03,$05,$09,$01,$7F,$5F
C837:00 83 51 53     83           dfb   $00,$83,$51,$53,$55,$57,$0F,$0D,$00,$80
C841:00 81 04 06     84 swtbl1    dfb   $00,$81,$04,$06,$0A,$02,$7F,$60
C849:00 84 52 54     85           dfb   $00,$84,$52,$54,$56,$58,$10,$0E,$00,$7F
C853:00 11 13 14     86 rswtbl    dfb   $00,$11,$13,$14,$16,$18,$FF,$7F
C85B:00 12 1A 1B     87           dfb   $00,$12,$1A,$1B,$1C,$1D,$1E,$1F,$00,$7E,$00
C866:                88           MSB   ON
C866:D2 C1 CD A0     89 rmess     asc   "RAM          ZP"
C86C:CD CD D5 C9     90 smess     asc   "MMUIOUGLU"


C875:D3 F9 F3 F4     92 success   asc   "System       OK"
C87E:        0002    56           ds    $C880-*,0     ;Protocol converter
C880:        0780    57           ds    $D000-*,0
D000:                58           include command     ;Serial port command processor
```

```
D000:                  2 ************************************************
D000:                  3 * The command routine now supports 5 new 2-character commands.  These
D000:                  4 * commands enable or disable a feature of the serial port and are
D000:                  5 * derived from their equivalent in the super serial card for the //.
D000:                  6 *
D000:                  7 * The new commands are as follows:
D000:                  8 *       L - send LF out after CR
D000:                  9 *       X - detect XOFF, and wait for XON
D000:                 10 *       F - accept keyboard input
D000:                 11 *       M - ignore LF in after CR
D000:                 12 *       C - auto CR when column count > printer width
D000:                 13 *
D000:                 14 * Usage of location $779 (port 1) and $77A (port 2) are as follows:
D000:                 15 *       bit 7 - echo output to screen if on
D000:                 16 *       bit 6 - generate LF after CR if on
D000:                 17 *       bit 5 - accept XOFF if on
D000:                 18 *       bit 4 - ignore keyboard input if on
D000:                 19 *       bit 3 - accept LF in after CR if on
D000:                 20 *       bit 2 - a character was received through the ACIA and is in
D000:                 21 *               location $5FE (port 1) or $67E (port 2) if on
D000:                 22 *       bit 1 - XOFF is accepted, awaiting XON if on
D000:                 23 *       bit 0 - signifies comm port if on, printer port if off
D000:                 24 ************************************************

D000:          000D   26 charCR    equ   $0D
D000:          0000   27 ucspace   equ   $00            ;need an "upper case" space character

D000:48               29 command   pha                  ;shove character on stack
D001:3C B8 03         30           bit   sermode,x      ;Already in command?
D004:30 1C    D022    31           bmi   incmd          ;If so, go do it
D006:BC 38 06         32           ldy   eschar,x       ;If eschar = 0 ignore commands
D009:F0 14    D01F    33           beq   nocmd
D00B:5D 38 06         34           eor   eschar,x       ;Is it the command char?
D00E:0A               35           asl   A              ;Ignore high bit
D00F:D0 0E    D01F    36           bne   nocmd          ;char not command char
D011:AC FB 07         37 command1  ldy   cursor         ;Save the cursor
D014:8C 79 06         38           sty   oldcur
D017:A0 BF            39           ldy   #cmdcur        ;Set command cursor
D019:8C FB 07         40           sty   cursor
D01C:4C B5 D0         41           jmp   cominit1       ;initiate command mode

D01F:38               43 nocmd     sec                  ;Mark char not handled
D020:68               44 nocmd2    pla                  ;Restore original char
D021:60               45           rts

D022:          D022   47 incmd     equ   *              ;Command mode
D022:BC 42 C1         48           ldy   devno2,x       ;Get index for ACIA
D025:29 5F            49           and   #$5F           ;High bit doesn't matter, upshift lower
case
D027:48               50           pha                  ;save character
D028:BD B8 03         51           lda   sermode,x      ;need to see if in 2-chr command
D02B:89 08            52           bit   #$08           ;bit 3 set if so
D02D:D0 03    D032    53           bne   incmd2         ;branch if so
D02F:68               54           pla                  ;pull char back, not in 2-chr cmd
D030:80 52    D084    55           bra   incmd1         ;go on with regular command mode

D032:          D032   57 incmd2    equ   *              ;handle 2nd chr of 2-chr commands
D032:68               58           pla                  ;pull char off stack
D033:48               59           pha                  ; & reshove it to keep stack neat
```

```
D034:C9 00          60           cmp   #ucspace      ;is it a space? (uppercased)
D036:D0 04   D03C   61           bne   incmd3        ;no, go on with 2-chr cmd handling
D038:18             62           clc                 ;yes, ignore spaces between chrs of
                                                      2-chr cmds
D039:68             63           pla                 ;pull uppercased char off stack
D03A:80 E4   D020   64           bra   nocmd2        ;ie mark them "handled"

D03C:BD B8 03       66 incmd3    lda   sermode,x     ;get sermode back
D03F:48             67           pha                 ;save sermode for a minit
D040:29 07          68           and   #7            ;throw out all but bits 0-2
D042:8D F8 06       69           sta   temp          ;save - this is index of which cmd it is
D045:68             70           pla                 ;get sermode back
D046:29 F0          71           and   #$F0          ;now clear bits 0-3
D048:9D B8 03       72           sta   sermode,x     ;since we're done with them now
D04B:68             73           pla                 ;get character back
D04C:DA             74           phx                 ;shove x (Cn) on stack
D04D:AE F8 06       75           ldx   temp          ;get index to command's 1st chr
D050:C9 45          76           cmp   #$45          ;is it an E?
D052:F0 71   D0C5   77           beq   enable        ;yes
D054:C9 44          78           cmp   #$44          ;no, is it a D?
D056:F0 6F   D0C7   79           beq   disable       ;yes
D058:FA             80           plx                 ;retrieve X=Cn
D059:DA             81           phx                 ;push it back to keep stack neat
D05A:DD 38 06       82           cmp   eschar,x      ;compare to the command character
D05D:08             83           php                 ;save result of comparison for a bit
D05E:AE F8 06       84           ldx   temp          ;reload X= index to cmd's first chr
D061:28             85           plp                 ;retrieve result
D062:F0 13   D077   86           beq   flagit        ;yes tis 1-chr cmd followd by nother cmd
D064:C9 0D          87           cmp   #charCR       ;is it a (guess what) CR?
D066:F0 17   D07F   88           beq   oneletter     ;yes - a 1-chr command
D068:        D068   89 cmd2null   equ   *             ;unimplemented but legal 2-chr cmds
D068:FA             90           plx                 ;pull x (Cn) off stack
D069:AD 79 06       91           lda   oldcur        ;restore non-cmd-mode cursor
D06C:8D FB 07       92           sta   cursor
D06F:1E B8 03       93           asl   sermode,x     ;clear cmd-mode bit (bit 7 of sermode)
D072:5E B8 03       94           lsr   sermode,x     ;by shifting out bit 7 & shifting in a 0
D075:80 A8   D01F   95           bra   nocmd         ;return marking character not handled

D077:        D077   97 flagit     equ   *             ;come here if get eschar after LXFM or T
D077:FA             98           plx                 ;need X=Cn to set bit 0 of sermode
D078:DA             99           phx                 ;but leave Cn on stack too
D079:FE B8 03      100           inc   sermode,x     ;bit 0 was 0, is now 1 — means new cmd
                                                      mode
D07C:AE F8 06      101           ldx   temp          ;reload X=index to cmd's first chr
D07F:        D07F  102 oneletter  equ   *             ;come here if 2-chr cmd turns out 1 chr
D07F:BD 25 D2      103           lda   cmd2list,x    ;get command chr
D082:80 0B   D08F  104           bra   backto1       ;treat it as if we just got it

D084:        D084  106 incmd1     equ   *             ;in command mode, not 2-chrs tho
D084:DA            107           phx                 ;Save slot
D085:A2 04         108           ldx   #4            ;check 5 possible 2-chr cmds
D087:DD 25 D2      109 cmd2loop  cmp   cmd2list,x    ;is it there?
D08A:F0 71   D0FD  110           beq   cmd2found     ;yes, need to flag it for next time
D08C:CA            111           dex                 ;nope
D08D:10 F8   D087  112           bpl   cmd2loop      ;try next if there is one
D08F:        D08F  113 backto1    equ   *             ;come here to check for 1-chr cmds
D08F:A2 0C         114           ldx   #12           ;Check 13 commands
D091:DD 18 D2      115 cmdloop   cmp   cmdlist,x
D094:F0 74   D10A  116           beq   cmfound       ;Right char?
D096:CA            117           dex
```

```
D097:10 F8   D091  118           bpl   cmdloop
D099:FA            119           plx                 ;We didn't find it
D09A:68            120           pla
D09B:48            121           pha
D09C:29 7F         122           and   #$7F          ;if char is cntl char
D09E:C9 20         123           cmp   #$20          ;it can be the new comd char
D0A0:B0 03   D0A5  124           bcs   ckdig         ;branch if not cntl character
D0A2:9D 38 06      125 cmdz2     sta   eschar,x      ;Save comd char, drop thru ckdig to
                                                     cdone
D0A5:49 30         126 ckdig     eor   #$30          ;zap it down to 0n if char was a digit
D0A7:C9 0A         127           cmp   #$0A          ;if not a digit, it is unexpected
                                                     intruder
D0A9:B0 33   D0DE  128           bcs   cdone         ;If not, branch
D0AB:A0 0A         129           ldy   #10           ;A = A + 10 * current number
D0AD:6D 7E 04      130 digloop   adc   number        ;C=0 on first entry
D0B0:88            131           dey
D0B1:D0 FA   D0AD  132           bne   digloop
D0B3:80 0A   D0BF  133           bra   cominit       ;not starting new cmd mode, just save #

D0B5:        D0B5  135 cominit1   equ   *             ;start new cmd mode here
D0B5:BD B8 03      136           lda   sermode,x     ;get sermode
D0B8:29 C0         137           and   #$C0          ;clear bits 0-5 (starting a new cmd seq
—
D0BA:9D B8 03      138           sta   sermode,x     ;they are used for misc during cmd mode)
D0BD:A9 00         139           lda   #0            ;load a 0 to stuff in NUMBER
D0BF:8D 7E 04      140 cominit   sta   number
D0C2:38            141           sec                 ;Mark in command mode
D0C3:80 25   D0EA  142           bra   cmset

D0C5:        D0C5  144 enable     equ   *             ;got a 2-chr command aE
D0C5:38            145           sec                 ;set carry
D0C6:90            146           dfb   $90           ;bcc to skip next byte (the CLC)
D0C7:        D0C7  147 disable    equ   *             ;got a 2-chr command aD
D0C7:18            148           clc                 ;clear carry
D0C8:08            149           php                 ;push P to save carry
D0C9:E0 00         150           cpx   #0            ;if X=0 then command is LE or LD
D0CB:F0 27   D0F4  151           beq   cmd21         ;so just make it act like L or K
D0CD:E0 04         152           cpx   #4            ;if X=4 then command is CE or CD
D0CF:F0 41   D112  153           beq   cmd.c         ;skip if so

D0D1:              155 ************************************************
D0D1:              156 * for other 2-chr cmds, their FLAGS masks' indexes are 2X+3
D0D1:              157 * for an E or 2X+4 for a D
D0D1:              158 ************************************************

D0D1:8A            160           txa                 ;copy x to acc for arithmetic
D0D2:18            161           clc                 ;clear carry for arithmetic
D0D3:0A            162           asl   A             ;multiply index by 2
D0D4:69 03         163           adc   #3            ;add 3 to get mask index
D0D6:AA            164           tax                 ;put mask index in X
D0D7:28            165           plp                 ;get carry back
D0D8:B0 01   D0DB  166           bcs   xready        ;carry set = Enable so X is ready
D0DA:E8            167           inx                 ;cmd was Disable so inc X to next mask
D0DB:4C 39 D1      168 xready    jmp   cmdi          ;go do mask stuff to FLAGS

D0DE:        D0DE  170 cdone      equ   *             ;sermode bit 0 tells whether to set or
                                                     clear cmd mode
D0DE:BD B8 03      171           lda   sermode,x     ;so get it
D0E1:4A            172           lsr   A             ;shift bit 0 to carry
D0E2:B0 D1   D0B5  173           bcs   cominit1      ;if set, start new cmd mode
D0E4:AD 79 06      174           lda   oldcur        ;Restore the cursor
D0E7:8D FB 07      175           sta   cursor        ;& fall through to cmset with carry
                                                     clear
```

```
D0EA:08              176 cmset     php
D0EB:1E B8 03        177           asl   sermode,x      ;set command mode according to carry
D0EE:28              178           plp
D0EF:7E B8 03        179           ror   sermode,x      ;leaves carry clear
D0F2:68              180           pla                  ;character handled
D0F3:60              181           rts                  ;because carry clear...

D0F4:        D0F4    183 cmd2l     equ   *              ;come here to handle LE & LD
D0F4:A9 4C           184           lda   #$4C           ;make LE look like L
D0F6:28              185           plp                  ;get P back with carry indicating E or D
D0F7:B0 96   D08F    186           bcs   backto1        ;carry set means it was an E
D0F9:A9 4B           187           lda   #$4B           ;make LD look like K
D0FB:80 92   D08F    188           bra   backto1

D0FD:8A              190 cmd2found txa                  ;copy index of cmd to acc
D0FE:FA              191           plx                  ;restore X to Cn
D0FF:1D B8 03        192           ora   sermode,x      ;copy top 2 bits of sermode
D102:09 08           193           ora   #$08          ;& set bit 3 - 2-chr-command-mode flag
D104:9D B8 03        194           sta   sermode,x      ;sermode holds index to 2-chr command
D107:38              195           sec                  ;set carry so we stay in command mode
D108:80 E0   D0EA    196           bra   cmset          ;for next time

D10A:A9 D1           198 cmfound   lda   #<cmdcr        ;get hi byte of where to go
D10C:48              199           pha                  ;save it on stack
D10D:BD F5 D1        200           lda   cmdtable,x     ;get lo byte of where to go
D110:48              201           pha                  ;save it on stack
D111:60              202           rts                  ;go there by RTSing

D112:28              204 cmd.c     plp                  ;restore status to check carry bit
D113:FA              205           plx                  ;restore slot number in x
D114:B0 05   D11B    206           bcs   cmd.c1         ;skip if enable
D116:9E B8 04        207           stz   pwdth,x        ;CD is same as PWDTH=0, no CR
D119:80 C3   D0DE    208           bra   cdone          ;we're done here

D11B:BC 86 D1        210 cmd.c1    ldy   defidx2-$C1,x  ;get y index into aux screenholes
D11E:20 2A D2        211           jsr   r.getalt       ;go get it from aux
D121:9D B8 04        212           sta   pwdth,x        ;restore default PWDTH
D124:80 B8   D0DE    213           bra   cdone          ;we're done here


D126:FA              215 cmdz      plx                  ;Zero escape character
D127:9E B8 04        216           stz   pwdth,x        ;And the width
D12A:A9 00           217           lda   #0
D12C:4C A2 D0        218           jmp   cmdz2


D12F:        D12F    220 cmdcr     equ   *
D12F:        D12F    221 cmdn      equ   *
D12F:7A              222           ply
D130:AD 7E 04        223           lda   number         ;Get number inputted
D133:F0 05   D13A    224           beq   cmdi2          ;skip if 0
D135:99 B8 04        225           sta   pwdth,y        ;Update printer width
D138:F0              226           dfb   $F0            ;BEQ opcode to skip next byte (the PLY)
D139:        D139    227 cmdi      equ   *
D139:        D139    228 cmdk      equ   *
D139:        D139    229 cmdl      equ   *
D139:7A              230           ply
D13A:B9 B8 06        231 cmdi2     lda   flags,y
```

```
D13D:3D 02 D2      232            and   mask1,x       ;Mask off bit we'll change
D140:1D 0D D2      233            ora   mask2,x       ;Change it
D143:99 B8 06      234            sta   flags,y       ;Back it goes
D146:98            235            tya                 ;Put slot back in x
D147:AA            236            tax                 ;(via acc)
D148:4C DE D0      237 cdone2     jmp   cdone         ;Good bye


D14B:88            239 cmdp       dey                 ;Make y point to command reg
D14C:A9 1F         240 cmdd       lda   #$1F          ;Mask off high three bits
D14E:38            241            sec                 ;C=1 means high 3 bits
D14F:90            242            dfb   $90           ;BCC opcode to skip next byte
D150:A9 F0         243 cmdb       lda   #$F0          ;Mask off lower 4 bits F0 = BNE
D152:18            244            clc                 ;F0 will skip this if cmdp or cmdd
D153:39 FB BF      245            and   scntl,y       ;Mask off bits being changed
D156:8D F8 06      246            sta   temp          ;Save it
D159:FA            247            plx
D15A:AD 7E 04      248            lda   number        ;Get inputed number
D15D:29 0F         249            and   #$0F          ;Only lower nibble valid
D15F:90 05   D166  250            bcc   noshift       ;If C=1 shift to upper 3 bits
D161:0A            251            asl   A
D162:0A            252            asl   A
D163:0A            253            asl   A
D164:0A            254            asl   A
D165:0A            255            asl   A
D166:0D F8 06      256 noshift    ora   temp          ;Get the rest of the bits
D169:C8            257            iny                 ;Put them in the ACIA
D16A:80 17   D183  258            bra   cmdp2         ;increment puts em away where they go.


D16C:B9 FA BF      260 cmds       lda   scomd,y       ;Transmit a break
D16F:48            261            pha                 ;Save current ACIA state
D170:09 0C         262            ora   #$0C          ;Do the break
D172:99 FA BF      263            sta   scomd,y
D175:A9 E9         264            lda   #233          ;For 233 ms
D177:A2 53         265 mswait     ldx   #83           ;Wait 1 ms
D179:48            266 msloop     pha                 ;((12*82)+11)+2+3=1000us
D17A:68            267            pla
D17B:CA            268            dex
D17C:D0 FB   D179  269            bne   msloop
D17E:3A            270            dec   a
D17F:D0 F6   D177  271            bne   mswait
D181:68            272            pla
D182:FA            273            plx
D183:        D183  274 cmdp2      equ   *
D183:99 FA BF      275            sta   scomd,y
D186:80 C0   D148  276            bra   cdone2


D188:        D188  278 cmdr       equ   *
D188:99 F9 BF      279            sta   sstat,y       ;Reset the ACIA
D18B:AD 7B 06      280            lda   vfactv        ;Check if video firmware active
D18E:0A            281            asl   A             ;Save it in C
D18F:20 97 C7      282            jsr   swsthk2       ;assume video firmware active
D192:90 03   D197  283            bcc   cmdq          ;branch if good guesser...
D194:20 9D C7      284            jsr   swzzqt2       ;Reset the hooks
D197:18            285 cmdq       clc                 ;Quit terminal mode
D198:B0            286            dfb   $B0           ;BCS to skip next byte
```

```
D199:38              287 cmdt      sec                 ;Into terminal mode
D19A:FA              288           plx                 ;Recover X
D19B:20 A0 D1        289           jsr   setterm
D19E:80 A8   D148    290           bra   cdone2


D1A0:        D1A0    292 setterm   equ   *             ;set/clear terminal mode
D1A0:BD B8 03        293           lda   sermode,x     ;Get terminal mode status
D1A3:89 40           294           bit   #$40          ;Z=1 if not in terminal mode
D1A5:90 12   D1B9    295           bcc   stclr         ;Branch if clearing terminal mode
D1A7:D0 20   D1C9    296           bne   stwasok       ;Was already set
D1A9:E4 39           297           cpx   kswh          ;Are we in the input hooks
D1AB:D0 47   D1F4    298           bne   strts         ;Leaves C=1 if =
D1AD:09 40           299           ora   #$40          ;Set term mode bit
D1AF:AC 79 06        300           ldy   oldcur        ;Save what was in oldcur
D1B2:8C 7A 06        301           sty   oldcur2
D1B5:A0 DF           302           ldy   #termcur      ;Get new cursor value
D1B7:80 07   D1C0    303           bra   stset
D1B9:F0 0E   D1C9    304 stclr     beq   stwasok       ;Branch if already clear
D1BB:29 BF           305           and   #$BF          ;Clear the bit
D1BD:AC 7A 06        306           ldy   oldcur2       ;Restore the cursor
D1C0:9D B8 03        307 stset     sta   sermode,x
D1C3:8C 79 06        308           sty   oldcur        ;Save cursor to restore later
D1C6:8C FB 07        309           sty   cursor
D1C9:BC 42 C1        310 stwasok   ldy   devno2,x
D1CC:58              311           cli                 ;want to leave with interrupts active
D1CD:08              312           php
D1CE:78              313           sei                 ;but off while we twittle bits
D1CF:B9 FA BF        314           lda   scomd,y
D1D2:09 02           315           ora   #$2           ;disable receiver interrupts if
D1D4:90 02   D1D8    316           bcc   cmdt2         ; not in terminal mode
D1D6:29 FD           317           and   #$FD          ;enable when in terminal mode
D1D8:        D1D8    318 cmdt2     equ   *
D1D8:99 FA BF        319           sta   scomd,y
D1DB:A9 00           320           lda   #0
D1DD:6A              321           ror   a             ;set kbd interrupts according to t-mode
D1DE:8D FA 05        322           sta   typhed
D1E1:10 07   D1EA    323           bpl   cmdt3         ;branch if leaving terminal mode
D1E3:9C 7F 05        324           stz   twser         ; and ser buf...
D1E6:9C 7F 06        325           stz   trser
D1E9:8A              326           txa                 ;use x to enable serial buffering
D1EA:8D FF 04        327 cmdt3     sta   aciabuf
D1ED:28              328           plp                 ;restore carry, enable interrupts.
D1EE:8E FF 05        329 flush     stx   twkey         ;Flush the type ahead buffer
D1F1:8E FF 06        330           stx   trkey
D1F4:60              331 strts     rts

D1F5:                333           MSB   OFF
D1F5:        D1F5    334 cmdtable  equ   *             ;command routines' lo bytes
D1F5:38              335           dfb   >cmdi-1
D1F6:38              336           dfb   >cmdk-1
D1F7:38              337           dfb   >cmdl-1
D1F8:2E              338           dfb   >cmdn-1
D1F9:2E              339           dfb   >cmdcr-1
D1FA:4F              340           dfb   >cmdb-1
D1FB:4B              341           dfb   >cmdd-1
D1FC:4A              342           dfb   >cmdp-1
D1FD:96              343           dfb   >cmdq-1
```

```
26 COMMAND          Command processor for serial & comm 31-MAY-85        PAGE 115

D1FE:87             344           dfb   >cmdr-1
D1FF:6B             345           dfb   >cmds-1
D200:98             346           dfb   >cmdt-1
D201:25             347           dfb   >cmdz-1

D202:               349 * masks for:     I   K   L   N   CR  XE  XD  FE  FD  ME  MD
D202:7F BF BF 7F    350 mask1     dfb   $7F,$BF,$BF,$7F,$FF,$DF,$DF,$EF,$EF,$F7,$F7
D20D:80 00 40 00    351 mask2     dfb   $80,$00,$40,$00,$00,$20,$00,$00,$10,$00,$08

D218:        D218   353 cmdlist   equ   *
D218:49 4B 4C 4E    354           asc   "IKLN"
D21C:0D             355           dfb   $0D           ;cr (part of cmdlist)
D21D:42 44 50 51    356           asc   "BDPQRSTZ"
D225:        D225   357 cmd2list  equ   *
D225:4C 58 46 4D    358           asc   "LXFMC"       ;2-chr commands' first chrs

D22A:               360 ************************************************
D22A:               361 * R.GETALT is the same as GETALT in main rom.  Only the
D22A:               362 * location is different.
D22A:               363 ************************************************

D22A:AD 13 C0       365 r.getalt  lda   rdramrd       ;save state of aux memory
D22D:0A             366           asl
D22E:AD 18 C0       367           lda   rd80col       ;and the 80STORE switch
D231:08             368           php
D232:8D 00 C0       369           sta   clr80col      ;no 80STORE to get page 1
D235:8D 03 C0       370           sta   rdcardram     ;pop in the other half of RAM
D238:B9 78 04       371           lda   $478,y        ;read the desired byte
D23B:28             372           plp                 ;and restore memory
D23C:B0 03   D241   373           bcs   r.getalt1
D23E:8D 02 C0       374           sta   rdmainram
D241:10 03   D246   375 r.getalt1 bpl   r.getalt2
D243:8D 01 C0       376           sta   set80col
D246:60             377 r.getalt2 rts

D247:03 07          379 defidx2   dfb   3,7            ;same as DEFIDX in main rom.
D249:                59           include mbasic      ;Mouse BASIC routines @ 2:C100

27 MBASIC           Mouse BASIC routines                  31-MAY-85        PAGE 116

D249:        01B7    2           ds    $D400-*,0
```

```
D400:                4 ****************************************
D400:                5 *
D400:                6 * BASICIN - Input from basic
D400:                7 *
D400:                8 * Creates  +XXXXX,+YYYYY,+SS
D400:                9 *  XXXXX = X position
D400:               10 *  YYYYY = Y position
D400:               11 *  SS = Status
D400:               12 *       - = Key pressed
D400:               13 *       1 = Button pressed
D400:               14 *       2 = Button just pressed
D400:               15 *       3 = Button just released
D400:               16 *       4 = Button not pressed
D400:               17 *
D400:               18 ****************************************
D400:         D400  19 basicin   equ   *
D400:91 28          20           sta   (basl),y      ;Fix flashing char
D402:A9 05          21           lda   #>inent       ;Fix input entry
D404:85 38          22           sta   kswl
D406:AD 00 C0       23           lda   kbd           ;test the keyboard
D409:0A             24           asl   A
D40A:08             25           php                  ;Save kbd and int stat for later
D40B:78             26           sei                  ;No interrupts while getting position
D40C:20 41 D4       27           jsr   xmread2
D40F:A0 05          28           ldy   #5            ;Move X position into the buffer
D411:AE 7C 05       29           ldx   mouxh
D414:AD 7C 04       30           lda   mouxl
D417:20 5C D4       31           jsr   hextodec      ;Convert it
D41A:A0 0C          32           ldy   #12
D41C:AE FC 05       33           ldx   mouyh
D41F:AD FC 04       34           lda   mouyl
D422:20 5C D4       35           jsr   hextodec
D425:AD 7C 07       36           lda   moustat
D428:2A             37           rol   A
D429:2A             38           rol   A
D42A:2A             39           rol   A
D42B:29 03          40           and   #3
D42D:49 03          41           eor   #3
D42F:1A             42           inc   A
D430:28             43           plp                  ;Restore int & kbd status
D431:A0 10          44           ldy   #16
D433:20 6D D4       45           jsr   hexdec2       ;X=0 from last div10
D436:7A             46           ply
D437:A2 11          47           ldx   #17           ;X = EOL
D439:A9 8D          48           lda   #$8D          ;Carriage return
D43B:9D 00 02       49 putinbuf  sta   inbuf,x
D43E:4C 84 C7       50           jmp   swrts2        ;Goback


D441:               52 ****************************************
D441:               53 *
D441:               54 * XMREAD2 - duplicate of xmread
D441:               55 *
D441:               56 ****************************************
D441:         D441  57 xmread2   equ   *
D441:A9 20          58           lda   #movarm       ;Has mouse moved?
D443:2D 7C 06       59           and   mouarm
```

```
D446:1C 7C 06       60           trb   mouarm        ;Clear arm bit
D449:2C 63 C0       61           bit   moubut        ;Button pressed?
D44C:30 02   D450   62           bmi   xrbut3
D44E:09 80          63           ora   #$80
D450:2C 7C 07       64 xrbut3    bit   moustat       ;Pressed last time?
D453:10 02   D457   65           bpl   xrbut4
D455:09 40          66           ora   #$40
D457:8D 7C 07       67 xrbut4    sta   moustat
D45A:18             68           clc
D45B:60             69           rts
D45C:               70 ****************************************
D45C:               71 *
D45C:               72 * HEXTODEC - Puts +0000, into the input buffer
D45C:               73 *  inputs: A = Low byte of number
D45C:               74 *          X = High byte of number
D45C:               75 *          Y = Position of ones digit
D45C:               76 *
D45C:               77 ****************************************
D45C:       D45C    78 hextodec  equ   *
D45C:E0 80          79           cpx   #$80          ;Is it a negative number?
D45E:90 0D   D46D   80           bcc   hexdec2
D460:49 FF          81           eor   #$FF          ;Form two's complement
D462:69 00          82           adc   #0            ;C = 1 from compare
D464:48             83           pha                  ;Save it
D465:8A             84           txa
D466:49 FF          85           eor   #$FF
D468:69 00          86           adc   #0
D46A:AA             87           tax
D46B:68             88           pla
D46C:38             89           sec
D46D:8D 14 02       90 hexdec2   sta   binl          ;Store the number to convert
D470:8E 15 02       91           stx   binh
D473:A9 2B          92           lda   #'+'          ;Store the sigh in the buffer
D475:90 02   D479   93           bcc   hdpos2
D477:A9 2D          94           lda   #'-'
D479:48             95 hdpos2    pha                  ;Save the sign
D47A:A9 2C          96           lda   #','          ;Store a comma after the number
D47C:99 01 02       97           sta   inbuf+1,y
D47F:       D47F    98 hdloop    equ   *             ;Divide by 10
D47F:               99 *
D47F:              100 * Divide BINH,L by 10 and leave remainder in A
D47F:              101 *
D47F:A2 11         102           ldx   #16+1         ;16 bits and first time do nothing
D481:A9 00         103           lda   #0
D483:18            104           clc                  ;C=0 so first ROL leaves A=0
D484:2A            105 dv10loop  rol   A
D485:C9 0A         106           cmp   #10           ;A >= 10?
D487:90 02   D48B  107           bcc   dv10lt        ;Branch if <
D489:E9 0A         108           sbc   #10           ;C = 1 from compare and is left set
D48B:2E 14 02      109 dv10lt    rol   binl
D48E:2E 15 02      110           rol   binh
D491:CA            111           dex
D492:D0 F0   D484  112           bne   dv10loop
D494:09 30         113           ora   #'0'          ;Make a ascii char
D496:99 00 02      114           sta   inbuf,y
D499:88            115           dey
D49A:F0 08   D4A4  116           beq   hddone        ;Stop on 0,6,12
D49C:C0 07         117           cpy   #7
```

```
D49E:F0 04    D4A4  118           beq   hddone
D4A0:C0 0E          119           cpy   #14
D4A2:D0 DB    D47F  120           bne   hdloop
D4A4:68             121 hddone    pla                 ;Get the sign
D4A5:99 00 02       122           sta   inbuf,y
D4A8:60             123           rts
D4A9:                60           include banger
```

```
D4A9:                  3 * These routines test all 128K ram.  All combinations of soft switches
D4A9:                  4 * applicable to the //c are tested and verified.
D4A9:                  5 *
D4A9:                  6 * In the event of any failure, the diagnostic is halted.  A message
D4A9:                  7 * is written to screen memory indicating the source of the failure.
D4A9:                  8 * When RAM fails the message is composed of "RAM ZP" (failure
D4A9:                  9 * detected in the first page of RAM) or "RAM" (other 63.75K),
D4A9:                 10 * followed by a binary representation of the failing bits set to "1".
D4A9:                 11 * For example, "RAM 0 1 1 0 0 0 0 0" indicates bits 5 and 6 were
D4A9:                 12 * detected as failing.  To represent auxillary memory, a "*" symbol is
D4A9:                 13 * printed preceeding the message.
D4A9:                 14 *
D4A9:                 15 * When the MMU or IOU fail, the message is simply "MMU" or "IOU".
D4A9:                 16 * If the IOUDIS or DHIRES switch fails, the message is "GLU".
D4A9:                 17 *
D4A9:                 18 * The test will run continuously for as long as the Open and Closed
D4A9:                 19 * Apple keys remain depressed (or no keyboard is connected) and no
D4A9:                 20 * failures are encountered.  The message "System OK" will appear in
D4A9:                 21 * the middle of the screen when a successful cycle has been run and
D4A9:                 22 * either of the Apple keys are no longer depressed.  Another cycle
D4A9:                 23 * may be initiated by pressing both Apple keys while this message
D4A9:                 24 * is on the screen.  To exit diagnostics, Control-Reset must be
D4A9:                 25 * pressed without the Apple keys depressed.
D4A9:                 26 *
D4A9:                 27 *
D4A9:          0011   28 GLUIDX    EQU   $11
D4A9:          0009   29 IOUIDX    EQU   $09
D4A9:          0001   30 MMUIDX    EQU   $01
D4A9:          05B8   31 SCREEN    EQU   $5B8
D4A9:                 32 *
D4A9:8D 50 C0         33 DIAGS     sta   txtclr         ;text mode off
D4AC:8D 78 C0         34           sta   ioudsbl        ;Disable IOU
D4AF:8D 5F C0         35           sta   setan3         ;Double hires off
D4B2:                 36 * Test Zero-Page, then all of memory.  Report errors when encountered.
D4B2:                 37 * Accumulator can be anything on entry. All registers used, but no
                           stack.
D4B2:                 38 * Addresses between $C000 and $CFFF are mapped to main $D000 bank.

D4B2:A0 04            40 TSTZPG    ldy   #$4
D4B4:A2 00            41           ldx   #0
D4B6:18               42 zp1       clc                  ;fill zero page with a pattern
D4B7:79 2A C8         43           adc   ntbl,y
D4BA:95 00            44           sta   $00,x
D4BC:E8               45           inx
D4BD:D0 F7   D4B6     46           bne   zp1            ;after all bytes filled,
D4BF:18               47 zp2       clc                  ; ACC has original value again.
D4C0:79 2A C8         48           adc   ntbl,y         ;so values can be tested
D4C3:D5 00            49           cmp   $00,x
D4C5:D0 10   D4D7     50           bne   ZPERROR        ;branch if memory failed
D4C7:E8               51           inx
D4C8:D0 F5   D4BF     52           bne   zp2            ;loop until all 256 bytes tested
D4CA:6A               53           ror   a              ;change ACC so location $FF will change
D4CB:2C 19 C0         54           bit   rdvblbar       ; use RDVBL for a little randomness...
D4CE:10 02   D4D2     55           bpl   zp3
D4D0:49 A5            56           eor   #$A5
D4D2:88               57 zp3       dey                  ;use a different pattern now
D4D3:10 E1   D4B6     58           bpl   zp1            ;branch to retest with other value
D4D5:30 06   D4DD     59           bmi   TSTMEM2        ;branch always
```

```
D4D7:55 00          61 ZPERROR   eor   $00,x          ;which bits are bad?
D4D9:18             62           clc                  ;indicate zero page failure
D4DA:4C 73 C4       63           jmp   BADBITS
D4DD:4C C6 C3       64 TSTMEM2   JMP   TSTMEM         ;Off to the rest of it


D4E0:        D4E0   66 zznm      equ   *
D4E0:20 9D C7       67           jsr   swzzqt2        ;Get out of the hooks
D4E3:68             68           pla                   ;Get junk off of stack
D4E4:7A             69           ply
D4E5:68             70           pla
D4E6:A9 FF          71           lda   #$FF
D4E8:AA             72           tax
D4E9:E8             73 zzloop    inx
D4EA:5D F5 D4       74           eor   qtbl,x
D4ED:9D 00 02       75           sta   inbuf,x
D4F0:10 F7   D4E9   76           bpl   zzloop
D4F2:4C 84 C7       77           jmp   swrts2

D4F5:AD 3B 0A 0B    79 qtbl      dfb   $AD,$3B,$0A,$0B,$48,$77,$3E,$05
D4FD:00 05 08 0C    80           dfb   $00,$05,$08,$0C,$1E,$53,$65,$37
D505:1C 07 0C 45    81           dfb   $1C,$07,$0C,$45,$62,$27,$00,$17
D50D:1C 07 07 05    82           dfb   $1C,$07,$07,$05,$4B,$6D,$24,$02
D515:0E 45 61 32    83           dfb   $0E,$45,$61,$32,$18,$02,$07,$1D
D51D:53 6A 2B 0C    84           dfb   $53,$6A,$2B,$0C,$08,$16,$53,$68
D525:3D 06 07 1B    85           dfb   $3D,$06,$07,$1B,$01,$E3
D52B:               61           include vectors2


D52B:                2 ****************************************
D52B:                3 * VECTORS
D52B:                4 ****************************************
D52B:        2ACF    5           ds    $FFFA-*,$00
FFFA:88 C7           6           dw    swreset2       ;NMI
FFFC:88 C7           7           dw    swreset2       ;RESET
FFFE:8E C7           8           dw    swirq2         ;INT
```

```
   3D A1H          3C A1L          FE78 A1PCLP     FE75 A1PC
 FE7F A1PCRTS      3F A2H            3E A2L          41 A3H
   40 A3L          43 A4H            42 A4L          45 A5H
   44 A5L          45 ACC          C1B3 ACDONE     04FF ACIABUF
 C24E ACIADONE   C1B4 ACIAINT      C1BA ACIAINT2     C1C2 ACIATST
 FDD1 ADD        FD84 ADDINP       FBF8 ADV2       ?FBF4 ADVANCE
 C24C AIAUX      C208 AIEATIT      C24D AIEAT        C200 AINOFLSH
 C20A AIPASS     C1DC AIPORT2      C1D6 AITST2       C01E ALTCHARSET
 C91D AMOD1      C93A AMOD2        C93C AMOD3        C93B AMOD4
 C94A AMOD5      C94F AMOD6        CA29 AMOD7        CA38 AMOD8
?03F5 AMPERV     C5BB APPLE2C      FB60 APPLEII      0438 ASTAT
 C580 ATALK      D08F BACKT01      C473 BADBITS      C4A9 BADMAIN
 C4B0 BADPRIM    C6A2 BADRD1       C6D3 BADREAD      C4CA BADSWTCH
 C7C1 BANGER       2B BAS2H          2A BAS2L        FBC1 BASCALC
 FBD0 BASCLC2    FEB3 BASCONT        29 BASH         E000 BASIC
 E003 BASIC2     C324 BASICENT     D400 BASICIN      C317 BASICINIT
   28 BASL       C47D BBITS1       C4BB BBITS2       FD71 BCKSPC
 FA85 BEEPSKIP   FF3A BELL        ?FBDD BELL1         FBE4 BELL2
 C53F BIGLOOP    0215 BINH         0214 BINL         C329 BINPUT
 FE00 BL1        FE04 BLANK        FCD0 BLAST      ?C543 BLP2
 C547 BLP3       C556 BLP4           4F BOOTDEV      C5F5 BOOTFAIL
   3C BOOTTMP  ?C326 BPRINT        CAF1 BRANCH     ?FA4C BREAK
 03F0 BRKV       C4CE BSWTCH1      C4D8 BSWTCH2      C4E4 BSWTCH2A
?FC10 BS         C4E7 BSWTCH3        04 BUTMODE      C061 BUTN0
 C062 BUTN1      C38A C03          C307 C3COUT1    ?C300 C3ENTRY
 C305 C3KEYIN    FD62 CANCEL       D0DE CDONE        D148 CDONE2
?CD7D CGO        F9BA CHAR1        F9B4 CHAR2        05FE CHARBUF
   0D CHARCR     C234 CHARPTR      CDCD CHK80        FBD9 CHKBELL
   24 CH         C136 CHKMOU       CB4E CHKRT        C130 CHOK
 FF7A CHRSRCH    FFCC CHRTBL       D0A5 CKDIG        FC9E CLEOLZ
 FC46 CLEOP1     C504 CLICK        CBEE CLR0         CBFC CLR1
 CBF1 CLR2       CC02 CLR3         CBC7 CLR40        C00C CLR80VID
 CBDA CLR80      C000 CLR80COL     C00E CLRALTCHAR ?C058 CLRAN0
?C05A CLRAN1   ?C05C CLRAN2      ?C05E CLRAN3       FEE9 CLRCH
 C2A5 CLRCOL     FC9C CLREOL       FC5D CLREOP1      FC44 CLREOP2
 FC42 CLREOP     CBCF CLRHALF      CD9B CLRIT        CFBD CLRKBD2
 CC99 CLRKBD     FCA0 CLRLIN       CC04 CLRPORT    ?CFFF CLRROM
 F838 CLRSC2   ?F832 CLRSCR        C481 CLRSTS       C491 CLRS
 F83C CLRSC3     F836 CLRTOP       D11B CMD.C1       D112 CMD.C
 D0FD CMD2FOUND  D225 CMD2LIST     D087 CMD2LOOP     D0F4 CMD2L
?D068 CMD2NULL   D150 CMDB         D12F CMDCR          BF CMDCUR
 D14C CMDD       D139 CMDI         D13A CMDI2        D139 CMDK
 D139 CMDL       D218 CMDLIST      D091 CMDLOOP      D12F CMDN
 D14B CMDP       D183 CMDP2        D197 CMDQ         D188 CMDR
 D16C CMDS       D1D8 CMDT2        D1EA CMDT3        D199 CMDT
 D1F5 CMDTABLE   D0A2 CMDZ2        D126 CMDZ         D10A CMFOUND
 C168 CMLOK      C14B CMLOOP       C18A CMNOINT      C1A1 CMNOVBL
 C18E CMNOY      C170 CMNT0        C175 CMRGHT       C182 CMROK
 D0EA CMSET      C155 CMXMOV       C37F CO1          FCCA COLDSTART
 0738 COL          30 COLOR        FCE6 COM1         FCF5 COM2
 FCFB COM3       D0BF COMINIT      D0B5 COMINIT1     D000 COMMAND
?D011 COMMAND1   C24F COMMPORT     C24C COMOUT       C200 COMSLOT
 CF8C COMTBL     C348 COPYROM2     C338 COPYROM      FDED COUT
 FDF0 COUT1      FDF6 COUTZ        FEF6 CRMON        FC62 CR
?FD8B CROUT1     FD8E CROUT        FC85 CRRTS          37 CSWH
   36 CSWL       CD2A CTLADR       CD54 CTLCHAR0     CD58 CTLCHAR
 FCA4 CTLDO      CD6F CTLDONE      CD71 CTLGO        CD80 CTLGO1
   14 CTLNUM     CD91 CTLOFF       CD95 CTLON        CD15 CTLTAB
```

```
 07FB CURSOR      C118 CVNOVBL      25 CV          C12B CVBUT
 C124 CVMOVED     FDB6 DATAOUT    FBBC DCX         FEE2 DECCH
 C2B6 DEFAULT     C2DF DEFCOM     C2C7 DEFFF       C2EA DEFIDX
 D247 DEFIDX2     C2BC DEFLOOP    C6D9 DENIB1      C6D7 DENIBL
 C22B DEVNO       C142 DEVNO2     D4A9 DIAGS       D0AD DIGLOOP
 FF8A DIG         D0C7 DISABLE  ?C983 DISLIN       0356 DNIBL
 CBC2 DOCLR       FBB4 DOCOUT1    FB54 DOCTL       C9D8 DOINST
 C9F4 DOLIN       C186 DONE       FD20 DONXTCUR    FECE DOPR0
 C566 DQUIT     ?C60B DRV2ENT     D484 DV10LOOP    D48B DV10LT
 D0C5 ENABLE      C219 ENTR       C111 ENTR1       F8A1 ERR
 C9C9 ERR2      ?C9CB ERR3          9B ESC       CCD7 ESC0
?CCE3 ESC1        CCE5 ESC2       CCC0 ESC3        CD0C ESCCHAR
 0638 ESCHAR      0013 ESCNUM     CCED ESCRDKEY    CCF8 ESCTAB
 C275 EXIT1       C273 EXITX      C63D EXTENT1   ?C65C EXTENT
 0538 EXTINT      05F9 EXTINT2    F800 F8ORG       FBB3 F8VERSION
 C140 FIXCH       C80E FIXLC    ?FA9B FIXSEV       D077 FLAGIT
 06B8 FLAGS     ?D1EE FLUSH       F962 FMT1        F9A6 FMT2
 CD67 FNDCTL        2E FORMAT   ?C648 FUGIT       F847 GBASCALC
   27 GBASH         26 GBASL      C8C9 GBBRK       F856 GBCALC
 C321 GBDONE      C8C1 GBNOC      C30D GBNOOVR     C8C7 GBNOTROM
 C346 GDEAT       C334 GDNOLF     C340 GDNXON      C348 GDOK
 C393 GETALT1     C398 GETALT2    C37C GETALT      C2FD GETBUF2
 C2F7 GETBUF      C3A6 GETCOUT    CCA7 GETCUR1     CCAD GETCUR2
 CCB7 GETCUR3     CCBF GETCURX    CC9D GETCUR      C322 GETDATA
 F8A5 GETFMT      C9E7 GETI1      FC80 GETINDX     C986 GETINST1
 C816 GETLC     ?FD6F GETLN1      FD67 GETLNZ    ?FD6A GETLN
 FFA7 GETNUM      C98F GETOP      C2AC GETSTAT     CB57 GETST
 C2B2 GETSTAT2    C5B4 GETUP      CEFA GETX      ?CF06 GETY
 CF38 GKEY        C826 GLCBNK1    C829 GLCDONE     0011 GLUIDX
 C5EE GOBASICIN   C8A7 GOBREAK    CB25 GODDONE     CB22 GODREG
 CB0D GODSP       C9EC GOERR2     C96E GOERR         06 GOODF8
 C278 GOREMOTE    FEB6 GO         C19B GOSER3      C279 GOTERM
?FD25 GOTKEY      F8CC GOTONE     C2C1 GSTNOINT    C2B4 GSTTST
   2C H2          C4C8 HANGX      C4ED HANGY       D4A4 HDDONE
 D47F HDLOOP      D479 HDPOS2   ?FCC9 HEADR        D46D HEXDEC2
 D45C HEXTODEC  ?C057 HIRES     ?F819 HLINE      F81C HLINE1
 FC58 HOME        CDA5 HOMECUR    CE1B HOOKITUP    CE20 HOOKUP
 F897 IEVEN       0200 INBUF      D084 INCMD1      CB05 INITBL
 0200 IN          D032 INCMD2     D022 INCMD       D03C INCMD3
 FF15 INDX        C405 INENT      C41A INITMOUSE   FB2F INIT
?FE8B INPORT      FE8D INPRT      F882 INSDS1      F88E INSDS2
 F8D0 INSTDSP     CC12 INVERT       32 INVFLG      CC1C INVX
 C000 IOADR       FEDE IOPRT1     FEAB IOPRT2      FE9B IOPRT
 C078 IOUDSBL     C079 IOUENBL    0009 IOUIDX      C058 IOU
 C82A IRQ21       C826 IRQ2       C834 IRQ3        C83E IRQ4
 C848 IRQ5        C85B IRQ6       C85E IRQ7        C870 IRQ8
 C88C IRQDN1      C88E IRQDN2     C896 IRQDN3      C89C IRQDN4
 C8A4 IRQDN5      C87F IRQDONE    C804 IRQENT    ?03FE IRQLOC
 FFFE IRQVECT   ?FA40 IRQ         C882 IRQLCOK     CF86 IRQTBLE
 C663 ISMRK1      C3C3 JMPDEST    C32C JPINIT      C32F JPREAD
 C335 JPSTAT      C332 JPWRITE    C010 KBDSTRB     C000 KBD
 FB88 KBDWAIT     FD1B KEYIN    ?FD18 KEYIN0         39 KSWH
   38 KSWL        CFDB LACR       CFD8 LADIG       CFDE LADONE
 C08B LCBANK1     C083 LCBANK2      2F LENGTH        8A LFEED
 FC66 LF          0400 LINE1      FE63 LIST2       FE5E LIST
   2C LMNEM         00 LOC0         01 LOC1        CFC5 LOOKASC
 FD38 LOOKPICK    C056 LORES      FE20 LT          FE22 LT2
?  40 M.40         20 M.CTL2       08 M.CTL         10 M.CURSOR
```

```
   08 M.GOXY          01 M.MOUSE         80 M.PASCAL        04 M.VMODE
   44 MACSTAT       C58E MAKTBL        D202 MASK1         D20D MASK2
   2E MASK          05F8 MAXH          04F8 MAXL          077D MAXXH
 067D MAXXL       ?07FD MAXYH        ?06FD MAXYL        C400 MBASIC
 C5EA MBBAD       C3D0 MEM1          C3D8 MEM2          C3F3 MEM3
 C3F5 MEM4        C3FA MEM5          C405 MEM6          C412 MEM7
 C42A MEM8        C42C MEM9          C431 MEMA          C440 MEMB
 C44F MEMC        C456 MEMD          C472 MEMERROR      C46C MEMF
 0578 MINH        C9C7 MINIERR       FE6C MINI          0478 MINL
 057D MINXH       047D MINXL       ?05FD MINYH        ?04FD MINYL
 CFA3 MIRQLP      CFBA MIRQSTD     ?C052 MIXCLR         C053 MIXSET
 0001 MMUIDX      F9C0 MNEML         FA00 MNEMR         F8BE MNNDX1
 F8C2 MNNDX2      F8C9 MNNDX3        FDAD MOD8CHK         31 MODE
 FF69 MONZ        FF65 MON           067C MOUARM        C063 MOUBUT
 C048 MOUCLR    ?C058 MOUDSBL      ?C059 MOUENBL        07FC MOUMODE
 C100 MOUSEINT    CD9F MOUSOFF       CD99 MOUSON        077C MOUSTAT
 0478 MOUTEMP     C066 MOUX1         057C MOUXH         C015 MOUXINT
 047C MOUXL       C067 MOUY1         05FC MOUYH         C017 MOUYINT
 04FC MOUYL       C972 MOV1            20 MOVARM        C34E MOVEAUX
 C361 MOVEC2M     CF9A MOVEIRQ       C367 MOVELOOP      FE2C MOVE
 C393 MOVERET     C367 MOVESTRT      C970 MOVINST         02 MOVMODE
 C900 MPADDLE     D179 MSLOOP        07F8 MSLOT         D177 MSWAIT
 CAFF NBRNCH      0300 NBUF1         FBB0 NEWADV1       FBA0 NEWADV
 FA47 NEWBRK      FC99 NEWC1         FC90 NEWCLEOLZ     FC8D NEWCLREOL
 FC73 NEWCR       CCCC NEWESC        C803 NEWIRQ      ?FA81 NEWMON
 FC38 NEWOP1      FC35 NEWOPS        CAD1 NEWPCL        FC86 NEWVTAB
 FC88 NEWVTABZ    C371 NEXTA1        03FB NMI           CA3B NNBL
 D020 NOCMD2      D01F NOCMD         C469 NOERROR       C254 NOESC
?FD45 NOESC1      FD4A NOESC2        FD44 NOESCAPE      FAA3 NOFIX
 C5AA NOPATRN     C371 NOREAD        D166 NOSHIFT       C4E6 NOSTAT2
 C36A NOT1        C1B2 NOTACIA       FD5F NOTCR1        FD4D NOTCR
 CC53 NOTINV    ?CC68 NOTINV1        CC6B NOTINV2       FEA7 NOTPRT0
 FB94 NOWAIT      C82A NTBL          047E NUMBER        0016 NUMOPS
 FCBA NXTA1       FCB4 NXTA4         FF98 NXTBAS        FF90 NXTBIT
 FFA2 NXTBS2      C9F8 NXTCH         FD75 NXTCHAR       FFAD NXTCHR
?F85F NXTCOL      077B NXTCUR        FF73 NXTITM        CA06 NXTMN
 C9BD NXTOP       FA59 OLDBRK        047B OLDCH         0679 OLDCUR
 067A OLDCUR2   ?FF59 OLDRST         D07F ONELETTER     FEC2 OPRT0
 FEFE OPTBL       057B OURCH         05FB OURCV         C407 OUTENT
?FE95 OUTPORT     FE97 OUTPRT        C1D5 P1ERR         C19E P1INIT
 C1A8 P1READ      C1AF P1READ2       C9AD P1SKIP        C1BB P1STATUS
 C1CE P1STRD      C1CC P1STWR        C1B4 P1WRITE       C211 P2INIT
 C213 P2READ      C217 P2STATUS      C215 P2WRITE       C064 PADDL0
 CF71 PASCALC   ?CF7F PASCLC2        CC0B PASINVERT     CF35 PASREAD
 C850 PASSKIP1    C23D PBFULL        C235 PBOK          F953 PCADJ
 F954 PCADJ2      F956 PCADJ3        F95C PCADJ4          3B PCH
 CAB4 PCINC2      CAB6 PCINC3          3A PCL           C5F8 PCNVRST
 C880 PCNV        CF19 PCTL          C918 PDOK          C90D PDON
 CC3D PICK1       CC33 PICK2         CC3F PICK3         CC4A PICK4
 CC1D PICKY         95 PICK          CF41 PINIT         CEBC PIORDY
 F800 PLOT        F80E PLOT1         CEC0 PNOTRDY       C402 PNULL
 FD92 PRA1        F910 PRADR1        F914 PRADR2        F926 PRADR3
 F92A PRADR4      F930 PRADR5        F94A PRBL2       ?F94C PRBL3
 F948 PRBLNK      FDDA PRBYTE      ?FB1E PREAD          FB25 PREAD2
?FF2D PRERR       CEF7 PRET        ?FDE3 PRHEX          FDE5 PRHEXZ
 F8F5 PRMN1       F8F9 PRMN2         C166 PRNOW       ?F941 PRNTAX
 F8DB PRNTBL      C14A PRNT          F8D4 PRNTOP      ?F944 PRNTX
 F940 PRNTYX        33 PROMPT        FD96 PRYX2         CF66 PS1
```

```
 CF54 PSETUP2      CF51 PSETUP       CF30 PSETX        CEB1 PSTATUS
 CEBE PSTERR      ?C070 PTRIG        C228 PUTBUF      ?D43B PUTINBUF
 CE3B PVMODE       04B8 PWDTH        CEDD PWR1         FAFD PWRCON
 03F4 PWREDUP      CEF4 PWRET        CEC2 PWRITE       CEF1 PWRITERET
 FB12 PWRUP2       FAA6 PWRUP        D4F5 QTBL         CE45 QUIT
 CE44 QX          D241 R.GETALT1     D246 R.GETALT2    D22A R.GETALT
?C060 RD40SW       C018 RD80COL      C01F RD80VID      C63F RDADR
 C016 RDALTZP      C6A8 RDAT0        C6AA RDAT1        C6BA RDAT2
 C6BC RDAT3        C6CB RDAT4        C6A6 RDATA        C003 RDCARDRAM
?FD35 RDCHAR       C642 RDDHDR       C656 RDHD0        C65E RDHD1
 C667 RDHD2        C671 RDHD3       ?C01D RDHIRES      FD0C RDKEY
 C011 RDLCBNK2     C012 RDLCRAM      C002 RDMAINRAM   ?C01B RDMIX
 C01C RDPAGE2      C013 RDRAMRD      C014 RDRAMWRT     C685 RDSEC1
 C687 RDSEC2       C68F RDSEC3       C683 RDSECT       FAE4 RDSP1
 C01A RDTEXT       C019 RDVBLBAR    ?FEFD READ         FAD7 REGDSP
 FEBF REGZ         C961 REL1        ?F938 RELADR       C955 REL
 C96B REL2         FABD RESET.X      C354 RESETLC      FA62 RESET
 FF3F RESTORE     ?FF44 RESTR1       C641 RETRY1       C657 RETRY
 FADA RGDSP1       FB02 RGDSP2       C866 RMESS         2D RMNEM
   4F RNDH           4E RNDL         C028 ROMBANK      C081 ROMIN
 C37B ROMOK        0478 ROMSTATE     C853 RSWTBL       CF94 RTBL
 F80C RTMASK       F87F RTMSKZ         2D RTNH         CAD9 RTNJMP2
?CAD5 RTNJMP         2C RTNL         F831 RTS1         FBEF RTS2B
 FB2E RTS2D        F961 RTS2         FBFC RTS3         FCC8 RTS4B
?FC34 RTS4        ?FDC5 RTS4C        FE17 RTS5        ?FCB3 RTS6
?FF4C SAV1        FF4A SAVE         BFFB SCNTL        BFFA SCOMD
 CE58 SCR1         CE5E SCR2         CE66 SCR3         CE79 SCR4
 CE82 SCR5         CE8B SCR6         CE96 SCR7        ?CE8D SCR8
 CEAD SCR9         05B8 SCREEN       CBB9 SCRL3        CB9B SCRLEVEN
 CBA2 SCRLFT       CB6D SCRLIN       CBB0 SCRLODD      F879 SCRN2
?F871 SCRN         CE80 SCRN48       CE53 SCRN84       CB30 SCROLLDN
?FC70 SCROLL       CB38 SCROLLIT     CB35 SCROLLUP     BFF8 SDATA
 C61F SEEKZERO     C27F SERIN        C11C SERISOUT     03B8 SERMODE
 C24F SEROUT3      C18A SEROUT       C18F SEROUT2      C255 SEROUT4
 C117 SERPORT      C189 SERRTS       C100 SERSLOT      C144 SERVID
 CDC0 SET40        C001 SET80COL     C00D SET80VID     CDBE SET80
 C00F SETALTCHAR   C009 SETALTZP    ?C059 SETAN0      ?C05B SETAN1
?C05D SETAN2       C05F SETAN3       C182 SETCH       ?F864 SETCOL
 FEEC SETCUR       FEEE SETCUR1      CB67 SETDBAS     ?FB40 SETGR
 CE23 SETHOOKS     FE86 SETIFLG      FE80 SETINV      ?C452 SETIOU
 CDA1 SETIT        FE89 SETKBD       FE1D SETMDZ       FE18 SETMODE
 FE84 SETNORM     ?FAA9 SETPG3       FAAB SETPLP      ?FB6F SETPWRC
 C360 SETROM       CB88 SETSRC       C008 SETSTDZP     D1A0 SETTERM
?FB39 SETTXT       CB83 SETUP2       C21C SETUP        FE93 SETVID
 C82A SETV         FB4B SETWND       CE1A SETX         CBC1 SEV1
 CC4C SHOWCUR      C5C4 SHOWINST     C28E SIDATA       C45C SILOOP
 C463 SINOCH       C280 SINOKBD      C2AC SINOMOD      C205 SIN
 CBA8 SKPLFT       CBB4 SKPRT          2B SLOTZ          C1 SLTDMY
 C86C SMESS        C46A SMINVALID    C2AA SODONE       03F2 SOFTEV
 C28D SOOK         C25E SORDY        C2AB SORTS        C286 SOTST
 C207 SOUT         C030 SPKR           49 SPNT         BFF9 SSTAT
 CF29 STARTXY        48 STATUS       D1B9 STCLR        CA43 STEP
 FE71 STEPZ        FB65 STITLE       FBF0 STORADV      C3B8 STORCH
 C3DB STORE1       C3EE STORE2      ?C3F2 STORE3       C3C1 STORE
 C3F9 STORE5      ?FE0B STOR        ?C3F7 STORE4       C3B3 STORY
 D1F4 STRTS        D1C0 STSET        D1C9 STWASOK      FFE3 SUBTBL
 C56B SUC2         C875 SUCCESS      C22F SUDODEF      C245 SUDONE
 C232 SUNODEF      C240 SUOUT       ?C7C7 SWATALK      C7AF SWAUX
```

```
 C79D SWBASICIN      C4EF SWCHTST        C7A9 SWCMD          C806 SWCMD3
 C537 SWERR          C7DF SWGETB         C7D3 SWGETST        C78E SWIRQ2
 C7BB SWMINT        ?C797 SWPCNV         C7D9 SWREAD        ?C788 SWRESET
 C788 SWRESET2       C780 SWRTI         ?C780 SWRTI2         C784 SWRTS
 C784 SWRTS2         C787 SWRTSOP        C7CD SWSER3         C797 SWSTHK2
 C7F1 SWSTHK3        C7A3 SWSTTM         C7F1 SWSTTM3        C82F SWTBL0
 C841 SWTBL1         C4F1 SWTST1         C4F3 SWTST2         C4FE SWTST3
 C508 SWTST4         C51A SWTST5         C521 SWTST6         C533 SWTST7
 C7B5 SWXFER        ?C7EB SWXFG0         C7EB SWXFG02        C7E5 SWZZNM
 C79D SWZZQT2        C7F6 SWZZQT3        C15E TAB           ?FB5B TABV
 C592 TBLLOOP        C5A0 TBLLOOP2       0578 TEMPA          06F8 TEMP
 04F8 TEMP1          05F8 TEMPY          C27C TERM1            DF TERMCUR
 C25E TESTKBD        0800 THBUF         ?FB09 TITLE          C15C TOOFAR
 FFBE TOSUB          FE6F TRACE          06FF TRKEY          067F TRSER
 C3C6 TSTMEM         D4DD TSTMEM2        D4B2 TSTZPG         05FF TWKEY
 057F TWSER          C050 TXTCLR         C054 TXTPAGE1       C055 TXTPAGE2
 C051 TXTSET         05FA TYPHED           00 UCSPACE        CC93 UD2
 CC70 UPDATE         C399 UPSHIFT0       C39B UPSHIFT        FC1A UP
 FECA USR            03F8 USRADR           2D V2             C070 VBLCLR
 C019 VBLINT           0C VBLMODE        FE36 VERIFY         067B VFACTV
 FE58 VFYOK          CE31 VIDMODE        FBFD VIDOUT         FC04 VIDOUT1
 FB78 VIDWAIT        F826 VLINEZ         F828 VLINE          04FB VMODE
 FC22 VTAB           FB59 VTAB23         FC30 VTAB40         FC24 VTABZ
 FCA9 WAIT2          FCAA WAIT3          FCA8 WAIT           FEEB WDTHCH
 CDD5 WIN0           CDE0 WIN1           CDED WIN2           CDF2 WIN3
 CE02 WIN4           CDD2 WIN40          CE18 WIN5           CDD4 WIN80
   23 WNDBTM           20 WNDLFT         CE0A WNDREST          22 WNDTOP
   21 WNDWDTH        C005 WRCARDRAM     ?FECD WRITE          C004 WRMAINRAM
 CD8D X.CUR.OFF      CD89 X.CUR.ON       CDB7 X.SI           CDB0 X.SO
 C3A5 X.UPSHIFT      FDB3 XAM            FDA3 XAM8           FDC6 XAMPM
 FEB0 XBASIC         C8E6 XBITKBD        C8F9 XBKB1          C8FB XBKB2
 CAA6 XBRK           06FB XCOORD         C3C0 XFERAZP        C3AA XFERC2M
 C3B0 XFERZP         C397 XFER           CAC9 XJMPAT         CAE3 XJMPATX
 CAC8 XJMP           CAC0 XJSR           CAEE XJXNOC         C5CF XMBASIC
 C5DC XMBOUT         C48E XMCDONE        C4BD XMCLAMP        C482 XMCLEAR
 C1AD XMDONE         C471 XMH2           C46F XMHLOOP        C46B XMHOME
 C4B6 XMRD2          C493 XMREAD         D441 XMREAD2        C4DC XMTSTINT
 C8D4 XNOKEY         C2D5 XNOSBUF          93 XOFF             91 XON
 CA98 XQ1            CA9A XQ2            CA64 XQINIT         CA50 XQNOBT0
 CA90 XQNTBRA          3C XQT            CA4A XQWAIT         C4AA XRBUT
 C4B1 XRBUT2         D450 XRBUT3         D457 XRBUT4         C2F4 XRDDONE
 C8D5 XRDKBD         C2E9 XRDNOBUF       C2C3 XRDSER         C2C9 XRDSER2
 D0DB XREADY           46 XREG           C8CC XRKBD1         C421 XRLOOP
 CAAC XRTI           CAB0 XRTS           C43B XSETMOU        C450 XSOFF
?C100 XXX            0008 YHI              47 YREG             34 YSAV
   35 YSAV1          FFC7 ZMODE          D4B6 ZP1            D4BF ZP2
 D4D2 ZP3            D4D7 ZPERROR        D4E9 ZZLOOP         D4E0 ZZNM
 CE4D ZZQUIT
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 30-APR-85 22:46
** TOTAL LINES ASSEMBLED  5727
** FREE SPACE PAGE COUNT   38
```

```
SOURCE   FILE #01 =>PC
 INCLUDE FILE #02 =>PC.EQUATES
 INCLUDE FILE #03 =>PC.BOOTSPACE
 INCLUDE FILE #04 =>PC.BOOT
 INCLUDE FILE #05 =>PC.PACKET
 INCLUDE FILE #06 =>PC.CREAD
 INCLUDE FILE #07 =>PC.MAIN
0000:         0001     1 IIc        equ   1              ;Which machine?
0000:         0001     2 ROM        equ   1              ;RAM or ROM based
0000:         C000     3 TheOrg     equ   $C000
0000:         1000     4 version    equ   $1000
0000:                  5            lst   nou
0000:                  6 *
0000:         0001     7            ifeq  IIc
0000:                  9            else
```

```
0000:                11           fin
0000:                12 *
0000:          0001  13           X6502
0000:                14 *
0000:                15 *
0000:                16 *
0000:                17 *
0000:                18 *
0000:                19 ; PPPP  RRRR   000  TTTTT  000   CCC   000  L
0000:                20 ; P   P R   R 0   0   T   0   0 C   C 0   0 L
0000:                21 ; PPPP  RRRR  0   0   T   0   0 C     0   0 L
0000:                22 ; P     R R   0   0   T   0   0 C   C 0   0 L
0000:                23 ; P     R  R   000    T    000   CCC   000  LLLLL
0000:                24 ;
0000:                25 ;  CCC   000  N   N V   V EEEEE RRRR  TTTTT EEEEE RRRR
0000:                26 ; C   C 0   0 NN  N V   V E     R   R   T   E     R   R
0000:                27 ; C     0   0 N N N V   V EEEE  RRRR    T   EEEE  RRRR
0000:                28 ; C   C 0   0 N  NN  V V  E     R R     T   E     R R
0000:                29 ;  CCC   000  N   N   V   EEEEE R  R    T   EEEEE R  R
0000:                30 ;
```

```
0000:                32 *
0000:                33 *        UniDisk 3.5 Driver Firmware   Version 1.0
0000:                34 *
0000:                35 *    Written by Michael Askins   x6243   May 15, 1985
0000:                36 *
0000:                37 *           Copyright Apple Computer, Inc. 1985
0000:                38 *                   All Rights Reserved
0000:                39 *
0000:                40 *
0000:                41           MSB   ON
0000:                42 *
```

```
0000:  44 ************************************************************
0000:  45 *                                                          *
0000:  46 *  Modification History:                                   *
0000:  47 *                                                          *
0000:  48 * Rel  Date        Who  Action                              *
0000:  49 * ---------------------------------------- *
0000:  50 * ***  18 Dec 84   MSA  RELEASE VERSION 0.02 (Sony)         *
0000:  51 *      10 Jan 85   MSA  Added //c support:                  *
0000:  52 *                         General conditional assembly overhead *
0000:  53 *      16 Jan 85   MSA  Added retries and timeouts          *
0000:  54 *                       MSlot handled correctly            *
0000:  55 *                       Finished Boot code                 *
0000:  56 *                       Altered ProDOS errors - add $27 catchall *
0000:  57 *      18 Jan 85   MSA  Remove call to WAIT in monitor      *
0000:  58 *                       Add Boot failure messages          *
0000:  59 *      22 Jan 85   MSA  Add IWM reconfigure for //c version *
0000:  60 *      23 Jan 85   MSA  Move Comm routines to $C800 ($C900) *
0000:  61 *                       Fixed zero page preservation       *
0000:  62 * ***  23 Jan 85   MSA  RELEASE VERSION 0.03 (Apple)       *
0000:  63 *      25 Jan 85   MSA  Swap slot dep read and boot code (//c) *
0000:  64 *                       Add other //c differences...       *
0000:  65 *      30 Jan 85   MSA  Add auxtype byte                    *
0000:  66 *                       Fix comm error on receive packet   *
0000:  67 *                       Fix checksum to include MSBs of overhead *
0000:  68 *      07 Feb 85   MSA  Add COUT support on boot fail       *
0000:  69 * ***  08 Feb 85   MSA  RELEASE VERSION 1.00A (alpha)       *
0000:  70 *      22 Feb 85   MSA  Add bytecount in X,Y on PC calls    *
0000:  71 *                       Change hard reset time to 1 ms (was 83) *
0000:  72 *                       Crunched code by adding ClrPhases  *
0000:  73 *                       Add zeroing of third block byte (ProDOS) *
0000:  74 *      06 Mar 85   MSA  Fixed slot 7 goof (stack screw up)  *
0000:  75 *                       No clear phases on retries         *
0000:  76 *                       Hard reset time to 40 ms           *
0000:  77 *                       Pass #parms instead of unit# and no chk *
0000:  78 *                       Init code (all reset vs. comm reset) *
0000:  79 *                       Add 2 bytes to pass a full 9 byte cmd *
0000:  80 *      16 Mar 85   MSA  Fix bytecount on retries            *
0000:  81 *                       Boot block must be $800=$01, $801<>$00 *
0000:  82 *      17 Mar 85   MSA  Remove WRREQ while waiting for motor TO *
0000:  83 *                       Remove glitch on /ENBL2 in AssignID *
0000:  84 *      20 Mar 85   MSA  Add interrupt on/off/poll support   *
0000:  85 *                       Reset pulse to 80 ms               *
0000:  86 *                       //c delay of 100 ms on initial AssignID *
0000:  87 *                       ID bytes changed                   *
0000:  88 *                       Retransmit implemented (RecPack)   *
0000:  89 *                       Add send data packet retries (5)   *
0000:  90 *                       Rearrange PC stack adjust          *
0000:  91 *                       Add //c Appletalk vector           *
0000:  92 *      24 Mar 85   MSA  Add //c millisecond wait each call  *
0000:  93 * ***  25 Mar 85   MSA  RELEASE VERSION 1.00B (beta) (//e)  *
0000:  94 *      18 Apr 85   MSA  Clear decimal mode                  *
0000:  95 *                       Eight bytes are returned on stat unit#0 *
0000:  96 *                       Stat Unit#0 scode<>0 is rejected   *
0000:  97 *                       X and Y set to 0008 on status unit#0 *
0000:  98 *                       Enable interrupts done correctly   *
0000:  99 *                       Add unit#0 parameter count checking *
0000: 100 * ***  22 Apr 85   MSA  RELEASE VERSION 1.01B               *
0000: 101 * ***  15 May 85   MSA  RELEASE VERSION 1.0                 *
```

```
0000:              102 *                                                               *
0000:              103 ****************************************************************
0000:              104 *
0000:              105 *
0000:              106           include pc.equates
```

```
0000:                    2 *
0000:          00BF      3 PDIDByte  equ   $BF          ;ProDOS attributes byte
0000:          0000      4 PCID2     equ   $0          ;This means a Liron card
0000:                    5 *
0000:                    6 ************************
0000:                    7 *                          *
0000:                    8 *  Zero Page  (temps)      *
0000:                    9 *                          *
0000:                   10 ************************
0000:                   11 *
0000:                   12           dsect
0000:          0040     13 zeropage  equ   $0040
0040:          0040     14           org   zeropage
0040:                   15 *
0040:00                 16 checksum  dfb   0
0041:00                 17 topbits   dfb   0
0042:00                 18 CMDCode   dfb   0          ;ProDOS parameter passing area
0043:          0043     19 CMDPCount equ   *
0043:00                 20 CMDUnit   dfb   0
0044:          0044     21 CMDBuffer equ   *
0044:00                 22 CMDBufferl dfb  0
0045:00                 23 CMDBufferh dfb  0
0046:          0046     24 CMDSCode  equ   *
0046:          0046     25 CMDBlock  equ   *
0046:00                 26 CMDBlockl dfb   0
0047:00                 27 CMDBlockh dfb   0
0048:00                 28 CMDBlocks dfb   0
0049:00                 29 CMDSpare1 dfb   0
004A:00                 30 CMDSpare2 dfb   0
004B:          004B     31 rcvbuf    equ   *
004B:00                 32 grp7ctr   dfb   0
004C:00                 33 oddbytes  dfb   0
004D:          004D     34 statbyte  equ   *
004D:          004D     35 bytecount equ   *
004D:          004D     36 bytecountl equ  *
004D:          004D     37 next      equ   *
004D:00                 38 next1     dfb   0
004E:          004E     39 AuxType   equ   *
004E:          004E     40 bytecounth equ  *
004E:00                 41 next2     dfb   0
004F:          004F     42 RPacketType equ *
004F:00                 43 next3     dfb   0
0050:          0050     44 DeviceID  equ   *
0050:00                 45 next4     dfb   0
0051:          0051     46 HostID    equ   *
0051:00                 47 next5     dfb   0
0052:          0052     48 pointer   equ   *
0052:00                 49 next6     dfb   0
0053:00                 50 next7     dfb   0
0054:00 00              51 buffer    dw    0
0056:          0056     52 auxptr    equ   *
0056:00 00              53 buffer2   dw    0
0058:00                 54 slot      dfb   0
0059:          0059     55 temp      equ   *
0059:00                 56 tbodd     dfb   0
005A:00                 57 Unit      dfb   0          ;Current target unit
005B:00                 58 WPacketType dfb 0
005C:                   59 *
```

```
005C:                60 *
005C:          001C  61 ZPSize    equ   *-zeropage
005C:                62 *
005C:                63 *
0000:                64           dend
0000:                65 *
0000:          CFFF  66 ClearIOROMs equ $CFFF
0000:          0100  67 stack     equ   $100
0000:                68 *
0000:                69 *
0000:                70 ************************
0000:                71 *                      *
0000:                72 *   Screenhole Storage *
0000:                73 *                      *
0000:                74 ************************
0000:                75 *
0000:                76 * The screenhole layout is as follows:
0000:                77 *
0000:                78 *                //e          //c
0000:                79 *  ------------------
0000:                80 *  ProFlag     $478+n        $478
0000:                81 *  Retry       $4F8+n        $4F8
0000:                82 *  SHTemp1     $578+n        $578
0000:                83 *  SHTempX     $5F8+n        $5F8
0000:                84 *  SHTempY     $678+n        $678
0000:                85 *  Power1      $6F8+n        --
0000:                86 *  Power2      $778+n        --
0000:                87 *  NumDevices  $7F8+n        $6FE
0000:                88 *  SvBcL       $6F8          $6F8
0000:                89 *  SvBcH       $778          $778
0000:                90 *
0000:          0001  91           do    IIc
0000:          0473  92 scholes   equ   $473          ;Use the slot 0 sholes for temps
0000:                93           else
0000:                95           fin
0000:                96 *
0000:          0473  97 ProFlag   equ   scholes
0000:          04F3  98 Retry     equ   scholes+$80
0000:          0573  99 SHTemp1   equ   scholes+$100
0000:          0573 100 Retry2    equ   SHTemp1
0000:          05F3 101 SHTempX   equ   scholes+$180
0000:          0673 102 SHTempY   equ   scholes+$200
0000:          0001 103           ifeq  IIc
0000:               107           else
0000:          06F9 108 NumDevices equ  $6F9          ;Actually in slot 6
0000:               109           fin
0000:               110 *
0000:          06F8 111 SvBcL     equ   $6F8
0000:          0778 112 SvBcH     equ   $778
0000:               113 *
0000:          0025 114 cv        equ   $25
0000:          0024 115 ch        equ   $24
0000:          FC22 116 vtab      equ   $FC22
0000:          FDED 117 cout      equ   $FDED
0000:          07DB 118 bootscrn  equ   $7DB
0000:          07F8 119 MSlot     equ   $7F8
0000:          FE93 120 setvid    equ   $FE93
0000:          FE89 121 setkbd    equ   $FE89
```

```
0000:          FABA  122 AutoScan  equ   $FABA
0000:          E000  123 Basic     equ   $E000
0000:          0000  124 loc0      equ   $0          ;Boot parms
0000:          0001  125 loc1      equ   $1          ;
0000:                126 *
0000:          C797  127 SWPROTO   equ   $C797       ;//c bank switch to $C800
0000:          C784  128 SWRTS2    equ   $C784       ;RTS to bank 1
0000:                129 *
0000:                130 *
0000:                131 ************************
0000:                132 *                      *
0000:                133 *    General Equates   *
0000:                134 *                      *
0000:                135 ************************
0000:                136 *
0000:          00A5  137 PBBValue  equ   $A5         ;Powerup Byte Base Value
0000:          00FF  138 PBCValue  equ   $FF         ;Powerup Byte Complement Value
0000:                139 *
0000:          0000  140 PowerReset equ  $00
0000:          0080  141 CommReset equ   $80
0000:                142 *
0000:          0032  143 bsyto1    equ   50          ;(.55 ms) T/O on /BSY before send
0000:          000A  144 bsyto2    equ   10          ;(.12 ms) T/O on /BSY after send
0000:          001E  145 statmto   equ   30          ;30 bytes stat mark timeout
0000:          0009  146 cmdlength equ   9           ;Command packet length
0000:          00C3  147 packetbeg equ   $C3         ;Mark at beginning of packet
0000:          00C8  148 packetend equ   $C8         ;End of packet mark
0000:          0080  149 cmdmark   equ   $80         ;Command packet identifier
0000:          0081  150 statmark  equ   $81         ;Status Packet identifier
0000:          0082  151 datamark  equ   $82         ;Data Packet identifier
0000:                152 *
0000:          0007  153 iwmmode   equ   $07         ;No timer, asynch, latch
0000:                154 *
0000:          0000  155 SCDeviceStat equ 0          ;Get Device Specific Status
0000:          0001  156 SCGetDCB  equ   1           ;Get Dev Ctrl Block (modebits)
0000:          0002  157 SCRetNLStat equ 2           ;Return Newline Status
0000:          0003  158 SCGetDevInfo equ 3          ;Get Device Info Block
0000:                159 *
0000:          C080  160 iwm       equ   $C080
0000:                161 *
0000:          C080  162 reqclr    equ   iwm+0
0000:          C081  163 reqset    equ   iwm+1
0000:          C082  164 ca1clr    equ   iwm+2
0000:          C083  165 ca1set    equ   iwm+3
0000:          C084  166 ca2clr    equ   iwm+4
0000:          C085  167 ca2set    equ   iwm+5
0000:          C086  168 lstrbclr  equ   iwm+6
0000:          C087  169 lstrbset  equ   iwm+7
0000:          C088  170 monclr    equ   iwm+8
0000:          C089  171 monset    equ   iwm+9
0000:          C08A  172 enable1   equ   iwm+10
0000:          C08B  173 enable2   equ   iwm+11
0000:          C08C  174 l6clr     equ   iwm+12
0000:          C08D  175 l6set     equ   iwm+13
0000:          C08E  176 l7clr     equ   iwm+14
0000:          C08F  177 l7set     equ   iwm+15
0000:                178 *
0000:                179 *
```

```
0000:                 180 * Error0 codes
0000:                 181 *
0000:          0001   182 noanswer   equ   1
0000:          0002   183 nomark     equ   2
0000:          0004   184 wasreset   equ   4
0000:          0008   185 bytecmp    equ   8
0000:          0010   186 csumerr    equ   $10
0000:          0020   187 nopackend  equ   $20
0000:          0040   188 bushog     equ   $40
0000:                 189 *
0000:                 190 * Command Codes
0000:                 191 *
0000:          0000   192 StatusCmd  equ   $00
0000:          0001   193 ReadCmd    equ   $01
0000:          0002   194 WriteCmd   equ   $02
0000:          0003   195 FormatCmd  equ   $03
0000:          0004   196 ControlCmd equ   $04
0000:          0005   197 InitCmd    equ   $05
0000:                 198 *
0000:                 199 *
0000:          0040   200 Soft       equ   %01000000      ;The soft error bit in statbyte
0000:                 201 *
0000:          0001   202 BadCmd     equ   $01
0000:          0004   203 BadPCnt    equ   $04
0000:          0006   204 BusErr     equ   $06
0000:          0011   205 BadUnit    equ   $11
0000:          001F   206 NoInt      equ   $1F
0000:          0021   207 BadCtl     equ   $21
0000:          0022   208 BadCtlParm equ   $22
0000:          0027   209 IOError    equ   $27
0000:          0028   210 NoDrive    equ   $28
0000:          002B   211 WriteProt  equ   $2B
0000:          002D   212 BadBlock   equ   $2D
0000:          002F   213 OffLine    equ   $2F
0000:          0068   214 LastOne    equ   Soft+NoDrive
0000:          0067   215 SoftError  equ   Soft+IOError
0000:                 216 *
0000:          0010   217 SVMask1    equ   $10
0000:                 218 *
0000:          0BB8   219 RC1        equ   3000           ;Send a command pack 3000 times (3 sec)
0000:          0005   220 RC2        equ   5              ;Data Packs (sent/rcd) get tried only 5
                                                           times
0000:                 221 *
0000:                 222 *
0000:                 107 *
0000:          0001   108            do    IIc^ROM        ;If //c ROM start is $C500
--- NEXT OBJECT FILE NAME IS CPC.0
C500:          C500   109            org   $C500
C500:                 110            else
C500:                 116            fin
C500:                 117 *
C500:                 118            include pc.bootspace
```

```
C500:                2 *lst off
C500:                3 *
C500:        0001    4          ifeq  IIc^ROM       ;If NOT the //c ROM version, do this
C500:              937          fin
C500:              938 *
```

```
C500:                 940 *
C500:          0001   941           do    IIc
C500:          0060   942 TheOff     equ   $60            ;On //c IWM in slot 6
C500:                 943           else
C500:                 945           fin
C500:                 946 *
C500:                 947 *lst on
C500:                 948 *
C500:                 949 * Here beginneth that code which resideth in the boot space
C500:                 950 *  at the time the card resteth in slot the fifth.
C500:                 951 *
C500:          C500   952 C500org    equ   *
C500:                 953 *
C500:                 954 * Auto Boot signature bytes
C500:                 955 *  This is also the boot (auto & PR#5) entry point.
C500:                 956 *
C500:A2 20            957           ldx   #$20
C502:A2 00            958           ldx   #$00
C504:A2 03            959           ldx   #$03
C506:                 960 *
C506:C9 00            961           cmp   #0             ;Flag that this is a boot
C508:          0001   962           do    IIc^ROM
C508:B0 17   C521     963           bcs   BootC
C50A:                 964           else
C50A:                 966           fin
C50A:                 967 *
C50A:                 968 * Here is the ProDOS normal entry point
C50A:                 969 *
C50A:          C50A   970 ProDOSEntry equ *
C50A:                 971 *
C50A:                 972 * Set up so that ProFLAG will have the top bit set
C50A:                 973 *
C50A:38               974           sec
C50B:B0 01   C50E     975           bcs   *+3              ;Skip the clear
C50D:                 976 *
C50D:                 977 * This is the MLIxface entry point
C50D:                 978 *
C50D:          C50D   979 MLIEntry   equ   *              ;Only use this label in //c version
C50D:18               980           clc
C50E:A2 05            981           ldx   #$05
C510:7E 73 04         982           ror   ProFLAG,x      ;ProFLAG[7]=1 if ProDOS, =0 if MLI
C513:18               983           clc                  ;This is not a boot entry
C514:                 984 *
C514:                 985 * Now save mslot and clear all $C800 ROMs
C514:                 986 *
C514:          C514   987 bootcase5 equ   *
C514:A2 C5            988           ldx   #$C5           ;Load value for MSLOT
C516:8E F8 07         989           stx   MSlot
C519:A2 05            990           ldx   #$05
C51B:AD FF CF         991           lda   ClearIOROMs    ;Clear all $C800 latches but ours
C51E:                 992 *
C51E:          0001   993           do    IIc^ROM
C51E:4C 97 C7         994           jmp   SWPROTO
C521:          C521   995 BootC      equ   *
C521:A2 05            996           ldx   #$05           ;Need slot number
C523:                 997           else
C523:                1189           fin
C523:                1190 *
```

```
C523:              1191 * lst off
C523:              1192 *
C523:        0001  1193           ifeq  IIc^ROM       ;If not the //c ROM, more boot spaces
C523:              1658           fin
C523:              1659 *lst on
C523:               119 *
C523:        0001   120           do    IIc^ROM
C523:               121           include pc.boot
```

```
C523:                  2 *
C523:          C523    3 Bootcode  equ   *
C523:86 58             4           stx   slot
C525:                  5 *
C525:          0001    6           do    IIc^ROM
C525:A9 C5             7           lda   #$C5
C527:8D F8 07          8           sta   MSlot
C52A:20 76 C5          9           jsr   reset
C52D:                 10           else
C52D:                 14           fin
C52D:                 15 *
C52D:A0 05            16           ldy   #5            ;Copy a command table
C52F:B9 70 C5         17 bc1       lda   boottab,y
C532:99 42 00         18           sta   cmdcode,y
C535:88               19           dey
C536:10 F7    C52F    20           bpl   bc1
C538:                 21 *
C538:                 22 * Now on //e, patch the Unit number (slot*16)
C538:                 23 *
C538:          0001   24           ifeq  IIc^ROM
C538:                 31           fin
C538:                 32 *
C538:                 33 * Now do the read from block zero
C538:                 34 *
C538:          0001   35           do    IIc^ROM
C538:20 0A C5         36           jsr   ProDOSEntry
C53B:                 37           else
C53B:                 39           fin
C53B:B0 15    C552    40           bcs   bootfail      ;If fail, check loc
C53D:                 41 *
C53D:AE 00 08         42           ldx   $800          ;If ($800)<>1 this is no A// boot disk
C540:CA               43           dex
C541:D0 0F    C552    44           bne   bootfail
C543:                 45 *
C543:AE 01 08         46           ldx   $801          ;If $801 is zero, no boot
C546:F0 0A    C552    47           beq   bootfail
C548:                 48 *
C548:                 49 * It all looks okay.  Jump to the code with N0 in X.
C548:                 50 *
C548:A5 58            51           lda   Slot
C54A:0A               52           asl   a
C54B:0A               53           asl   a
C54C:0A               54           asl   a
C54D:0A               55           asl   a
C54E:AA               56           tax
C54F:4C 01 08         57           jmp   $801          ;Jump to it
C552:                 58 *
C552:                 59 * Do this code if the boot can't be done.
C552:                 60 *  If this was an autoboot (loc=$CN00), continue the slot scan.
C552:                 61 *  If not, drop into basic after issuing appropriate message
C552:                 62 *
C552:                 63 *
C552:          C552   64 bootfail  equ   *
C552:                 65 *
C552:          0001   66           do    IIc
C552:A2 10            67           ldx   #>bmsglen-1
C554:          C554   68 morchrs   equ   *
C554:BD 5F C5         69           lda   bootmsg,x
```

```
C557:9D DB 07       70            sta   bootscrn,x
C55A:CA             71            dex
C55B:10 F7   C554   72            bpl   morchrs
C55D:80 FE   C55D   73 coma       bra   coma          ;He's dead Jim.
C55F:               74 *
C55F:C3 E8 E5 E3    75 bootmsg    asc   'Check        Disk Drive.'
C570:       0011    76 bmsglen    equ   *-bootmsg
C570:               77            else
C570:              131            fin
C570:              132 *
C570:01 50 00 08   133 boottab    dfb   ReadCMD,$50,0,8,0,0 ;Read from 1st; blk0->$801
C576:              134 *
C576:              135 *
C576:              136 * This routine is called from the //c reset code.  It forces a
C576:              137 *  reset of the PC Bus.
C576:              138 *
C576:       0001   139            do    IIc^ROM
C576:       C576   140 Reset      equ   *
C576:A2 08         141            ldx   #8
C578:       C578   142 rst1       equ   *
C578:BD 83 C5      143            lda   rcode,x
C57B:95 00         144            sta   loc0,x
C57D:CA            145            dex
C57E:10 F8   C578  146            bpl   rst1
C580:4C 00 00      147            jmp   loc0
C583:              148 *
C583:       C583   149 rcode      equ   *
C583:20 0D C5      150            jsr   MLIEntry
C586:05            151            dfb   InitCMD
C587:07 00         152            dw    $0007
C589:60            153            rts
C58A:              154 *
C58A:01 00         155 cmdlist   dfb   1,0           ;One parm - the unit $00
C58C:              156            fin
C58C:              157 *
C58C:              158 *
--- NEXT OBJECT FILE NAME IS CPC.1
C5F5:       C5F5   122            org   $C5F5
C5F5:4C 52 C5      123            jmp   bootfail      ;Jump to the boot failure message
C5F8:4C 76 C5      124            jmp   reset         ;Reset vector
C5FB:00            125            dfb   PCID2
C5FC:00 00         126            dw    0
C5FE:BF            127            dfb   PDIDByte
C5FF:0A            128            dfb   >ProDOSEntry
C600:              129 *
--- NEXT OBJECT FILE NAME IS CPC.2
C880:       C880   130            org   $C880
C880:4C 4B CD      131            jmp   Entry         ;The //c bank switch jumps here
C883:4C E8 CF      132            jmp   AppleTalkEntry
C886:              133            fin
C886:              134 *
C886:              135            include pc.packet
C886:                   1         lst   cyc
C886:                   2 *
```

```
C886:                    4 ************************************************************
C886:                    5 *                                                          *
C886:                    6 *  SendOnePack                    Send a CBus Packet      *
C886:                    7 *                                                          *
C886:                    8 *    This routine sends a packet of data across the      *
C886:                    9 *  bus.  The protocol is as follows:                       *
C886:                   10 *                  ----------------------------          *
C886:                   11 *  REQ _______|2                          5|_________  *
C886:                   12 *               ---------------------------------        *
C886:                   13 * /BSY ___|1              3              4|__________  *
C886:                   14 *                                                         *
C886:                   15 *    1)  Device signals ready for data                    *
C886:                   16 *    2)  Host signals data iminent                        *
C886:                   17 *    3)  Packet is transmitted (sync, command mark,       *
C886:                   18 *         ids, contents, checksum [msb=1])                *
C886:                   19 *    4)  Device signals packet recieved                   *
C886:                   20 *    5)  Host finishes send data cycle                    *
C886:                   21 *                                                         *
C886:                   22 *    The bytes are sent in slow mode (32 cycles/byte)     *
C886:                   23 *  and the timing is critical.  Branches which should     *
C886:                   24 *  not cross page boundaries are marked.                  *
C886:                   25 *                                                         *
C886:                   26 *   Input:  buffer (2 bytes) <- ptr to data to send      *
C886:                   27 *           bytecount (2)    <- length (bytes) of data   *
C886:                   28 *           packettype (1)   <- command or data packet   *
C886:                   29 *           CMDUnit (1)      <- # of device to receive   *
C886:                   30 *                                                         *
C886:                   31 *   Output: carry set- handshake error                   *
C886:                   32 *                   clr- bytes sent                      *
C886:                   33 *                                                         *
C886:                   34 ************************************************************
C886:                   35 *
C886:         C886      36 SendOnePack equ *
C886:                   37 *
C886:                   38 * Prep for the transmission
C886:                   39 *
C886:20 64 CB    (6)    40         jsr   WritePrep      ;Does a bunch of stuff
C889:                   41 *
C889:                   42 * Enable PC chain.
C889:                   43 *
C889:20 80 CA    (6)    44         jsr   enablechain    ;This sets X reg
C88C:A0 07       (2)    45         ldy   #iwmmode       ;This is the mode value
C88E:20 1F CC    (6)    46         jsr   SetIWMode      ;Don't mess unless we gotta
C891:                   47 *
C891:                   48 * Turn on the IWM
C891:                   49 *
C891:BD 8B C0    (4)    50         lda   enable2,x      ;Don't disturb //c internal drive
C894:BD 89 C0    (4)    51         lda   monset,x
C897:                   52 *
C897:                   53 * Loop until the chain becomes unbusy
C897:                   54 *
C897:A0 32       (2)    55         ldy   #bsyto1        ;Each loop is 11 microseconds
C899:BD 8E C0    (4)    56 ubsy1   lda   17clr,x        ;Test if /BSY is hi or lo
C89C:30 07    C8A5(3)   57         bmi   chainunbsy     ;If hi, bus is not busy
C89E:88          (2)    58         dey
C89F:D0 F8    C899(3)   59         bne   ubsy1          ;Keep trying
C8A1:                   60 *
C8A1:38          (2)    61         sec
```

```
C8A2:4C CF C9    (3)   62          jmp   sd10
C8A5:                  63 *
C8A5:                  64 * Tell the bus that data is coming and send the sync bytes
C8A5:                  65 *  Sync is groups of eight 2's separated by a 6 (micS cell)
C8A5:                  66 *  (11111111001111111100111111100 ...)
C8A5:                  67 *
C8A5:        C8A5      68 chainunbsy equ *
C8A5:BD 81 C0    (4)   69          lda   reqset,x      ;Raise REQ
C8A8:                  70 *
C8A8:A0 05       (2)   71          ldy   #5            ;Sync plus packet begin
C8AA:                  72 *
C8AA:A9 FF       (2)   73          lda   #$FF          ;Send out the 1st byte sync
C8AC:9D 8F C0    (5)   74          sta   17set,x
C8AF:                  75 *
C8AF:B9 D6 C9    (4)   76 ssb      lda   preamble,y
C8B2:                  77 *
C8B2:                  78 *
C8B2:1E 8C C0    (7)   79 ssd      asl   16clr,x       ;Wait 'til buffer empty
C8B5:90 FB   C8B2(3)   80          bcc   ssd
C8B7:                  81 *
C8B7:9D 8D C0    (5)   82          sta   16set,x
C8BA:88          (2)   83          dey
C8BB:10 F2   C8AF(3)   84          bpl   ssb           ;Back for more bytes
C8BD:                  85 *
C8BD:                  86 * Send over the desination ID
C8BD:                  87 *
C8BD:A5 5A       (3)   88          lda   Unit
C8BF:09 80       (2)   89          ora   #$80          ;Make the device ID
C8C1:20 53 CA    (6)   90          jsr   sendbyte
C8C4:                  91 *
C8C4:                  92 * Send the source ID (that's us... we're an $80)
C8C4:                  93 *
C8C4:20 51 CA    (6)   94          jsr   send80
C8C7:                  95 *
C8C7:                  96 * Send over the packet type (command or data)
C8C7:                  97 *
C8C7:A5 5B       (3)   98          lda   Wpackettype
C8C9:20 53 CA    (6)   99          jsr   sendbyte
C8CC:                 100 *
C8CC:                 101 * Send the Auxilliary Type byte (an $80 from this rev PC)
C8CC:                 102 *
C8CC:20 51 CA    (6)  103          jsr   send80
C8CF:                 104 *
C8CF:                 105 * Send the status byte (null for us), and length bytes
C8CF:                 106 *
C8CF:20 51 CA    (6)  107          jsr   send80
C8D2:A5 4C       (3)  108          lda   oddbytes
C8D4:09 80       (2)  109          ora   #$80
C8D6:20 53 CA    (6)  110          jsr   sendbyte
C8D9:A5 4B       (3)  111          lda   grp7ctr
C8DB:09 80       (2)  112          ora   #$80
C8DD:20 53 CA    (6)  113          jsr   sendbyte
C8E0:                 114 *
C8E0:                 115 * Now send the "oddbytes" part of the packet contents
C8E0:                 116 *
C8E0:A5 4C       (3)  117          lda   oddbytes      ;Get # of "odd" bytes
C8E2:F0 15   C8F9(3)  118          beq   sob2          ;Skip if no odd bytes
C8E4:                 119 *
```

```
C8E4:A0 FF      (2)  120         ldy   #$FF
C8E6:A5 59      (3)  121         lda   tbodd          ;Get the odd bytes msb's (A[7]=1)
C8E8:                122 *
C8E8:1E 8C C0   (7)  123 sob1    asl   l6clr,x       ;Do a write handshake
C8EB:90 FB  C8E8(3)  124         bcc   sob1
C8ED:9D 8D C0   (5)  125         sta   l6set,x
C8F0:C8         (2)  126         iny
C8F1:B1 54      (5)  127         lda   (buffer),y    ;Get the data byte
C8F3:09 80      (2)  128         ora   #$80          ;Flip on the hi bit
C8F5:C4 4C      (3)  129         cpy   oddbytes      ;Are we done?
C8F7:90 EF  C8E8(3)  130         blt   sob1
C8F9:                131 *
C8F9:                132 * Now send over the groups of seven contents
C8F9:                133 * Currently assume there must be at least one group of 'em
C8F9:                134 *
C8F9:       C8F9     135 sob2    equ   *
C8F9:A5 4B      (3)  136         lda   grp7ctr       ;Check if there are groups to send
C8FB:D0 03  C900(3)  137         bne   sob3          ;=> At least one group
C8FD:4C 99 C9   (3)  138         jmp   datdone       ;Skip to send checksum
C900:                139 *
C900:       C900     140 sob3    equ   *
C900:EA         (2)  141         nop                 ;Waste 2 cycles
C901:A0 00      (2)  142         ldy   #0
C903:A5 41      (3)  143 start   lda   topbits
C905:9D 8D C0   (5)  144         sta   l6set,x
C908:                145 *
C908:                146 * Send first byte
C908:                147 *
C908:A5 4D      (3)  148         lda   next1
C90A:09 80      (2)  149         ora   #$80
C90C:84 59      (3)  150         sty   temp          ;Swap Y for short handshake
C90E:BC 8C C0   (4)  151 ache1   ldy   l6clr,x       ;Wait 'til buffer ready
C911:10 FB  C90E(3)  152         bpl   ache1
C913:9D 8D C0   (5)  153         sta   l6set,x       ;Send the byte
C916:A4 59      (3)  154         ldy   temp          ;Get back Y
C918:                155 *
C918:                156 * Prep the next "1st" byte for next time
C918:                157 *
C918:B1 56      (5)  158         lda   (buffer2),y
C91A:85 4D      (3)  159         sta   next1
C91C:0A         (2)  160         asl   a
C91D:26 41      (5)  161         rol   topbits       ;Store the top bit
C91F:C8         (2)  162         iny                 ;Next byte
C920:                163 *
C920:                164 * It's possible that we're at a page boundary now. If so, bump the
C920:                165 *  hi order part of the pointer.
C920:                166 *
C920:D0 05  C927(3)  167         bne   skip1
C922:E6 57      (5)  168         inc   buffer2+1
C924:4C 29 C9   (3)  169         jmp   skip2
C927:48         (3)  170 skip1   pha                 ;Equalize the cases
C928:68         (4)  171         pla
C929:                172 *
C929:                173 * Push us ahead by an additional 8 cycles for margin reasons
C929:                174 *  Plus I gotta get the topbits MSB set somehow...
C929:                175 *
C929:       C929     176 skip2   equ   *
C929:A9 02      (2)  177         lda   #%00000010    ;Flip what will be MSB
```

```
C92B:05 41      (3)  178         ora   topbits
C92D:85 41      (3)  179         sta   topbits
C92F:                180 *
C92F:                181 * Send the second byte
C92F:                182 *
C92F:A5 4E      (3)  183         lda   next2
C931:09 80      (2)  184         ora   #$80
C933:9D 8D C0   (5)  185         sta   16set,x       ;Send the byte
C936:B1 56      (5)  186         lda   (buffer2),y
C938:85 4E      (3)  187         sta   next2
C93A:0A         (2)  188         asl   a
C93B:26 41      (5)  189         rol   topbits       ;Store the top bit
C93D:C8         (2)  190         iny                 ;Next byte
C93E:                191 *
C93E:                192 * Send the third byte
C93E:                193 *
C93E:A5 4F      (3)  194         lda   next3
C940:09 80      (2)  195         ora   #$80
C942:9D 8D C0   (5)  196         sta   16set,x       ;Send the byte
C945:B1 56      (5)  197         lda   (buffer2),y
C947:85 4F      (3)  198         sta   next3
C949:0A         (2)  199         asl   a
C94A:26 41      (5)  200         rol   topbits       ;Store the top bit
C94C:C8         (2)  201         iny                 ;Next byte
C94D:                202 *
C94D:                203 * Send the fourth byte
C94D:                204 *
C94D:A5 50      (3)  205         lda   next4
C94F:09 80      (2)  206         ora   #$80
C951:9D 8D C0   (5)  207         sta   16set,x       ;Send the byte
C954:B1 56      (5)  208         lda   (buffer2),y
C956:85 50      (3)  209         sta   next4
C958:0A         (2)  210         asl   a
C959:26 41      (5)  211         rol   topbits       ;Store the top bit
C95B:C8         (2)  212         iny                 ;Next byte
C95C:                213 *
C95C:                214 * After the first 256 bytes, we will cross pages here.  If we did
C95C:                215 *  cross, bump the buffer pointer. If not, equalize the cases with
C95C:                216 *  seven cycles of time wasting.
C95C:                217 *
C95C:D0 05   C963(3) 218         bne   skip3
C95E:E6 57      (5)  219         inc   buffer2+1
C960:4C 65 C9   (3)  220         jmp   skip4
C963:48         (3)  221 skip3   pha
C964:68         (4)  222         pla
C965:        C965    223 skip4   equ   *
C965:                224 *
C965:                225 * Send the fifth byte
C965:                226 *
C965:A5 51      (3)  227         lda   next5
C967:09 80      (2)  228         ora   #$80
C969:9D 8D C0   (5)  229         sta   16set,x       ;Send the byte
C96C:B1 56      (5)  230         lda   (buffer2),y
C96E:85 51      (3)  231         sta   next5
C970:0A         (2)  232         asl   a
C971:26 41      (5)  233         rol   topbits       ;Store the top bit
C973:C8         (2)  234         iny                 ;Next byte
C974:                235 *
```

```
C974:                   236 * Send the sixth byte
C974:                   237 *
C974:A5 52        (3)   238         lda   next6
C976:09 80        (2)   239         ora   #$80
C978:9D 8D C0     (5)   240         sta   l6set,x       ;Send the byte
C97B:B1 56        (5)   241         lda   (buffer2),y
C97D:85 52        (3)   242         sta   next6
C97F:0A           (2)   243         asl   a
C980:26 41        (5)   244         rol   topbits       ;Store the top bit
C982:C8           (2)   245         iny                 ;Next byte
C983:                   246 *
C983:                   247 * Send the last byte of the group
C983:                   248 *
C983:A5 53        (3)   249         lda   next7
C985:09 80        (2)   250         ora   #$80
C987:9D 8D C0     (5)   251         sta   l6set,x       ;Send the byte
C98A:B1 56        (5)   252         lda   (buffer2),y
C98C:85 53        (3)   253         sta   next7
C98E:0A           (2)   254         asl   a
C98F:26 41        (5)   255         rol   topbits       ;Store the top bit
C991:C8           (2)   256         iny                 ;Next byte
C992:                   257 *
C992:                   258 * Now see if we have sent enough groups of seven
C992:                   259 *
C992:C6 4B        (5)   260         dec   grp7ctr
C994:F0 03   C999(3)    261         beq   datdone
C996:                   262 *
C996:                   263 * Otherwise, back to do more.  Note it's too far for a branch.
C996:                   264 *
C996:4C 03 C9     (3)   265         jmp   start
C999:                   266 *
C999:                   267 * Whew! Now send the damn checksum as two FM bytes
C999:                   268 *
C999:         C999      269 datdone equ   *
C999:A5 40        (3)   270         lda   checksum      ;c7 c6 c5 c4 c3 c2 c1 c0
C99B:09 AA        (2)   271         ora   #$AA          ; 1 c6  1 c4  1 c2  1 c0
C99D:BC 8C C0     (4)   272 scm1    ldy   l6clr,x
C9A0:10 FB   C99D(3)    273         bpl   scm1          ;Handshake this byte
C9A2:9D 8D C0     (5)   274         sta   l6set,x       ;These are even bits
C9A5:                   275 *
C9A5:A5 40        (3)   276         lda   checksum      ;c7 c6 c5 c4 c3 c2 c1 c0
C9A7:4A           (2)   277         lsr   a             ; 0 c7 c6 c5 c4 c3 c2 c1
C9A8:09 AA        (2)   278         ora   #$AA          ; 1 c7  1 c5  1 c3  1 c1
C9AA:20 53 CA     (6)   279         jsr   sendbyte
C9AD:                   280 *
C9AD:                   281 * Send the end of packet mark
C9AD:                   282 *
C9AD:A9 C8        (2)   283         lda   #packetend
C9AF:20 53 CA     (6)   284         jsr   sendbyte
C9B2:                   285 *
C9B2:                   286 * Wait until write underflow
C9B2:                   287 *
C9B2:BD 8C C0     (4)   288 sd7     lda   l6clr,x
C9B5:29 40        (2)   289         and   #$40
C9B7:D0 F9   C9B2(3)    290         bne   sd7           ;Still writing data
C9B9:                   291 *
C9B9:9D 8D C0     (5)   292         sta   l6set,x       ;Back to sense mode (dummy write)
C9BC:                   293 *
```

```
C9BC:                    294 * Now wait until the drive acknowledges reciept of the
C9BC:                    295 *  string or until timeout
C9BC:                    296 *
C9BC:A0 0A       (2)     297         ldy   #bsyto2       ;Load timeout to see bsy low
C9BE:88          (2)     298 patch1 dey                   ;A little closer to an error
C9BF:D0 08   C9C9(3)     299         bne   sd9           ;There's still time
C9C1:                    300 *
C9C1:                    301 * Too much time has elapsed.  Drive didn't get string.
C9C1:                    302 *
C9C1:A9 01       (2)     303         lda   #noanswer     ;Report error in comm error byte
C9C3:        C9C3        304 dberror equ  *
C9C3:20 9A CA    (6)     305         jsr   SetXN0        ;For dberror entry
C9C6:38          (2)     306         sec                 ;Signal a problem
C9C7:B0 06   C9CF(3)     307         bcs   sd10
C9C9:                    308 *
C9C9:                    309 * See if drive has acknowledged the bytes yet
C9C9:                    310 *
C9C9:BD 8E C0    (4)     311 sd9     lda   l7clr,x       ;Wait 'til /BSY lo
C9CC:30 F0   C9BE(3)     312         bmi   patch1
C9CE:                    313 *
C9CE:                    314 * Finish the sequence
C9CE:                    315 *
C9CE:18          (2)     316         clc                 ;This is a normal exit
C9CF:BD 80 C0    (4)     317 sd10    lda   reqclr,x      ;Set REQ lo
C9D2:BD 8C C0    (4)     318         lda   l6clr,x       ;Back into read mode
C9D5:                    319 *
C9D5:                    320 * Pull back the bytecount in all cases
C9D5:                    321 *
C9D5:60          (6)     322         rts
C9D6:                    323 *
C9D6:                    324 *
C9D6:                    325 * This table, when sent in reverse order, provides a
C9D6:                    326 *  sync pattern used to synchronize the drive IWM with
C9D6:                    327 *  the data stream.  The first byte (last sent) is the
C9D6:                    328 *  packet begin mark.
C9D6:                    329 *
C9D6:C3                  330 preamble dfb packetbeg
C9D7:FF FC F3 CF         331 synctab dfb  $FF,$FC,$F3,$CF,$3F
C9DC:                    332 *
C9DC:                    333 *
```

```
C9DC:                   335 *
C9DC:                   336 * These routines are for wasting specific amounts of time
C9DC:                   337 *  This code segment should not cross page boundaries.
C9DC:                   338 *
C9DC:20 E1 C9     (6)   339 waste32 jsr  waste14
C9DF:EA           (2)   340 waste18 nop
C9E0:EA           (2)   341 waste16 nop
C9E1:EA           (2)   342 waste14 nop
C9E2:60           (6)   343 waste12 rts
C9E3:                   344 *
C9E3:                   345 *
C9E3:         C9E3      346 markerr equ  *
C9E3:4C C3 C9     (3)   347        jmp   dberror
```

```
C9E6:                  349 ********************************************************
C9E6:                  350 *                                                      *
C9E6:                  351 *  ReceivePack       Get a packet from bus resident    *
C9E6:                  352 *                                                      *
C9E6:                  353 *                  ____________________________        *
C9E6:                  354 *  REQ _______|2                          5|_________  *
C9E6:                  355 *              ____________________________            *
C9E6:                  356 * /BSY ___|1              3             4|__________  *
C9E6:                  357 *                                                      *
C9E6:                  358 *    1)  Drive signals ready to send packet            *
C9E6:                  359 *    2)  Host signals ready to recieve data            *
C9E6:                  360 *    3)  Packet is transmitted (sync, mark, IDs, data, *
C9E6:                  361 *         checksum [msb=1])                            *
C9E6:                  362 *    4)  Drive signals packet dispatched               *
C9E6:                  363 *    5)  Host acknowledges reciept of packet           *
C9E6:                  364 *                                                      *
C9E6:                  365 *    The bytes are sent in slow mode (32 cycles/byte)  *
C9E6:                  366 *  and the timing is critical.  Branches which should  *
C9E6:                  367 *  not cross page boundaries are marked.               *
C9E6:                  368 *                                                      *
C9E6:                  369 *   Input:  buffer <- address where packet guts left   *
C9E6:                  370 *                                                      *
C9E6:                  371 *   Output: carry set- handshake error                 *
C9E6:                  372 *                   clr- bytes recieved                *
C9E6.                  373 *            A <- error0 if carry set                  *
C9E6:                  374 *                                                      *
C9E6:                  375 ********************************************************
C9E6:                  376 *
C9E6:         C9E6     377 grabstatus equ *
C9E6:         C9E6     378 ReceivePack equ *
C9E6:                  379 *
C9E6:                  380 * Init the checksum
C9E6:                  381 *
C9E6:A9 00       (2)   382         lda   #$00
C9E8:85 40       (3)   383         sta   checksum
C9EA:                  384 *
C9EA:                  385 * Copy over buffer -> buffer2
C9EA:                  386 *
C9EA:A5 54       (3)   387         lda   buffer
C9EC:85 56       (3)   388         sta   buffer2
C9EE:A5 55       (3)   389         lda   buffer+1
C9F0:85 57       (3)   390         sta   buffer2+1
C9F2:                  391 *
C9F2:                  392 * Set up the indirect pointer for jump to 2nd part of code
C9F2:                  393 *
C9F2:         0001     394         ifeq  IIc^ROM       ;Don't do in //c version
C9F2:                  401         fin
C9F2:                  402 *
C9F2:20 80 CA    (6)   403         jsr   enablechain   ;Set X register to $N0
C9F5:                  404 *
C9F5:BD 8D C0    (4)   405         lda   l6set,x       ;Prep for sense mode
C9F8:                  406 *
C9F8:                  407 * Now wait for BSY to go hi, signalling 'ready w/ status'
C9F8:                  408 *
C9F8:BD 8E C0    (4)   409 rdh1    lda   l7clr,x       ;Read sense
C9FB:10 FB   C9F8(3)   410         bpl   rdh1          ;Wait til a high
C9FD:                  411 *
C9FD:                  412 * Signal Liron we're ready to recieve
```

```
C9FD:                   413 *
C9FD:BD 81 C0     (4)   414         lda   reqset,x      ;Raise /REQ
CA00:                   415 *
CA00:                   416 * Wait for a byte from Liron or timeout
CA00:                   417 *
CA00:A0 1E        (2)   418         ldy   #statmto      ;Max bytes 'til stat mark
CA02:BD 8C C0     (4)   419 rdh2    lda   16clr,x
CA05:10 FB    CA02(3)   420         bpl   rdh2          ;*** No Page Cross ***
CA07:88           (2)   421         dey
CA08:30 D9    C9E3(3)   422         bmi   markerr       ;Didn't find a packet in time
CA0A:                   423 *
CA0A:                   424 * Is it the beginning of the packet?
CA0A:                   425 *
CA0A:C9 C3        (2)   426         cmp   #packetbeg    ;Find the packet begin mark
CA0C:D0 F4    CA02(3)   427         bne   rdh2          ;Back again - no timeout for now
CA0E:                   428 *
CA0E:                   429 * Okay load up the table with this stuff
CA0E:                   430 *
CA0E:         CA0E      431 rdh5    equ   *
CA0E:                   432 *
CA0E:A0 06        (2)   433         ldy   #6            ;Seven bytes of overhead
CA10:BD 8C C0     (4)   434 rdh3    lda   16clr,x       ;If byte ready, grab it
CA13:10 FB    CA10(3)   435         bpl   rdh3          ;*** No Page Cross ***
CA15:29 7F        (2)   436         and   #%01111111    ;Strip start bit
CA17:99 4B 00     (5)   437         sta   rcvbuf,y
CA1A:49 80        (2)   438         eor   #$80          ;Pop MSB back on for checksum
CA1C:45 40        (3)   439         eor   checksum
CA1E:85 40        (3)   440         sta   checksum
CA20:88           (2)   441         dey
CA21:10 ED    CA10(3)   442         bpl   rdh3
CA23:                   443 *
CA23:                   444 * Set groups of seven buffer pointer buffer2
CA23:                   445 *
CA23:A5 4C        (3)   446         lda   oddbytes
CA25:F0 27    CA4E(3)   447         beq   start2        ;Skip alteration if no oddbytes
CA27:18           (2)   448         clc
CA28:65 54        (3)   449         adc   buffer
CA2A:85 56        (3)   450         sta   buffer2
CA2C:A5 55        (3)   451         lda   buffer+1
CA2E:69 00        (2)   452         adc   #0
CA30:85 57        (3)   453         sta   buffer2+1
CA32:                   454 *
CA32:A0 00        (2)   455         ldy   #0
CA34:                   456 *
CA34:                   457 * Now receive the odd bytes
CA34:                   458 *
CA34:BD 8C C0     (4)   459 start0  lda   16clr,x       ;Read in the odd bytes topbits
CA37:10 FB    CA34(3)   460         bpl   start0
CA39:0A           (2)   461         asl   a             ;Pop off the start bit
CA3A:85 41        (3)   462         sta   topbits
CA3C:         CA3C      463 start1  equ   *
CA3C:BD 8C C0     (4)   464         lda   16clr,x       ;Get an odd byte
CA3F:10 FB    CA3C(3)   465         bpl   start1
CA41:06 41        (5)   466         asl   topbits       ;Get an MSB
CA43:B0 02    CA47(3)   467         bcs   gob1          ;If MSB set, leave start bit
CA45:49 80        (2)   468         eor   #$80          ;MSB clear- flip start bit
CA47:91 54        (6)   469 gob1    sta   (buffer),y    ;Squirrel it away
CA49:C8           (2)   470         iny                 ;Next spot
```

```
CA4A:C4 4C       (3)  471         cpy   oddbytes      ;Are we done?
CA4C:90 EE   CA3C(3)  472         blt   start1        ;If more, branch
CA4E:                 473 *
CA4E:        CA4E     474 start2 equ   *
CA4E:        0001     475         do    IIc^ROM
CA4E:4C 72 CC    (3)  476         jmp   SlotDepRd
CA51:                 477         else
CA51:                 479         fin
CA51:                 480 *
CA51:        CA51     481 Send80 equ   *
CA51:A9 80       (2)  482         lda   #$80
CA53:        CA53     483 SendByte equ *
CA53:BC 8C C0    (4)  484         ldy   l6clr,x
CA56:10 FB   CA53(3)  485         bpl   SendByte
CA58:9D 8D C0    (5)  486         sta   l6set,x
CA5B:45 40       (3)  487         eor   checksum
CA5D:85 40       (3)  488         sta   checksum
CA5F:60          (6)  489         rts
CA60:                 490 *
CA60:                 491 *
CA60:                 492 *
CA60:                 493 *
CA60:        CA60     494 resetchain equ *
CA60:20 8A CA    (6)  495         jsr   ClrPhases
CA63:BD 81 C0    (4)  496         lda   reqset,x
CA66:BD 85 C0    (4)  497         lda   ca2set,x
CA69:A0 50       (2)  498         ldy   #80             ;Hard reset for 80 ms
CA6B:20 73 CA    (6)  499         jsr   YMSWait
CA6E:                 500 *
CA6E:20 8A CA    (6)  501         jsr   ClrPhases
CA71:                 502 *
CA71:A0 0A       (2)  503         ldy   #10             ;About 10 mS reset time!
CA73:                 504 *
CA73:        CA73     505 YMSWait equ  *
CA73:20 7A CA    (6)  506         jsr   OneMS
CA76:88          (2)  507         dey
CA77:D0 FA   CA73(3)  508         bne   YMSWait
CA79:60          (6)  509         rts
CA7A:                 510 *
CA7A:        CA7A     511 OneMS  equ   *
CA7A:A2 C8       (2)  512         ldx   #200
CA7C:CA          (2)  513 onems1 dex
CA7D:D0 FD   CA7C(3)  514         bne   onems1
CA7F:60          (6)  515         rts
CA80:                 516 *
CA80:                 517 *
CA80:        CA80     518 enablechain equ *
CA80:20 9A CA    (6)  519         jsr   SetXN0
CA83:BD 83 C0    (4)  520         lda   ca1set,x
CA86:BD 87 C0    (4)  521         lda   lstrbset,x
CA89:60          (6)  522         rts
CA8A:                 523 *
CA8A:                 524 *
CA8A:        CA8A     525 ClrPhases equ *
CA8A:20 9A CA    (6)  526         jsr   SetXN0
CA8D:BD 80 C0    (4)  527         lda   reqclr,x
CA90:BD 82 C0    (4)  528         lda   ca1clr,x
CA93:BD 84 C0    (4)  529         lda   ca2clr,x
```

```
CA96:BD 86 C0    (4)  530         lda   lstrbclr,x
CA99:60          (6)  531         rts
CA9A:                 532 *
CA9A:                 533 *
CA9A:         CA9A    534 SetXN0 equ   *
CA9A:         0001    535         do    IIc
CA9A:A2 60       (2)  536         ldx   #$60
CA9C:                 537         else
CA9C:                 544         fin
CA9C:                 545 *
CA9C:60          (6)  546         rts
CA9D:                 547 *
CA9D:                 548 * Shift tables for use when reading.  Each table should not
CA9D:                 549 *  straddle pages.
CA9D:                 550 *
CA9D:80 80 80 80      551 shift1 dfb   $80,$80,$80,$80,$80,$80,$80,$80
CAA5:00 00 00 00      552         dfb   0,0,0,0,0,0,0,0
CAAD:80 80 80 80      553 shift2 dfb   $80,$80,$80,$80,0,0,0,0
CAB5:80 80 80 80      554         dfb   $80,$80,$80,$80,0,0,0,0
CABD:80 80 00 00      555 shift3 dfb   $80,$80,0,0,$80,$80,0,0
CAC5:80 80 00 00      556         dfb   $80,$80,0,0,$80,$80,0,0
CACD:80 00 80 00      557 shift4 dfb   $80,0,$80,0,$80,0,$80,0
CAD5:80 00 80 00      558         dfb   $80,0,$80,0,$80,0,$80,0
CADD:                 559 *
CADD:                 560 *
```

```
CADD:                   562 *
CADD:           CADD    563 SendData equ *
CADD:A9 05        (2)   564         lda   #>RC2
CADF:A0 00        (2)   565         ldy   #<RC2
CAE1:20 00 CB     (6)   566         jsr   SendPile
CAE4:90 05   CAEB(3)    567         bcc   sdoubt
CAE6:A9 80        (2)   568         lda   #CommReset
CAE8:20 90 CF     (6)   569         jsr   AssignID
CAEB:           CAEB    570 sdoubt equ   *
CAEB:60           (6)   571         rts
CAEC:                   572 *
CAEC:                   573 *
CAEC:           CAEC    574 SendPack equ *
CAEC:20 00 CB     (6)   575         jsr   SendPile      ;Try to send a pack
CAEF:90 FA   CAEB(3)    576         bcc   sdoubt
CAF1:A9 80        (2)   577         lda   #CommReset    ;This is a communications failure
CAF3:20 90 CF     (6)   578         jsr   AssignID      ;Reset to try again
CAF6:                   579 *
CAF6:AD F8 06     (4)   580         lda   SvBcL         ;Get back the packetlength
CAF9:85 4D        (3)   581         sta   bytecountl
CAFB:AD 78 07     (4)   582         lda   SvBcH
CAFE:85 4E        (3)   583         sta   bytecounth
CB00:                   584 *
CB00:           CB00    585 SendPile equ *
CB00:A9 B8        (2)   586         lda   #>RC1         ;Retry count (big!)
CB02:A0 0B        (2)   587         ldy   #<RC1
CB04:                   588 *
CB04:           CB04    589 AltSendPile equ *
CB04:A6 58        (3)   590         ldx   slot
CB06:9D F3 04     (5)   591         sta   Retry,x
CB09:98           (2)   592         tya
CB0A:9D 73 05     (5)   593         sta   Retry2,x
CB0D:                   594 *
CB0D:                   595 * SendPack destroys the bytecount
CB0D:                   596 *
CB0D:           CB0D    597 spile1 equ   *
CB0D:A5 4D        (3)   598         lda   bytecountl
CB0F:8D F8 06     (4)   599         sta   SvBcL
CB12:A5 4E        (3)   600         lda   bytecounth
CB14:8D 78 07     (4)   601         sta   SvBcH
CB17:                   602 *
CB17:20 86 C8     (6)   603         jsr   SendOnePack   ;Send the packet
CB1A:                   604 *
CB1A:AD F8 06     (4)   605         lda   SvBcL
CB1D:85 4D        (3)   606         sta   bytecountl
CB1F:AD 78 07     (4)   607         lda   SvBcH
CB22:85 4E        (3)   608         sta   bytecounth
CB24:                   609 *
CB24:90 0C   CB32(3)    610         bcc   spilout
CB26:A6 58        (3)   611         ldx   slot
CB28:DE F3 04     (7)   612         dec   Retry,x
CB2B:D0 E0   CB0D(3)    613         bne   spile1
CB2D:DE 73 05     (7)   614         dec   Retry2,x
CB30:10 DB   CB0D(3)    615         bpl   spile1       ;If all fails, carry is set
CB32:60           (6)   616 spilout rts
CB33:                   617 *
CB33:           CB33    618 RecPack equ  *
CB33:A4 58        (3)   619         ldy   Slot
```

```
CB35:A9 05      (2)  620         lda   #>RC2
CB37:99 F3 04   (5)  621         sta   Retry,y
CB3A:       CB3A     622 rpk1    equ   *
CB3A:20 E6 C9   (6)  623         jsr   ReceivePack
CB3D:90 0F   CB4E(3) 624         bcc   rpout
CB3F:A0 01      (2)  625         ldy   #1
CB41:20 73 CA   (6)  626         jsr   YMSWait
CB44:20 C3 C9   (6)  627         jsr   dberror       ;Recycle handshake and set carry
CB47:A6 58      (3)  628         ldx   Slot
CB49:DE F3 04   (7)  629         dec   Retry,x
CB4C:D0 EC   CB3A(3) 630         bne   rpk1          ;Carry set still
CB4E:       CB4E     631 rpout   equ   *
CB4E:60         (6)  632         rts
CB4F:                633 *
CB4F:                634 *
```

```
CB4F:                  636 ************************************************************
CB4F:                  637 *                                                          *
CB4F:                  638 *  Divide7                Do DIV and MOD 7 and set auxptr  *
CB4F:                  639 *                                                          *
CB4F:                  640 *   This routine divides the bytecount by seven.  The      *
CB4F:                  641 *  quotient gives the number of groups of seven bytes to   *
CB4F:                  642 *  be sent, and the remainder gives the number of "odd"    *
CB4F:                  643 *  bytes.                                                  *
CB4F:                  644 *                                                          *
CB4F:                  645 *  Input:   bytecount1,h <- # of bytes to write            *
CB4F:                  646 *           buffer       <- pointer to data                *
CB4F:                  647 *  Output:  auxptr       <- pointer to speed up csumming   *
CB4F:                  648 *           oddbytes     <- bytecount MOD 7                *
CB4F:                  649 *           grp7ctr      <- bytecount DIV 7                *
CB4F:                  650 *                                                          *
CB4F:                  651 ************************************************************
CB4F:                  652 *
CB4F:00 24 49          653 pdiv7tab dfb 0,36,73
CB52:00 04 01          654 pmod7tab dfb 0,4,1
CB55:00 01 02 04       655 div7tab dfb  0,1,2,4,9,18
CB5B:00 01 02 04       656 mod7tab dfb  0,1,2,4,1,2
CB61:                  657 *
CB61:00 7F FF          658 auxptrinc dfb 0,$7F,$FF
CB64:                  659 *
CB64:         CB64     660 WritePrep equ *
CB64:         CB64     661 Divide7 equ  *
CB64:                  662 *
CB64:                  663 * Set up auxptr <- buffer+$80   if $0FF < bytecount < $200
CB64:                  664 *     or auxptr <- buffer+$100  if $1FF < bytecount
CB64:                  665 *
CB64:A6 4E       (3)   666         ldx   bytecounth     ;0, 1 or 2
CB66:F0 13   CB7B(3)   667         beq   noauxptr       ;Auxptr used only for full pages
CB68:                  668 *
CB68:A5 55       (3)   669         lda   buffer+1
CB6A:85 57       (3)   670         sta   auxptr+1       ;Copy over hi order part
CB6C:                  671 *
CB6C:A9 80       (2)   672         lda   #$80           ;Anticipate smaller bytecount
CB6E:E0 01       (2)   673         cpx   #1             ;Check bytecount
CB70:F0 04   CB76(3)   674         beq   sap1           ;=> $0FF < bytecount < $200
CB72:                  675 *
CB72:E6 57       (5)   676         inc   auxptr+1       ;Add $100 to bytecount instead
CB74:A9 00       (2)   677         lda   #0             ;Make sure lo order unaltered
CB76:18          (2)   678 sap1    clc
CB77:65 54       (3)   679         adc   buffer
CB79:85 56       (3)   680         sta   auxptr
CB7B:                  681 *
CB7B:                  682 * Now look up the first order guess for DIV and MOD.  X still has
CB7B:                  683 *  bytecount DIV 256.
CB7B:                  684 *
CB7B:         CB7B     685 noauxptr equ *
CB7B:BD 4F CB    (4)   686         lda   pdiv7tab,x
CB7E:85 4B       (3)   687         sta   grp7ctr
CB80:BD 52 CB    (4)   688         lda   pmod7tab,x
CB83:85 4C       (3)   689         sta   oddbytes
CB85:                  690 *
CB85:                  691 * Now add in the mods and divs for each of the five hi order
CB85:                  692 *  bits in the lo order bytecount, correcting each time MOD becomes
CB85:                  693 *  bigger than 6.
```

```
CB85:                   694 *
CB85:A2 05        (2)   695         ldx   #5              ;Do for five bits
CB87:A5 4D        (3)   696         lda   bytecount1
CB89:85 59        (3)   697         sta   temp            ;Store lo order for shifting
CB8B:29 07        (2)   698         and   #%00000111      ;Save lo three for later
CB8D:A8           (2)   699         tay
CB8E:                   700 *
CB8E:         CB8E      701 divide3 equ   *
CB8E:06 59        (5)   702         asl   temp            ;C <- next from bytecount1
CB90:90 15    CBA7(3)   703         bcc   divide2         ;If clear, no effect on DIV,MOD
CB92:BD 5B CB     (4)   704         lda   mod7tab,x       ;Get MOD7 for 2^n
CB95:         CB95      705 divide4 equ   *
CB95:18           (2)   706         clc
CB96:65 4C        (3)   707         adc   oddbytes        ;Got new MOD value
CB98:C9 07        (2)   708         cmp   #7              ;Is it too big?
CB9A:90 02    CB9E(3)   709         blt   divide1         ;=> NO leave MOD - 0->C
CB9C:E9 07        (2)   710         sbc   #7              ;Bring MOD under 7 - C still set
CB9E:         CB9E      711 divide1 equ   *
CB9E:85 4C        (3)   712         sta   oddbytes
CBA0:BD 55 CB     (4)   713         lda   div7tab,x       ;Get DIV for this 2^n
CBA3:65 4B        (3)   714         adc   grp7ctr         ;Add to DIV along with correction (C)
CBA5:85 4B        (3)   715         sta   grp7ctr         ;Update the DIV
CBA7:         CBA7      716 divide2 equ   *
CBA7:CA           (2)   717         dex                   ;One less bit to deal with
CBA8:30 06    CBB0(3)   718         bmi   divide5         ;Escape after 6 times through loop
CBAA:D0 E2    CB8E(3)   719         bne   divide3         ;Take brnch 1st 5 loops
CBAC:                   720 *
CBAC:98           (2)   721         tya                   ;Get back the last three bits
CBAD:4C 95 CB     (3)   722         jmp   divide4         ;Sixth pass add in remains
CBB0:                   723 *
CBB0:         CBB0      724 divide5 equ   *
CBB0:                   725 *
CBB0:                   726 *
```

```
CBB0:                  728 ************************************************************
CBB0:                  729 *                                                          *
CBB0:                  730 *  PreCheck                    Does the checksumming prepass *
CBB0:                  731 *                                                          *
CBB0:                  732 *    Input:   bytecount   <- bytes in buffer               *
CBB0:                  733 *             buffer      <- pointer to data to send       *
CBB0:                  734 *             auxptr      <- extra pointer to speed process*
CBB0:                  735 *    Output:  checksum    <- 8 bit XOR of data to be sent  *
CBB0:                  736 *                                                          *
CBB0:                  737 ************************************************************
CBB0:                  738 *
CBB0:          CBB0    739 PreCheck equ *
CBB0:                  740 *
CBB0:                  741 * Checksum any full pages
CBB0:                  742 *
CBB0:A5 55        (3)  743         lda   buffer+1
CBB2:48           (3)  744         pha                   ;Preserve buffer pointer
CBB3:A9 00        (2)  745         lda   #0
CBB5:A6 4E        (3)  746         ldx   bytecounth
CBB7:F0 16   CBCF(3)   747         beq   lastpass        ;If no complete pages, skip this
CBB9:          CBB9    748 xor2    equ   *
CBB9:BC 61 CB     (4)  749         ldy   auxptrinc,x     ;Get number of bytes each ptr
CBBC:          CBBC    750 xor1    equ   *
CBBC:51 54        (5)  751         eor   (buffer),y
CBBE:51 56        (5)  752         eor   (auxptr),y
CBC0:88           (2)  753         dey                   ;One less
CBC1:D0 F9   CBBC(3)   754         bne   xor1
CBC3:51 54        (5)  755         eor   (buffer),y
CBC5:51 56        (5)  756         eor   (auxptr),y      ;Have to deal with 0 case
CBC7:                  757 *
CBC7:                  758 * Now move the buffer up for next section
CBC7:                  759 *
CBC7:E0 01        (2)  760         cpx   #1
CBC9:F0 02   CBCD(3)   761         beq   xor5            ;If 256 and up bytes, bump x1
CBCB:E6 55        (5)  762         inc   buffer+1        ; otherwise x2
CBCD:E6 55        (5)  763 xor5    inc   buffer+1
CBCF:                  764 *
CBCF:          CBCF    765 lastpass equ *
CBCF:                  766 *
CBCF:                  767 * Do the remaining less than a page with a single pointer
CBCF:                  768 *
CBCF:A4 4D        (3)  769         ldy   bytecount
CBD1:F0 09   CBDC(3)   770         beq   xor4
CBD3:51 54        (5)  771         eor   (buffer),y      ;Compensate for nth byte
CBD5:51 54        (5)  772 xor3    eor   (buffer),y
CBD7:88           (2)  773         dey
CBD8:D0 FB   CBD5(3)   774         bne   xor3
CBDA:51 54        (5)  775         eor   (buffer),y      ;Last damn (0th) byte
CBDC:                  776 *
CBDC:                  777 * Store result away.  Retrieve old buffer value.
CBDC:                  778 *
CBDC:          CBDC    779 xor4    equ   *
CBDC:85 40        (3)  780         sta   checksum
CBDE:68           (4)  781         pla
CBDF:85 55        (3)  782         sta   buffer+1
CBE1:                  783 *
CBE1:                  784 *
```

```
CBE1:                  786 ********************************************************
CBE1:                  787 *                                                      *
CBE1:                  788 * DetTopBits              Get topbits for odd bytes   *
CBE1:                  789 *                                                      *
CBE1:                  790 *   Also sets buffer2 pointer to pointer at groups of *
CBE1:                  791 *  seven bytes.                                        *
CBE1:                  792 *                                                      *
CBE1:                  793 *  Input:   oddbytes <- # of "odd" bytes               *
CBE1:                  794 *           buffer   <- pointer to data                *
CBE1:                  795 *  Output:  tbodd    <- topbits for odd bytes          *
CBE1:                  796 *           buffer2  <- buffer+oddbytes                 *
CBE1:                  797 *                                                      *
CBE1:                  798 ********************************************************
CBE1:                  799 *
CBE1:          CBE1    800 DetTopBits equ *
CBE1:                  801 *
CBE1:A4 4C       (3)   802         ldy   oddbytes
CBE3:88          (2)   803         dey
CBE4:A9 00       (2)   804         lda   #0
CBE6:85 59       (3)   805         sta   tbodd
CBE8:                  806 *
CBE8:B1 54       (5)   807 gtbob   lda   (buffer),y
CBEA:0A          (2)   808         asl   a
CBEB:66 59       (5)   809         ror   tbodd
CBED:88          (2)   810         dey
CBEE:10 F8   CBE8(3)   811         bpl   gtbob
CBF0:38          (2)   812         sec
CBF1:66 59       (5)   813         ror   tbodd
CBF3:                  814 *
CBF3:A5 4C       (3)   815         lda   oddbytes
CBF5:18          (2)   816         clc
CBF6:65 54       (3)   817         adc   buffer
CBF8:85 56       (3)   818         sta   buffer2
CBFA:A5 55       (3)   819         lda   buffer+1
CBFC:69 00       (2)   820         adc   #0
CBFE:85 57       (3)   821         sta   buffer2+1
CC00:                  822 *
CC00:                  823 *
```

```
CC00:                  825 ************************************************************
CC00:                  826 *                                                          *
CC00:                  827 *  Sun                     Set up next buffer and topbits  *
CC00:                  828 *                                                          *
CC00:                  829 *   Primes the pipe for the group of seven bytes routine *
CC00:                  830 *  setting the topbits byte and the "next" buffer.         *
CC00:                  831 *  The routine also advances the buffer pointer by 7 to    *
CC00:                  832 *  prepare for the groups of seven transfer.               *
CC00:                  833 *                                                          *
CC00:                  834 *    Input:   buffer2   <- points to groups of 7 data      *
CC00:                  835 *    Output:  next1,7   <- first 7 bytes in buffer         *
CC00:                  836 *             topbits   <- MSBs of first 7 bytes           *
CC00:                  837 *                                                          *
CC00:                  838 ************************************************************
CC00:                  839 *
CC00:          CC00    840 Sun    equ   *
CC00:                  841 *
CC00:                  842 * Copy first seven bytes into the pipeline
CC00:                  843 *
CC00:A0 06       (2)   844        ldy   #6
CC02:38          (2)   845 sun2   sec
CC03:B1 56       (5)   846        lda   (buffer2),y
CC05:99 4D 00    (5)   847        sta   next,y
CC08:30 01   CC0B(3)   848        bmi   sun1
CC0A:18          (2)   849        clc
CC0B:66 41       (5)   850 sun1   ror   topbits
CC0D:88          (2)   851        dey
CC0E:10 F2   CC02(3)   852        bpl   sun2
CC10:38          (2)   853        sec
CC11:66 41       (5)   854        ror   topbits
CC13:                  855 *
CC13:                  856 * Advance the pointer
CC13:                  857 *
CC13:A5 56       (3)   858        lda   buffer2
CC15:18          (2)   859        clc
CC16:69 07       (2)   860        adc   #7
CC18:85 56       (3)   861        sta   buffer2
CC1A:90 02   CC1E(3)   862        bcc   sun3
CC1C:E6 57       (5)   863        inc   buffer2+1
CC1E:          CC1E    864 sun3   equ   *
CC1E:60          (6)   865        rts
CC1F:                  866 *
CC1F:                  867 *
```

```
CC1F:                   869 *
CC1F:                   870 * X is slot*16, Y is the desired mode
CC1F:                   871 *
CC1F:                   872 * Set up the IWM mode register.  Extreme care should be taken
CC1F:                   873 *  here. Setting the mode byte with indexed stores causes a false
CC1F:                   874 *  byte to be written a cycle before the real value is written.
CC1F:                   875 *  This false value, if it enables the timer, causes the IWM Rev A
                            to
CC1F:                   876 *  pop the motor on, inhibiting the setting of the mode until the
CC1F:                   877 *  motor times out! We avoid this by setting the mode byte only
                            when
CC1F:                   878 *  it is not what we want, and if it's not we stay here until we
CC1F:                   879 *  see that it is what we want.
CC1F:                   880 *
CC1F:         CC1F      881 SetIWMode equ *
CC1F:BD 88 C0    (4)    882         lda   monclr,x      ;Motor must be off
CC22:BD 8D C0    (4)    883         lda   l6set,x       ;Set up to access mode register
CC25:4C 2C CC    (3)    884         jmp   careful       ;Don't mess unless we gotta
CC28:98          (2)    885 biz     tya
CC29:9D 8F C0    (5)    886         sta   l7set,x       ;Try storing the mode value
CC2C:         CC2C      887 careful equ   *
CC2C:98          (2)    888         tya                 ;Get back the target value
CC2D:5D 8E C0    (4)    889         eor   l7clr,x       ;Compare with observed value
CC30:29 1F       (2)    890         and   #$1F          ;Can only read low 5 bits
CC32:D0 F4    CC28(3)   891         bne   biz           ;If not right, back to try again
CC34:60          (6)    892         rts
CC35:                   893 *
CC35:                   894 *
CC35:         0001      895         Do    IIc
CC35:         CC35      896 WaitIWMOff equ *
CC35:                   897 *
CC35:                   898 * Make sure you're in read mode and wait 'til Disk // motor is off
CC35:                   899 *
CC35:20 9A CA    (6)    900         jsr   SetXN0        ;Set X
CC38:BD 8E C0    (4)    901         lda   l7clr,x
CC3B:BD 8D C0    (4)    902         lda   l6set,x
CC3E:         CC3E      903 wiwm1   equ   *
CC3E:BD 8E C0    (4)    904         lda   l7clr,x
CC41:29 20       (2)    905         and   #%00100000
CC43:D0 F9    CC3E(3)   906         bne   wiwm1
CC45:BD 8C C0    (4)    907         lda   l6clr,x
CC48:                   908 *
CC48:                   909 * Wait an additional 700 microseconds to allow 12V on Disk // to
                            decay
CC48:                   910 *
CC48:5A          (3)    911         phy
CC49:A0 8C       (2)    912         ldy   #140
CC4B:88          (2)    913 wiwm2   dey
CC4C:D0 FD    CC4B(3)   914         bne   wiwm2
CC4E:7A          (4)    915         ply
CC4F:                   916 *
CC4F:60          (6)    917         rts
CC50:                   918         fin
CC50:                   919 *
CC50:                   920 *
CC50:                   921 * This takes grp7ctr and oddbytes and calculates
                            7*grp7ctr+oddbytes.
CC50:                   922 *  The results are in Y(hi) and A(lo).  This is the number of bytes
CC50:                   923 *  that were received in the last ReceivePack.
CC50:                   924 *
CC50:         CC50      925 Rcvcount equ *
CC50:A5 4B       (3)    926         lda   grp7ctr
```

```
CC52:A8          (2)  927         tay
CC53:A2 00       (2)  928         ldx   #0
CC55:86 4B       (3)  929         stx   grp7ctr
CC57:A2 03       (2)  930         ldx   #3
CC59:0A          (2)  931 times7  asl   a
CC5A:26 4B       (5)  932         rol   grp7ctr
CC5C:CA          (2)  933         dex
CC5D:D0 FA    CC59(3) 934         bne   times7
CC5F:18          (2)  935         clc
CC60:65 4C       (3)  936         adc   oddbytes
CC62:90 02    CC66(3) 937         bcc   t71
CC64:E6 4B       (5)  938         inc   grp7ctr
CC66:84 4C       (3)  939 t71     sty   oddbytes
CC68:38          (2)  940         sec
CC69:E5 4C       (3)  941         sbc   oddbytes
CC6B:B0 02    CC6F(3) 942         bcs   t72
CC6D:C6 4B       (5)  943         dec   grp7ctr
CC6F:A4 4B       (3)  944 T72     ldy   grp7ctr
CC71:60          (6)  945         rts
CC72:                 946 *
CC72:                 947 *
CC72:                 136 *
CC72:         0001    137         do    IIc^ROM
CC72:                 138         include pc.read
CC72:         CC72      1 SlotDepRd equ *
CC72:         CC72      2 start25 equ  *
CC72:A0 00       (2)    3         ldy   #0
CC74:A5 4B       (3)    4         lda   grp7ctr
CC76:48          (3)    5         pha                    ;Save groups of seven counter
CC77:D0 03    CC7C(3)   6         bne   start35
CC79:4C 09 CD    (3)    7         jmp   done5            ;Go get the checksum
CC7C:                   8 *
CC7C:                   9 * Okay, get the groups of seven
CC7C:                  10 *  Start by getting the topbits for this group of seven
CC7C:                  11 *
CC7C:         CC7C     12 start35 equ  *
CC7C:AD EC C0    (4)   13         lda   l6clr+TheOff     ;Get topbits
CC7F:10 FB    CC7C(3)  14         bpl   start35
CC81:85 59       (3)   15         sta   temp             ;Just a second
CC83:                  16 *
CC83:                  17 * Split up the seven bits into two indices for topbit tables
CC83:                  18 *
CC83:4A          (2)   19         lsr   a                ;0   1  d1  d2  d3  d4  d5  d6
CC84:4A          (2)   20         lsr   a                ;0   0   1  d1  d2  d3  d4  d5
CC85:4A          (2)   21         lsr   a                ;0   0   0   1  d1  d2  d3  d4
CC86:29 0F       (2)   22         and   #%00001111       ;0   0   0   0  d1  d2  d3  d4
CC88:AA          (2)   23         tax                    ;First index into the tables
CC89:A5 59       (3)   24         lda   temp             ;1  d1  d2  d3  d4  d5  d6  d7
CC8B:29 07       (2)   25         and   #%00000111       ;0   0   0   0   0  d5  d6  d7
CC8D:85 59       (3)   26         sta   temp             ;Keep for last three bytes
CC8F:                  27 *
CC8F:                  28 * Now read the first byte, reunite its msb, store it, and checksum
                          it.
CC8F:                  29 *
CC8F:AD EC C0    (4)   30         lda   l6clr+TheOff
CC92:10 FB    CC8F(3)  31         bpl   *-3              ;Back 1 instruction
CC94:5D 9D CA    (4)   32         eor   shift1,x         ;Recombine the MSB with data
CC97:91 56       (6)   33         sta   (buffer2),y      ;Store it away
CC99:45 40       (3)   34         eor   checksum         ;Add it to the checksum
```

```
CC9B:85 40      (3)   35          sta   checksum
CC9D:C8         (2)   36          iny
CC9E:                 37 *
CC9E:                 38 * Now, the second Y turn over occurs at this point in the loop.
                        Update
CC9E:                 39 *  the buffer pointer if it occurred.
CC9E:                 40 *
CC9E:D0 02   CCA2(3)  41          bne   *+4
CCA0:E6 57      (5)   42          inc   buffer2+1
CCA2:                 43 *
CCA2:                 44 * Now the second byte
CCA2:                 45 *
CCA2:AD EC C0   (4)   46          lda   16clr+TheOff
CCA5:10 FB   CCA2(3)  47          bpl   *-3           ;Back 1 instruction
CCA7:5D AD CA   (4)   48          eor   shift2,x      ;Recombine the MSB with data
CCAA:91 56      (6)   49          sta   (buffer2),y   ;Store it away
CCAC:45 40      (3)   50          eor   checksum      ;Add it to the checksum
CCAE:85 40      (3)   51          sta   checksum
CCB0:C8         (2)   52          iny
CCB1:                 53 *
CCB1:                 54 * Now the third byte
CCB1:                 55 *
CCB1:AD EC C0   (4)   56          lda   16clr+TheOff
CCB4:10 FB   CCB1(3)  57          bpl   *-3           ;Back 1 instruction
CCB6:5D BD CA   (4)   58          eor   shift3,x      ;Recombine the MSB with data
CCB9:91 56      (6)   59          sta   (buffer2),y   ;Store it away
CCBB:45 40      (3)   60          eor   checksum      ;Add it to the checksum
CCBD:85 40      (3)   61          sta   checksum
CCBF:C8         (2)   62          iny
CCC0:                 63 *
CCC0:                 64 * Now the fourth byte
CCC0:                 65 *
CCC0:AD EC C0   (4)   66          lda   16clr+TheOff
CCC3:10 FB   CCC0(3)  67          bpl   *-3           ;Back 1 instruction
CCC5:5D CD CA   (4)   68          eor   shift4,x      ;Recombine the MSB with data
CCC8:91 56      (6)   69          sta   (buffer2),y   ;Store it away
CCCA:45 40      (3)   70          eor   checksum      ;Add it to the checksum
CCCC:85 40      (3)   71          sta   checksum
CCCE:C8         (2)   72          iny
CCCF:                 73 *
CCCF:                 74 * The first Y turn over occurs at this point in the loop.  Update
CCCF:                 75 *  the buffer pointer if it occurred.
CCCF:                 76 *
CCCF:D0 02   CCD3(3)  77          bne   *+4
CCD1:E6 57      (5)   78          inc   buffer2+1
CCD3:                 79 *
CCD3:A6 59      (3)   80          ldx   temp          ;Now we need the other index
CCD5:                 81 *
CCD5:                 82 * Now the fifth byte
CCD5:                 83 *
CCD5:AD EC C0   (4)   84          lda   16clr+TheOff
CCD8:10 FB   CCD5(3)  85          bpl   *-3           ;Back 1 instruction
CCDA:5D AD CA   (4)   86          eor   shift2,x      ;Recombine the MSB with data
CCDD:91 56      (6)   87          sta   (buffer2),y   ;Store it away
CCDF:45 40      (3)   88          eor   checksum      ;Add it to the checksum
CCE1:85 40      (3)   89          sta   checksum
CCE3:C8         (2)   90          iny
CCE4:                 91 *
CCE4:                 92 * Now the sixth byte
```

```
CCE4:                    93 *
CCE4:AD EC C0     (4)    94          lda   l6clr+TheOff
CCE7:10 FB   CCE4(3)     95          bpl   *-3              ;Back 1 instruction
CCE9:5D BD CA     (4)    96          eor   shift3,x         ;Recombine the MSB with data
CCEC:91 56        (6)    97          sta   (buffer2),y      ;Store it away
CCEE:45 40        (3)    98          eor   checksum         ;Add it to the checksum
CCF0:85 40        (3)    99          sta   checksum
CCF2:C8           (2)   100          iny
CCF3:                   101 *
CCF3:                   102 * And, finally, the seventh byte
CCF3:                   103 *
CCF3:AD EC C0     (4)   104          lda   l6clr+TheOff
CCF6:10 FB   CCF3(3)    105          bpl   *-3              ;Back 1 instruction
CCF8:5D CD CA     (4)   106          eor   shift4,x         ;Recombine the MSB with data
CCFB:91 56        (6)   107          sta   (buffer2),y      ;Store it away
CCFD:45 40        (3)   108          eor   checksum         ;Add it to the checksum
CCFF:85 40        (3)   109          sta   checksum
CD01:C8           (2)   110          iny
CD02:                   111 *
CD02:                   112 * Now see if this is the last group of seven to receive
CD02:                   113 *
CD02:C6 4B        (5)   114          dec   grp7ctr
CD04:F0 03   CD09(3)    115          beq   done5            ;Go to get the checksum etc
CD06:4C 7C CC     (3)   116          jmp   start35          ;Another topbits ...
CD09:                   117 *
CD09:                   118 * Get and reconstruct the checksum
CD09:                   119 *
CD09:         CD09      120 done5    equ   *
CD09:AD EC C0     (4)   121          lda   l6clr+TheOff
CD0C:10 FB   CD09(3)    122          bpl   *-3
CD0E:85 59        (3)   123          sta   temp             ;1 c6 1 c4 1 c2 1 c0
CD10:                   124 *
CD10:68           (4)   125          pla                    ;Restore groups of 7 counter
CD11:85 4B        (3)   126          sta   grp7ctr
CD13:AD EC C0     (4)   127          lda   l6clr+TheOff     ;1 c7 1 c5 1 c3 1 c1
CD16:10 FB   CD13(3)    128          bpl   *-3
CD18:38           (2)   129          sec
CD19:2A           (2)   130          rol   a                ;c7 1 c5 1 c3 1 c1 1
CD1A:25 59        (3)   131          and   temp             ;c7 c6 c5 c4 c3 c2 c1 c0
CD1C:45 40        (3)   132          eor   checksum         ;When we're done, should be zero
CD1E:                   133 *
CD1E:                   134 * Get the packet end mark. Is it correct?
CD1E:                   135 *
CD1E:AC EC C0     (4)   136 rdha5    ldy   l6clr+TheOff     ;Preserve A
CD21:10 FB   CD1E(3)    137          bpl   rdha5
CD23:                   138 *
CD23:C0 C8        (2)   139          cpy   #packetend
CD25:D0 1C   CD43(3)    140          bne   npenderr5
CD27:                   141 *
CD27:                   142 * Didn't have time before to checksum oddbytes.  Do it now
CD27:                   143 *  A still has the partial checksum
CD27:                   144 *
CD27:A6 4C        (3)   145          ldx   oddbytes
CD29:F0 08   CD33(3)    146          beq   icbt15
CD2B:A0 00        (2)   147          ldy   #0
CD2D:51 54        (5)   148 icbt5    eor   (buffer),y
CD2F:C8           (2)   149          iny
CD30:CA           (2)   150          dex
```

```
CD31:D0 FA   CD2D(3)  151         bne   icbt5
CD33:                 152 *
CD33:                 153 * Okay, checksum oughta be zero.  If not, checksum error.
CD33:                 154 *
CD33:        CD33     155 icbt15 equ   *
CD33:AA         (2)   156         tax
CD34:D0 11   CD47(3)  157         bne   cserror5
CD36:                 158 *
CD36:                 159 * Wait for /BSY to go low
CD36:                 160 *
CD36:        CD36     161 lstbsywait5 equ *
CD36:AD ED C0   (4)   162         lda   l6set+TheOff
CD39:AD EE C0   (4)   163 rdh45   lda   l7clr+TheOff
CD3C:30 FB   CD39(3)  164         bmi   rdh45
CD3E:                 165 *
CD3E:                 166 * Got the bytes, now acknowledge their reciept
CD3E:                 167 *
CD3E:AD E0 C0   (4)   168         lda   reqclr+TheOff ;Lower REQ
CD41:                 169 *
CD41:18         (2)   170         clc
CD42:60         (6)   171         rts
CD43:                 172 *
CD43:A9 20      (2)   173 npenderr5 lda #nopackend
CD45:D0 02   CD49(3)  174         bne   gserror5
CD47:A9 10      (2)   175 cserror5 lda #csumerr
CD49:38         (2)   176 gserror5 sec
CD4A:60         (6)   177         rts
CD4B:                 178 *
CD4B:                 139         fin
CD4B:                 140 *
CD4B:                 141         include pc.main
```

```
CD4B:                    2 *
CD4B:                    3 *
CD4B:          CD4B      4 Entry  equ   *
CD4B:90 03    CD50(3)    5        bcc   bentry        ;If non-boot, skip jump to boot
CD4D:4C 23 C5     (3)    6        jmp   bootcode
CD50:                    7 *
CD50:                    8 * X is still set to slot number.
CD50:                    9 *
CD50:          CD50     10 bentry equ   *
CD50:                   11 *
CD50:          0001     12        do    IIc*ROM
CD50:A9 40        (2)   13        lda   #%01000000
CD52:1C 78 04     (6)   14        trb   ProFlag+5     ;ProFlag is fixed in //c
CD55:                   15 *
CD55:          CD55     16 atentry equ  *
CD55:                   17        fin
CD55:                   18 *
CD55:D8           (2)   19        cld                 ;Don't want decimal mode!!
CD56:8A           (2)   20        txa
CD57:A8           (2)   21        tay                 ;Really want it in Y... no ROR ABS,Y!
CD58:                   22 *
CD58:                   23 * If this is a PC call, then get the address of the parm table
CD58:                   24 *
CD58:B9 73 04     (4)   25        lda   ProFlag,y
CD5B:30 11    CD6E(3)   26        bmi   noplay
CD5D:                   27 *
CD5D:68           (4)   28        pla                 ;Get lo order
CD5E:99 F3 05     (5)   29        sta   SHTempX,y     ;Keep lo parm address-1
CD61:18           (2)   30        clc
CD62:69 03        (2)   31        adc   #3
CD64:AA           (2)   32        tax                 ;Lo order new return address
CD65:68           (4)   33        pla                 ;Get hi order address
CD66:99 73 06     (5)   34        sta   SHTempY,y     ;Keep hi parm addr-1
CD69:69 00        (2)   35        adc   #0
CD6B:48           (3)   36        pha                 ;Push back new return address hi
CD6C:8A           (2)   37        txa
CD6D:48           (3)   38        pha                 ;Push new return address lo
CD6E:                   39 *
CD6E:          CD6E     40 noplay equ   *
CD6E:                   41 *
CD6E:                   42 * On the //c, it is important to have the Disk // enable lines
CD6E:                   43 *  off for as long as possible before using the IWM (phases, /WRREQ
CD6E:                   44 *  lines).  Wait here 'til the Disk // motors are off.
CD6E:                   45 *
CD6E:          0001     46        do    IIc
CD6E:20 35 CC     (6)   47        jsr   WaitIWMOff    ;Must preserve Y!!
CD71:                   48        fin
CD71:                   49 *
CD71:                   50 * We can't really tolerate interrupts in most of the code, so
                           disable
CD71:                   51 *
CD71:08           (3)   52        php                 ;Save interrupt status
CD72:78           (2)   53        sei                 ;No interrupts please
CD73:                   54 *
CD73:                   55 * Preserve the zero page work area
CD73:                   56 *
CD73:A2 1B        (2)   57        ldx   #ZPSize-1
CD75:B5 40        (4)   58 pzp    lda   ZeroPage,x
CD77:48           (3)   59        pha
```

```
CD78:CA          (2)   60          dex
CD79:10 FA   CD75(3)   61          bpl   pzp
CD7B:                  62 *
CD7B:                  63 * Okay, we're safe... now it's all right to store in zero page
CD7B:                  64 *
CD7B:84 58       (3)   65          sty   Slot
CD7D:                  66 *
CD7D:        0001      67          ifeq  IIc^ROM
CD7D:                  80          fin
CD7D:                  81 *
CD7D:                  82 * Now map any ProDOS unit references to our sequential ones.
CD7D:                  83 *  The method is bizzare and magicians never reveal their secrets.
CD7D:                  84 *
CD7D:        CD7D      85 allset equ   *
CD7D:A5 43       (3)   86          lda   CMDUnit       ;76543210 7&6 specify unit
CD7F:2A          (2)   87          rol   a             ;6543210X C<-7
CD80:08          (3)   88          php                 ;Save drive num
CD81:2A          (2)   89          rol   a             ;543210X7 C<-6
CD82:2A          (2)   90          rol   a             ;43210X76 (6 is grp of 2)
CD83:28          (4)   91          plp                 ;C<-7
CD84:2A          (2)   92          rol   a             ;3210X767
CD85:29 03       (2)   93          and   #%00000011    ;ProDOS only installs up to 4
CD87:49 02       (2)   94          eor   #%00000010    ;000000/67; 6 was /grpoftwo
CD89:C0 04       (2)   95          cpy   #4            ;If in slot 1,2,or3 reverse grps of two
CD8B:B0 02   CD8F(3)   96          bge   allset1
CD8D:49 02       (2)   97          eor   #%00000010
CD8F:AA          (2)   98 allset1 tax
CD90:E8          (2)   99          inx
CD91:86 43       (3)  100          stx   CMDUnit       ;You got it
CD93:                 101 *
CD93:                 102 * Now if this is through the MLI xface, gotta copy stuff into the
CD93:                 103 *  send buffer from the parameter list.
CD93:                 104 *
CD93:B9 73 04    (4)  105          lda   ProFlag,y
CD96:10 03   CD9B(3)  106          bpl   darnit
CD98:4C 3F CE    (3)  107          jmp   skipcopy
CD9B:                 108 *
CD9B:                 109 * Get the address of the in-line parameter table
CD9B:                 110 *
CD9B:        CD9B     111 darnit equ   *
CD9B:B9 F3 05    (4)  112          lda   SHTempX,y     ;Get back the low part buff addr
CD9E:85 54       (3)  113          sta   buffer
CDA0:B9 73 06    (4)  114          lda   SHTempY,y     ; and the hi part
CDA3:85 55       (3)  115          sta   buffer+1
CDA5:                 116 *
CDA5:                 117 * Now pull out the command code, and the address of the parameters.
CDA5:                 118 *
CDA5:A0 01       (2)  119          ldy   #1            ;Stacked address is EA-1
CDA7:B1 54       (5)  120          lda   (buffer),y
CDA9:85 42       (3)  121          sta   cmdcode       ;Nice
CDAB:C8          (2)  122          iny
CDAC:B1 54       (5)  123          lda   (buffer),y    ;Get lo part of parmlist address
CDAE:AA          (2)  124          tax                 ;Save it
CDAF:C8          (2)  125          iny
CDB0:B1 54       (5)  126          lda   (buffer),y    ;Get hi part
CDB2:85 55       (3)  127          sta   buffer+1
CDB4:86 54       (3)  128          stx   buffer
CDB6:                 129 *
```

```
CDB6:                    130 * Now buffer points to parmlist
CDB6:                    131 *  Check command type, and pidgeonhole the parmlist length
CDB6:                    132 *
CDB6:A9 01        (2)    133          lda   #BadCmd
CDB8:A6 42        (3)    134          ldx   cmdcode
CDBA:E0 0A        (2)    135          cpx   #$A              ;Only valid codes are 0-9
CDBC:90 03    CDC1(3)    136          blt   noeh             ;=> at least he got that right
CDBE:4C 0F CF     (3)    137 Errorhitch jmp Error           ;Gee, maybe we should promote this guy...
CDC1:         CDC1       138 noeh     equ   *
CDC1:A0 00        (2)    139          ldy   #0               ;Set for indct compare
CDC3:B1 54        (5)    140          lda   (buffer),y       ;Get # of parms?
CDC5:85 5A        (3)    141          sta   Unit
CDC7:                    142 *
CDC7:                    143 * Now copy the bytes
CDC7:                    144 *
CDC7:         CDC7       145 okaycnt equ  *
CDC7:A0 08        (2)    146          ldy   #>cmdlength-1 ;Always copy the maximum
CDC9:         CDC9       147 copyloop equ *
CDC9:B1 54        (5)    148          lda   (buffer),y       ;Pull it out of their hat
CDCB:99 42 00     (5)    149          sta   cmdcode,y        ;Stuff it into mine
CDCE:88           (2)    150          dey
CDCF:D0 F8    CDC9(3)    151          bne   copyloop         ;Copy 'em all
CDD1:                    152 *
CDD1:                    153 * Okay.  The caller of the PC could be making one of three calls
CDD1:                    154 *  with a unit number of $00, Control, Init or Status.  Check for
CDD1:                    155 *  these and do what is appropriate.
CDD1:                    156 *
CDD1:A5 43        (3)    157          lda   CMDUnit
CDD3:D0 6A    CE3F(3)    158          bne   skipcopy         ;Never mind
CDD5:                    159 *
CDD5:                    160 * Check the parameter count for this call to unit#0
CDD5:                    161 *
CDD5:A6 42        (3)    162          ldx   CMDCode
CDD7:BD 86 CF     (4)    163          lda   parmctab,x       ;Get the length this command
CDDA:29 7F        (2)    164          and   #$7F             ;Force 0 -> MSB
CDDC:A8           (2)    165          tay                    ;Hang on
CDDD:A9 04        (2)    166          lda   #BadPCnt         ;Antic bad count
CDDF:C4 5A        (3)    167          cpy   Unit             ;User's pcount is currently here
CDE1:D0 DB    CDBE(3)    168          bne   ErrorHitch       ;What a baby!
CDE3:                    169 *
CDE3:                    170 * Now service one of the three commands
CDE3:                    171 *
CDE3:E0 05        (2)    172          cpx   #InitCMD
CDE5:D0 0A    CDF1(3)    173          bne   notinit          ;Not an Init call
CDE7:A9 00        (2)    174          lda   #PowerReset      ;Just like powerup or reset key(//c)
CDE9:20 90 CF     (6)    175          jsr   AssignID         ;Do a reset cycle
CDEC:A9 00        (2)    176 Aokay    lda   #0               ;No error allowed
CDEE:4C 31 CF     (3)    177          jmp   sa2
CDF1:                    178 *
CDF1:8A           (2)    179 notinit  txa                    ;Equiv to 'cmp #StatusCMD'
CDF2:D0 24    CE18(3)    180          bne   maybectrl
CDF4:                    181 *
CDF4:A9 21        (2)    182          lda   #BadCtl          ;Antic a non zero stat code
CDF6:A6 46        (3)    183          ldx   CMDSCode         ;Stat unit#0 can only be code=0
CDF8:D0 C4    CDBE(3)    184          bne   ErrorHitch
CDFA:                    185 *
CDFA:8A           (2)    186          txa                    ;Equiv to 'lda #0'
CDFB:A6 58        (3)    187          ldx   Slot
```

```
CDFD:A0 07      (2)  188        ldy   #7
CDFF:91 44      (6)  189 nin1   sta   (CmdBuffer1),y ;Clear some space
CE01:88         (2)  190        dey
CE02:D0 FB  CDFF(3)  191        bne   nin1
CE04:                192 *
CE04:BD F9 06   (4)  193        lda   NumDevices,x
CE07:91 44      (6)  194        sta   (CMDBuffer1),y ;Stick it where they want it
CE09:C8         (2)  195        iny
CE0A:                196 *
CE0A:       0001     197        do    IIc
CE0A:AD F9 04   (4)  198        lda   $4F9          ;//c Port 1 interrupt status
CE0D:                199        else
CE0D:                201        fin
CE0D:                202 *
CE0D:91 44      (6)  203        sta   (CMDBuffer1),y ;Store PC interrupt status
CE0F:                204 *
CE0F:A9 08      (2)  205        lda   #8
CE11:88         (2)  206        dey                  ;A,Y has 0008; # bytes status
CE12:20 F2 CF   (6)  207        jsr   squirrel
CE15:                208 *
CE15:4C EC CD   (3)  209        jmp   Aokay          ;Skip down (up) with no error
CE18:       CE18     210 maybectrl equ *
CE18:C9 04      (2)  211        cmp   #ControlCMD
CE1A:D0 0B  CE27(3)  212        bne   BUnit          ;Unit #0 was a bad one
CE1C:                213 *
CE1C:A6 46      (3)  214        ldx   CMDSCode       ;We allow two control calls for Unit#0
CE1E:F0 0B  CE2B(3)  215        beq   enabint        ;0 means enable interrupts
CE20:CA         (2)  216        dex
CE21:F0 14  CE37(3)  217        beq   disabint       ;1 means disable interrupts
CE23:A9 21      (2)  218        lda   #badctl
CE25:       CE25     219 ErrorHitch2 equ *
CE25:D0 97  CDBE(3)  220        bne   ErrorHitch     ;No other codes allowed
CE27:                221 *
CE27:       CE27     222 BUnit  equ   *
CE27:A9 11      (2)  223        lda   #badUnit       ;Only certain calls can have Unit#0
CE29:D0 93  CDBE(3)  224        bne   ErrorHitch     ;Branch always
CE2B:                225 *
CE2B:       0001     226        do    IIc
CE2B:       CE2B     227 enabint equ  *
CE2B:A9 C0      (2)  228        lda   #$C0
CE2D:8D F9 05   (4)  229        sta   $5F9
CE30:A9 0F      (2)  230        lda   #$0F
CE32:0C 9A C0   (6)  231        tsb   $C09A
CE35:D0 05  CE3C(3)  232        bne   aokayhitch
CE37:                233 *
CE37:       CE37     234 disabint equ *
CE37:A9 01      (2)  235        lda   #$01
CE39:1C 9A C0   (6)  236        trb   $C09A
CE3C:4C EC CD   (3)  237 aokayhitch jmp AOkay
CE3F:                238 *
CE3F:                239        else
CE3F:                244        fin
CE3F:                245 *
CE3F:                246 * Okay, everything's all groovy.  ProDOS re-enters here.
CE3F:                247 *  Check Unit number to be sure there is a corresponding device
CE3F:                248 *
CE3F:       CE3F     249 skipcopy equ *
CE3F:A9 28      (2)  250        lda   #NoDrive       ;Anticpate bad unit number
```

```
CE41:A4 58       (3)  251         ldy   slot
CE43:BE F9 06    (4)  252         ldx   NumDevices,y
CE46:E4 43       (3)  253         cpx   CMDUnit
CE48:90 DB   CE25(3)  254         blt   ErrorHitch2   ;Safe- If C clr then Z is clr
CE4A:                 255 *
CE4A:                 256 * Set buffer and bytecount in anticpation of the inevitable
CE4A:                 257 * SendPack.
CE4A:A9 09       (2)  258         lda   #>cmdlength
CE4C:85 4D       (3)  259         sta   bytecount1
CE4E:A9 00       (2)  260         lda   #<cmdlength
CE50:85 4E       (3)  261         sta   bytecounth
CE52:85 55       (3)  262         sta   buffer+1
CE54:A9 42       (2)  263         lda   #>cmdcode
CE56:85 54       (3)  264         sta   buffer
CE58:                 265 *
CE58:                 266 * If it's a PC call, omit the next two steps
CE58:                 267 *
CE58:A6 58       (3)  268         ldx   Slot
CE5A:BD 73 04    (4)  269         lda   ProFlag,x     ;Is it a call from ProDOS?
CE5D:10 13   CE72(3)  270         bpl   notstat       ;=> Statcode already set...
CE5F:                 271 *
CE5F:                 272 * Need to generate a parameter count for a ProDOS call
CE5F:                 273 *
CE5F:A6 42       (3)  274         ldx   CMDCode
CE61:BD 86 CF    (4)  275         lda   ParmCTab,x
CE64:29 7F       (2)  276         and   #$7F
CE66:85 5A       (3)  277         sta   Unit
CE68:                 278 *
CE68:                 279 * ProDOS always needs the highest blockno byte zeroed
CE68:                 280 *
CE68:A9 00       (2)  281         lda   #0
CE6A:85 48       (3)  282         sta   CMDBlockS
CE6C:                 283 *
CE6C:                 284 * If this is a ProDOS status call, set stat code to zero
CE6C:                 285 *
CE6C:A5 42       (3)  286         lda   CMDCode
CE6E:D0 02   CE72(3)  287         bne   notstat       ;=> Not status so forget it
CE70:                 288 *lda #SCDeviceStat ;A is already zero
CE70:85 46       (3)  289         sta   CMDSCode      ;Store in command table
CE72:                 290 *
CE72:                 291 * Okay, finally send over the damn command
CE72:                 292 *
CE72:        CE72     293 notstat equ  *
CE72:A5 5A       (3)  294         lda   Unit
CE74:A6 43       (3)  295         ldx   CmdPCount     ;Swap the Parmcount & unit#
CE76:86 5A       (3)  296         stx   Unit
CE78:85 43       (3)  297         sta   CMDPCount     ;Now they're correct
CE7A:                 298 *
CE7A:A9 80       (2)  299         lda   #cmdmark
CE7C:85 5B       (3)  300         sta   WPacketType
CE7E:                 301 *
CE7E:20 8A CA    (6)  302         jsr   ClrPhases     ;Bring all phases off
CE81:                 303 *
CE81:20 EC CA    (6)  304         jsr   SendPack
CE84:B0 46   CECC(3)  305         bcs   behitch       ;If not okay, skip to bus error
CE86:                 306 *
CE86:                 307 * Now copy over the buffer address for any data xfer.
CE86:                 308 *
```

```
CE86:A5 44      (3)  309        lda   CMDBuffer
CE88:85 54      (3)  310        sta   buffer
CE8A:A5 45      (3)  311        lda   CMDBuffer+1
CE8C:85 55      (3)  312        sta   buffer+1
CE8E:                313 *
CE8E:                314 * Now for some commands, we have to send over a packet of data, too.
CE8E:                315 *  See if this command is one of THOSE.
CE8E:                316 *
CE8E:A6 42      (3)  317        ldx   cmdcode
CE90:BD 86 CF   (4)  318        lda   parmctab,x
CE93:10 3B   CED0(3) 319        bpl   noxtrasend    ;Encoded in top bit
CE95:                320 *
CE95:                321 * The buffer address and bytecount depend on the call type.
CE95:                322 *
CE95:E0 04      (2)  323        cpx   #ControlCmd
CE97:D0 18   CEB1(3) 324        bne   NOControl
CE99:                325 *
CE99:                326 * In the case of control, bytecount:=(buffer) then buffer:=buffer+2
CE99:                327 *
CE99:A0 01      (2)  328        ldy   #1
CE9B:B1 54      (5)  329        lda   (buffer),y    ;Get Hi order bytecount
CE9D:AA         (2)  330        tax
CE9E:88         (2)  331        dey
CE9F:B1 54      (5)  332        lda   (buffer),y
CEA1:48         (3)  333        pha                 ;Keep for later
CEA2:18         (2)  334        clc
CEA3:A9 02      (2)  335        lda   #2
CEA5:65 54      (3)  336        adc   buffer
CEA7:85 54      (3)  337        sta   buffer
CEA9:68         (4)  338        pla                 ;Get back Lo order bytecount
CEAA:90 13   CEBF(3) 339        bcc   secondsend    ;Skip hi ord increment
CEAC:E6 55      (5)  340        inc   buffer+1
CEAE:4C BF CE   (3)  341        jmp   secondsend    ;Skip to store bytecount
CEB1:                342 *
CEB1:        CEB1    343 NOControl equ *
CEB1:E0 02      (2)  344        cpx   #WriteCMD     ;Check for a writeblock
CEB3:D0 06   CEBB(3) 345        bne   NOWBlock      ;Must be control or write
CEB5:                346 *
CEB5:                347 * In the case of WriteBlock, the length is 512 and the buffer
CEB5:                348 *  address is at buffer in the command table
CEB5:                349 *
CEB5:A9 00      (2)  350        lda   #0
CEB7:A2 02      (2)  351        ldx   #2
CEB9:D0 04   CEBF(3) 352        bne   secondsend
CEBB:                353 *
CEBB:                354 * For FileWrite, the buffer address is at CMDbuffer
CEBB:                355 *  and the length is at CMDblock.
CEBB:                356 *
CEBB:        CEBB    357 NOWBlock equ *
CEBB:A6 47      (3)  358        ldx   CMDBlockh
CEBD:A5 46      (3)  359        lda   CMDBlockl
CEBF:                360 *
CEBF:        CEBF    361 secondsend equ *
CEBF:86 4E      (3)  362        stx   bytecounth
CEC1:85 4D      (3)  363        sta   bytecountl
CEC3:                364 *
CEC3:A9 82      (2)  365        lda   #datamark
CEC5:85 5B      (3)  366        sta   WPacketType   ;Identify this as a data packet
```

```
CEC7:                   367 *
CEC7:20 DD CA    (6)    368         jsr   SendData
CECA:90 04   CED0(3)    369         bcc   noxtrasend
CECC:        CECC       370 behitch equ   *
CECC:A9 06       (2)    371         lda   #BusErr       ;This is the bus error hitch
CECE:D0 3F   CF0F(3)    372         bne   Error
CED0:                   373 *
CED0:                   374 * On ProDOS status call, we've got to point the buffer pointer
CED0:                   375 *  correctly to zero page... it's the only case special case
CED0:                   376 *  (on Write, Format and Control no data comes back).
CED0:                   377 *
CED0:        CED0       378 noxtrasend equ *
CED0:A4 58       (3)    379         ldy   Slot
CED2:B9 73 04    (4)    380         lda   ProFlag,y
CED5:10 0C   CEE3(3)    381         bpl   getresults
CED7:A5 42       (3)    382         lda   cmdcode
CED9:D0 08   CEE3(3)    383         bne   getresults
CEDB:                   384 *
CEDB:A9 45       (2)    385         lda   #>CMDBufferh  ;Want status in these four
CEDD:A2 00       (2)    386         ldx   #<CMDBufferh
CEDF:85 54       (3)    387         sta   buffer
CEE1:86 55       (3)    388         stx   buffer+1
CEE3:                   389 *
CEE3:                   390 * Please to be calling ReceivePack
CEE3:                   391 *
CEE3:        CEE3       392 getresults equ *
CEE3:20 33 CB    (6)    393         jsr   RecPack       ;Get status byte (maybe read data too)
CEE6:B0 E4   CECC(3)    394         bcs   behitch
CEE8:                   395 *
CEE8:                   396 * Figure how many bytes were sent and put that in X,Y temps
CEE8:                   397 *
CEE8:20 50 CC    (6)    398         jsr   Rcvcount      ;Do the times 7...
CEEB:20 F2 CF    (6)    399         jsr   squirrel      ;Store away count in SHTEMPs
CEEE:                   400 *
CEEE:                   401 * For the ProDOS status call, we've got to look at the status byte
CEEE:                   402 *  returned and return a DIP error if appropriate.
CEEE:                   403 *  Also overwrite the X,Y temps with # blocks if this is a ProDOS
CEEE:                   404 * Stat call.
CEEE:A5 42       (3)    405         lda   CMDCode       ;Is it a ProDOS status call
CEF0:D0 1B   CF0D(3)    406         bne   noerror
CEF2:A6 58       (3)    407         ldx   Slot
CEF4:BD 73 04    (4)    408         lda   ProFlag,x
CEF7:10 14   CF0D(3)    409         bpl   noerror
CEF9:                   410 *
CEF9:A5 46       (3)    411         lda   CMDBlockl     ;This'll get loaded into the XY regs
                                                          later
CEFB:9D F3 05    (5)    412         sta   SHTempX,x
CEFE:A5 47       (3)    413         lda   CMDBlockh
CF00:9D 73 06    (5)    414         sta   SHTempY,x
CF03:                   415 *
CF03:A5 45       (3)    416         lda   CMDBufferh    ;Check status byte
CF05:29 10       (2)    417         and   #SVMask1
CF07:D0 04   CF0D(3)    418         bne   noerror       ;No DIP
CF09:A9 2F       (2)    419         lda   #OffLine
CF0B:D0 02   CF0F(3)    420         bne   Error
CF0D:                   421 *
CF0D:                   422 * Now it's time to think about returning to the caller
CF0D:                   423 *  Remember that ProDOS doesn't want to know about soft errors,
CF0D:                   424 *  only fatal ones.  If this is a ProDOS call, and the soft error
```

```
CF0D:                   425 *  bit in the statbyte is set, there IS NO error (statbyte is
                               cleared).
CF0D:                   426 *  Also, ProDOS wants only I/O, Write Protect, No Device, Offline.
CF0D:                   427 *  If any other hard error comes from the device on a ProDOS call,
CF0D:                   428 *  map it to an I/O Error.  (Gross me out.)
CF0D:                   429 *
CF0D:          CF0D     430 noerror equ  *
CF0D:A5 4D       (3)    431         lda   statbyte
CF0F:          CF0F     432 Error   equ   *
CF0F:A4 58       (3)    433         ldy   Slot            ;Need access to screenholes
CF11:99 F3 04    (5)    434         sta   Retry,Y         ;Keep unadulterated error in shole
CF14:AA          (2)    435         tax                   ;Set the Z flag
CF15:F0 1A   CF31(3)    436         beq   sa2             ;Special case the zero
CF17:                   437 *
CF17:BE 73 04    (4)    438         ldx   ProFlag,y       ;Set N to ProDOS call or not
CF1A:10 15   CF31(3)    439         bpl   sa2             ;If PC call, no mapping occurs
CF1C:                   440 *
CF1C:A2 00       (2)    441         ldx   #0              ;Assume a soft error
CF1E:C9 40       (2)    442         cmp   #%01000000      ;Soft error check
CF20:B0 0E   CF30(3)    443         bge   storeaway       ;If $40 or bigger, map to zero
CF22:                   444 *
CF22:A2 27       (2)    445         ldx   #IOError        ;Now anticipate ProDOS I/O error
CF24:C9 2B       (2)    446         cmp   #WriteProt
CF26:F0 09   CF31(3)    447         beq   sa2             ;OK to return Write Protect
CF28:C9 28       (2)    448         cmp   #NoDrive
CF2A:F0 05   CF31(3)    449         beq   sa2             ;OK to return Drive disconnected
CF2C:C9 2F       (2)    450         cmp   #OffLine
CF2E:F0 01   CF31(3)    451         beq   SA2
CF30:                   452 *
CF30:          CF30     453 storeaway equ *
CF30:8A          (2)    454         txa                   ;Use the default value
CF31:          CF31     455 sa2     equ   *
CF31:A4 58       (3)    456         ldy   Slot
CF33:99 73 05    (5)    457         sta   SHTemp1,y       ;Keep in screenhole
CF36:                   458 *
CF36:                   459 * If this is the //c version, we need to reset the IWM to its
CF36:                   460 *  former disk // state.  This is done by setting the mode register
CF36:                   461 *  to a little known (and less documented) mode which speeds up the
CF36:                   462 *  internal motor timeout.  When the motor enable has timed out,
CF36:                   463 *  mode can be set back to zero.  This method is necessary because
CF36:                   464 *  if the timer is enabled within the timeout period, the motor on
CF36:                   465 *  a Rev A IWM pops on for the full timeout period (since mode
                               changes
CF36:                   466 *  are disabled when the motor is on.  I know, it's bizzarre. Blame
                               Mac.
CF36:          0001     467          do    IIc
CF36:AD E8 C0    (4)    468          lda   monclr+$60      ;Motor off
CF39:2C ED C0    (4)    469          bit   16set+$60       ;Into mode reg access mode
CF3C:A9 2B       (2)    470          lda   #$2B            ;This is the magic "speed up" value
CF3E:8D EF C0    (4)    471          sta   17set+$60       ;Throw into mode register
CF41:EA          (2)    472          nop                   ;You're supposed to wait a while
CF42:EA          (2)    473          nop
CF43:EA          (2)    474          nop
CF44:EA          (2)    475          nop
CF45:          CF45     476 waitoff equ  *
CF45:AD EE C0    (4)    477          lda   17clr+$60       ;Wait 'til motor off
CF48:29 20       (2)    478          and   #$20
CF4A:D0 F9   CF45(3)    479          bne   waitoff
CF4C:A0 00       (2)    480          ldy   #0              ;Now set the reg back to $00
CF4E:A2 60       (2)    481          ldx   #$60            ;IWM's in slot 6
CF50:20 1F CC    (6)    482          jsr   SetIWMode
```

```
CF53:AD EC C0    (4)  483         lda   16clr+$60
CF56:AD E2 C0    (4)  484         lda   ca1clr+$60
CF59:AD E6 C0    (4)  485         lda   1strbclr+$60
CF5C:A4 58       (3)  486         ldy   Slot            ;Need Slot in Y
CF5E:                 487         fin
CF5E:                 488 *
CF5E:                 489 * Now, restore our zero page area.
CF5E:                 490 *
CF5E:A2 00       (2)  491         ldx   #0
CF60:68          (4)  492 rzp     pla
CF61:95 40       (4)  493         sta   zeropage,x
CF63:E8          (2)  494         inx
CF64:E0 1C       (2)  495         cpx   #ZPSize
CF66:90 F8   CF60(3)  496         blt   rzp
CF68:                 497 *
CF68:                 498 * We're into the stretch!  Restore interrupt mask, load X, Y, and A
CF68:                 499 *  and set the carry if the error byte is non-zero.
CF68:                 500 *
CF68:28          (4)  501         plp                   ;Restore interrupt flag
CF69:B9 F3 05    (4)  502         lda   SHTempx,y       ;Get X value
CF6C:AA          (2)  503         tax
CF6D:B9 73 05    (4)  504         lda   SHTemp1,y       ;Grab the error result code
CF70:48          (3)  505         pha
CF71:B9 73 06    (4)  506         lda   SHTempy,y       ;Pull out the Y value
CF74:A8          (2)  507         tay                   ;No more access to screenholes
CF75:18          (2)  508         clc                   ;Anticipate zero result code
CF76:68          (4)  509         pla                   ;Pull back result code
CF77:F0 01   CF7A(3)  510         beq   finalskip       ;Return with carry clear
CF79:38          (2)  511         sec                   ;Some type of error
CF7A:         CF7A    512 finalskip equ *
CF7A:                 513 *
CF7A:         0001    514         do    IIc^ROM
CF7A:08          (3)  515         php                   ;Save carry and Z flag
CF7B:2C 78 04    (4)  516         bit   ProFlag+5       ;Ick - ProFlag is fixed in //c
CF7E:70 04   CF84(3)  517         bvs   ick1            ;If bit 6=1, then return to alt ROM
CF80:28          (4)  518         plp                   ;Vclr so return across ROM bank bdy
CF81:4C 84 C7    (3)  519         jmp   SWRTS2
CF84:         CF84    520 ick1    equ   *
CF84:28          (4)  521         plp
CF85:60          (6)  522         rts                   ;Flags set correctly again
CF86:                 523         else
CF86:                 525         fin
CF86:                 526 *
CF86:                 527 *
CF86:         CF86    528 parmctab equ *
CF86:03               529         dfb   %00000011       ;Status:  3 parms/no data send
CF87:03               530         dfb   %00000011       ;Read:    3 parms/no data send
CF88:83               531         dfb   %10000011       ;Write:   3 parms/data send
CF89:01               532         dfb   %00000001       ;Format:  1 parm /no data send
CF8A:83               533         dfb   %10000011       ;Control: 3 parms/data send
CF8B:01               534         dfb   %00000001       ;Init:    1 parm /no data send
CF8C:01               535         dfb   %00000001       ;Open:    1 parm /no data send
CF8D:01               536         dfb   %00000001       ;Close:   1 parm /no data send
CF8E:03               537         dfb   %00000011       ;CharRead: 3 parms/data send
CF8F:83               538         dfb   %10000011       ;CharWrite: 3 parms/data send
CF90:                 539 *
CF90:                 540 *
```

```
CF90:                   542 *
CF90:          CF90     543 AssignID equ *
CF90:48           (3)   544         pha                 ;Save the init code
CF91:20 60 CA     (6)   545         jsr   resetchain    ;Reset all of those things
CF94:68           (4)   546         pla
CF95:AA           (2)   547         tax                 ;Save InitCode
CF96:                   548 *
CF96:                   549 * Save the command code, unit, and init code 'cause we'll trample
CF96:                   550 * 'em.
CF96:A5 42        (3)   551         lda   CMDCode
CF98:48           (3)   552         pha
CF99:A5 43        (3)   553         lda   CMDPCount
CF9B:48           (3)   554         pha
CF9C:A5 46        (3)   555         lda   CMDSCode
CF9E:48           (3)   556         pha
CF9F:86 46        (3)   557         stx   CMDSCode      ;Store away the type of INIT
CFA1:                   558 *
CFA1:                   559 * Set up to send DefID command packets
CFA1:                   560 *
CFA1:A9 05        (2)   561         lda   #InitCmd
CFA3:85 42        (3)   562         sta   CMDCode
CFA5:A9 00        (2)   563         lda   #0
CFA7:85 5A        (3)   564         sta   Unit
CFA9:A9 02        (2)   565         lda   #2            ;# parms in Init call
CFAB:85 43        (3)   566         sta   CMDPCount
CFAD:                   567 *
CFAD:                   568 * Point the buffer pointer
CFAD:                   569 *
CFAD:A9 42        (2)   570         lda   #>CMDCode
CFAF:85 54        (3)   571         sta   buffer
CFB1:A9 00        (2)   572         lda   #<CMDCode
CFB3:85 55        (3)   573         sta   buffer+1
CFB5:A9 80        (2)   574         lda   #cmdmark
CFB7:85 5B        (3)   575         sta   WPacketType
CFB9:                   576 *
CFB9:20 8A CA     (6)   577         jsr   ClrPhases     ;Make sure phases are off
CFBC:                   578 *
CFBC:                   579 * Send an ID for the next device in the chain
CFBC:                   580 *
CFBC:          CFBC     581 mordevices equ *
CFBC:E6 5A        (5)   582         inc   Unit
CFBE:A9 09        (2)   583         lda   #>cmdlength
CFC0:85 4D        (3)   584         sta   bytecountl    ;ReceivePack scrambles count
CFC2:A9 00        (2)   585         lda   #<cmdlength
CFC4:85 4E        (3)   586         sta   bytecounth
CFC6:                   587 *
CFC6:20 86 C8     (6)   588         jsr   SendOnePack   ;Send the command
CFC9:90 05    CFD0(3)   589         bcc   mdev2         ;If okay, skip to get response
CFCB:                   590 *
CFCB:C6 5A        (5)   591         dec   Unit
CFCD:4C D7 CF     (3)   592         jmp   mdev1
CFD0:                   593 *
CFD0:20 E6 C9     (6)   594 mdev2   jsr   ReceivePack   ;Get the response
CFD3:A5 4D        (3)   595         lda   statbyte
CFD5:F0 E5    CFBC(3)   596         beq   mordevices
CFD7:                   597 *
CFD7:                   598 * Okay, we done last device.  Squirrel away the number of devices.
CFD7:                   599 *
```

```
CFD7:A5 5A      (3)  600 mdev1  lda   Unit
CFD9:A4 58      (3)  601        ldy   slot
CFDB:99 F9 06   (5)  602        sta   NumDevices,y  ;Devices out there
CFDE:                603 *
CFDE:                604 * Recover the scrambled ProDOS parms
CFDE:                605 *
CFDE:68         (4)  606        pla
CFDF:85 46      (3)  607        sta   CMDSCode
CFE1:68         (4)  608        pla
CFE2:85 43      (3)  609        sta   CMDPCount
CFE4:68         (4)  610        pla
CFE5:85 42      (3)  611        sta   CMDCode
CFE7:                612 *
CFE7:       0001     613        ifeq  IIc^ROM
CFE7:                622        fin
CFE7:60         (6)  623        rts
CFE8:                624 *
CFE8:                625 *
CFE8:       0001     626        do    IIc
CFE8:       CFE8     627 AppleTalkEntry equ *
CFE8:                628 *
CFE8:                629 * This is an entry for the //c AppleTalk stump.
CFE8:                630 *
CFE8:A2 05      (2)  631        ldx   #5
CFEA:A9 40      (2)  632        lda   #%01000000    ;PC call & return to alt ROM
CFEC:9D 73 04   (5)  633        sta   ProFlag,x
CFEF:4C 55 CD   (3)  634        jmp   atentry       ;Just like normal
CFF2:                635        fin
CFF2:                636 *
CFF2:                637 *
CFF2:       CFF2     638 squirrel equ *
CFF2:A6 58      (3)  639        ldx   Slot
CFF4:9D F3 05   (5)  640        sta   SHTempX,x
CFF7:98         (2)  641        tya
CFF8:9D 73 06   (5)  642        sta   SHTempY,x
CFFB:60         (6)  643        rts
CFFC:                644 *
CFFC:                645 *
CFFC:                142 *
CFFC:       0001     143        ifeq  IIc^ROM
CFFC:                145        fin
CFFC:                146 *
CFFC:       CFFC     147 zzzzz  equ   *
CFFC:       0001     148        ifeq  IIc^ROM       ;If not //c ROM, pad bytes
CFFC:                153        fin
CFFC:                154 *
```

```
 C90E ACHE1          CD8F ALLSET1        ?CD7D ALLSET        ?CB04 ALTSENDPILE
 CE3C AOKAYHITCH     CDEC AOKAY           CFE8 APPLETALKENTRY CF90 ASSIGNID
 CD55 ATENTRY       ?FABA AUTOSCAN        CB61 AUXPTRINC        56 AUXPTR
?  4E AUXTYPE       ?  2D BADBLOCK          01 BADCMD        ?  22 BADCTLPARM
   21 BADCTL           04 BADPCNT           11 BADUNIT       ?E000 BASIC
 C52F BC1            CECC BEHITCH         CD50 BENTRY         CC28 BIZ
 0011 BMSGLEN        C521 BOOTC          ?C514 BOOTCASE5      C523 BOOTCODE
 C552 BOOTFAIL       C55F BOOTMSG         07DB BOOTSCRN       C570 BOOTTAB
   32 BSYTO1           0A BSYTO2            54 BUFFER           56 BUFFER2
 CE27 BUNIT            06 BUSERR         ?  40 BUSHOG        ?  08 BYTECMP
   4D BYTECOUNT        4E BYTECOUNTH        4D BYTECOUNTL    ?C500 C500ORG
 C082 CA1CLR         C083 CA1SET          C084 CA2CLR         C085 CA2SET
 CC2C CAREFUL        C8A5 CHAINUNBSY        40 CHECKSUM    ?  24 CH
 CFFF CLEARIOROMS    CA8A CLRPHASES         47 CMDBLOCKH        46 CMDBLOCKL
   48 CMDBLOCKS   ?  46 CMDBLOCK           45 CMDBUFFERH       44 CMDBUFFERL
   44 CMDBUFFER        42 CMDCODE           09 CMDLENGTH     ?C58A CMDLIST
   80 CMDMARK          43 CMDPCOUNT         46 CMDSCODE      ?  49 CMDSPARE1
?  4A CMDSPARE2        43 CMDUNIT         C55D COMA             80 COMMRESET
   04 CONTROLCMD     CDC9 COPYLOOP       ?FDED COUT          CD47 CSERROR5
   10 CSUMERR        ?  25 CV             CD9B DARNIT           82 DATAMARK
 C999 DATDONE         C9C3 DBERROR       ?CBE1 DETTOPBITS    ?  50 DEVICEID
 CE37 DISABINT        CB55 DIV7TAB         CB9E DIVIDE1        CBA7 DIVIDE2
 CB8E DIVIDE3         CB95 DIVIDE4         CBB0 DIVIDE5     ?CB64 DIVIDE7
 CD09 DONE5           CE2B ENABINT       ?C08A ENABLE1         C08B ENABLE2
 CA80 ENABLECHAIN     CD4B ENTRY           CF0F ERROR          CDBE ERRORHITCH
 CE25 ERRORHITCH2     CF7A FINALSKIP    ?  03 FORMATCMD        CEE3 GETRESULTS
 CA47 GOB1          ?C9E6 GRABSTATUS       4B GRP7CTR          CD49 GSERROR5
 CBE8 GTBOB         ?  51 HOSTID         CD33 ICBT15           CD2D ICBT5
 CF84 ICK1             01 IIC              05 INITCMD           27 IOERROR
 C080 IWM              07 IWMMODE        C08C L6CLR            C08D L6SET
 C08E L7CLR          C08F L7SET         ?  68 LASTONE         CBCF LASTPASS
   00 LOC0           ?  01 LOC1         ?CD36 LSTBSYWAIT5     C086 LSTRBCLR
 C087 LSTRBSET        C9E3 MARKERR         CE18 MAYBECTRL      CFD7 MDEV1
 CFD0 MDEV2           C50D MLIENTRY        CB5B MOD7TAB        C088 MONCLR
 C089 MONSET          C554 MORCHRS         CFBC MORDEVICES     07F8 MSLOT
   4D NEXT1             4E NEXT2             50 NEXT4            4D NEXT
   4F NEXT3             51 NEXT5             52 NEXT6            53 NEXT7
 CDFF NIN1              01 NOANSWER        CB7B NOAUXPTR       CEB1 NOCONTROL
   28 NODRIVE        CDC1 NOEH            CF0D NOERROR      ?  1F NOINT
?  02 NOMARK            20 NOPACKEND       CD6E NOPLAY         CDF1 NOTINIT
 CE72 NOTSTAT         CEBB NOWBLOCK        CED0 NOXTRASEND     CD43 NPENDERR5
 06F9 NUMDEVICES        4C ODDBYTES          2F OFFLINE       ?CDC7 OKAYCNT
 CA7C ONEMS1          CA7A ONEMS             C3 PACKETBEG        C8 PACKETEND
 CF86 PARMCTAB        C9BE PATCH1        ?  A5 PBBVALUE      ?  FF PBCVALUE
   00 PCID2             BF PDIDBYTE        CB4F PDIV7TAB       CB52 PMOD7TAB
?  52 POINTER           00 POWERRESET      C9D6 PREAMBLE     ?CBB0 PRECHECK
 C50A PRODOSENTRY     0473 PROFLAG         CD75 PZP            0BB8 RC1
   05 RC2             C583 RCODE             4B RCVBUF         CC50 RCVCOUNT
 C9F8 RDH1            CA02 RDH2            CA10 RDH3           CD39 RDH45
?CA0E RDH5            CD1E RDHA5             01 READCMD        C9E6 RECEIVEPACK
 CB33 RECPACK         C080 REQCLR          C081 REQSET         CA60 RESETCHAIN
 C576 RESET           0573 RETRY2          04F3 RETRY            01 ROM
?  4F RPACKETTYPE     CB3A RPK1            CB4E RPOUT          C578 RST1
 CF60 RZP             CF31 SA2             CB76 SAP1        ?  00 SCDEVICESTAT
?  01 SCGETDCB      ?  03 SCGETDEVINFO     0473 SCHOLES        C99D SCM1
?  02 SCRETNLSTAT     C9CF SD10            C9B2 SD7            C9C9 SD9
 CAEB SDOUBT          CEBF SECONDSEND      CA51 SEND80         CA53 SENDBYTE
 CADD SENDDATA        C886 SENDONEPACK     CAEC SENDPACK       CB00 SENDPILE
```

```
 CC1F SETIWMODE     ?FE89 SETKBD       ?FE93 SETVID        CA9A SETXN0
 CA9D SHIFT1         CAAD SHIFT2        CABD SHIFT3        CACD SHIFT4
 0573 SHTEMP1        05F3 SHTEMPX       0673 SHTEMPY       C927 SKIP1
 C929 SKIP2          C963 SKIP3         C965 SKIP4         CE3F SKIPCOPY
   58 SLOT           CC72 SLOTDEPRD     C8E8 SOB1          C8F9 SOB2
 C900 SOB3        ?  67 SOFTERROR        40 SOFT          CB0D SPILE1
 CB32 SPILOUT        CFF2 SQUIRREL      C8AF SSB           C8B2 SSD
?0100 STACK          CA34 START0        CA3C START1       ?CC72 START25
 C903 START          CA4E START2        CC7C START35         4D STATBYTE
?  81 STATMARK         1E STATMTO     ?  00 STATUSCMD      CF30 STOREAWAY
 CC0B SUN1           CC02 SUN2         ?CC00 SUN           CC1E SUN3
 0778 SVBCH          06F8 SVBCL           10 SVMASK1       C797 SWPROTO
 C784 SWRTS2        ?C9D7 SYNCTAB       CC66 T71           CC6F T72
   59 TBODD            59 TEMP           60 THEOFF       ?C000 THEORG
 CC59 TIMES7           41 TOPBITS       C899 UBSY1           5A UNIT
?1000 VERSION       ?FC22 VTAB         CC35 WAITIWMOFF     CF45 WAITOFF
?  04 WASRESET      ?C9E2 WASTE12      C9E1 WASTE14       ?C9E0 WASTE16
?C9DF WASTE18       ?C9DC WASTE32      CC3E WIWM1          CC4B WIWM2
   5B WPACKETTYPE      02 WRITECMD     CB64 WRITEPREP        2B WRITEPROT
 CBBC XOR1          ?CBB9 XOR2         CBD5 XOR3          CBDC XOR4
 CBCD XOR5           CA73 YMSWAIT        40 ZEROPAGE        1C ZPSIZE
?CFFC ZZZZZ
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 30-APR-85 22:46
** TOTAL LINES ASSEMBLED  3969
** FREE SPACE PAGE COUNT   70
```

```
SOURCE   FILE #01 =>INCLUDES.2CROM
 INCLUDE FILE #02 =>APTALK.2CVARS
 INCLUDE FILE #03 =>APTALK.C700
 INCLUDE FILE #04 =>APTALK.ROMSTUFF

0000:                2 ********************************************************
0000:                3 *                                                      *
0000:                4 *                     AppleTalk //c                    *
0000:                5 *                                                      *
0000:                6 *                     INCLUDES File                    *
0000:                7 *                                                      *
0000:                8 *                          by                          *
0000:                9 *                     Fern Bachman                     *
0000:               10 *                                                      *
0000:               11 *          Copyright Apple Computer, Inc. 1985         *
0000:               12 *                All Rights Reserved.                  *
0000:               13 *                                                      *
0000:               14 ********************************************************

0000:               16 * This file contains the includes necessary to
0000:               17 * generate the AppleTalk //c code which goes in the
0000:               18 * //c ROM.


0000:        0002   20           X6502                ;Allow 65C02 opcodes!!
0000:               21           MSB   ON

0000:               23           INCLUDE APTALK.2CVARS
```

```
0000:                3 ********************************************************
0000:                4 *                                                      *
0000:                5 *           AppleTalk //c Protocol Converter           *
0000:                6 *                                                      *
0000:                7 *                      Variables                       *
0000:                8 *                                                      *
0000:                9 *                         by                           *
0000:               10 *                    Fern Bachman                      *
0000:               11 *                                                      *
0000:               12 *         Copyright Apple Computer, Inc. 1985          *
0000:               13 *               All Rights Reserved.                   *
0000:               14 *                                                      *
0000:               15 ********************************************************

0000:               17 * Apple //c zero page used at boot and not restored.

0000:        0008   19 ZP8       EQU   $8            ;Used and not restored

0000:               21 * AppleTalk //c Converter Box stuff

0000:        0081   23 DIAGCMD   EQU   $81           ;Diag call command #

0000:               25 * The following table contains the only
0000:               26 * valid CODESCMD's recognized by the
0000:               27 * AppleTalk//c box when using the protocol
0000:               28 * converter's STATUS command.

0000:               30 *  $0=Short status request
0000:               31 *  $1=Return DCB info
0000:               32 *  $2=NEWLINE info
0000:               33 *  $3=Return DIB
0000:        0004   34 CMDCINIT  EQU   $4            ;$4=AppleTalk Init command
0000:        0005   35 CMDCSTATUS EQU  $5            ;$5=AppleTalk Status command
0000:        0006   36 CMDCREADREST EQU $6           ;$6=AppleTalk Readrest cmd
0000:        0007   37 CMDCREADPROT EQU $7           ;$7=AppleTalk Readprot cmd
0000:        0008   38 CMDCDIAG  EQU   $8            ;$8=AppleTalk Diag command
0000:        0009   39 CMDCREBOOT EQU  $9            ;$9=AppleTalk Reboot command
0000:        000A   40 CMDCID1   EQU   $A            ;$A=AppleTalk ID call 1
0000:               41 *  $B=AppleTalk ID call 2


0000:               43 * Protocol converter commands used by the
0000:               44 * AppleTalk //c firmware.

0000:        0000   46 PCSTATUSCMD EQU $0            ;Prot Conv status command
0000:        0009   47 PCWRITECMD EQU  $9            ;Prot Conv write command


0000:               49 * RELVERNUM is the version number
0000:               50 * for 65C02 RELease VERsion NUMber.
0000:               51 * It must be kept updated as this product
0000:               52 * is updated.

0000:        0000   54 RELVERNUM EQU   0             ;Release version #=0


0000:               56 *  STATBYTE codes
```

```
0000:          00A8   58 NODEVCON  EQU   $28+$80        ;Dev to access not connected

0000:                 60 * AppleTalk specific error codes for STATBYTE
0000:                 61 * used by the AppleTalk //c firmware.

0000:          00B4   63 NOUNIQUEID EQU  $80+$30+$04    ;No unique node addr found
0000:          00B5   64 BYTEGTR603 EQU  $80+$30+$05    ;# bytes to send >603
0000:          00BF   65 LASTPACKET EQU  $80+$30+$0F    ;Last packet in series
0000:          00C6   66 ID1        EQU  'F             ;ID byte1 for finding ApTalk
0000:          00C2   67 ID2        EQU  'B             ;ID byte2 for finding ApTalk


0000:          0004   69 PCOUNTW    EQU  $4             ;Write call PCOUNT
0000:          0006   70 PCOUNTS4.B EQU  $6             ;Status call PCOUNT for 4-B


0000:                 72 * Apple //c zero page usage

0000:          0039   74 KSWH       EQU  $39            ;Input hook hi byte

0000:                 76           DSECT
00C0:          00C0   77           ORG  $C0            ;
00C0:          00C0   78 ZP2CUSE    EQU  *
00C0:          0001   79 PARAMNUM   DS   1,0            ;Number of parameters
00C1:          0001   80 NUMUNIT    DS   1,0            ;Unit number
00C2:          0002   81 PTRBUFF    DS   2,0            ;Buffer pointer
00C4:          00C4   82 CODECMDS   EQU  *              ;Command code
00C4:          0001   83 NUMLOWRITE DS   1,0            ;# of bytes to write lo byte
00C5:          00C5   84 BYTELONUM  EQU  *              ;# of bytes to read from box
00C5:          00C5   85 NUMHIWRITE EQU  *              ;# of bytes to write hi byte
00C5:          0001   86 BYTEUSER   DS   1,0            ;User info byte
00C6:          00C6   87 BYTEHINUM  EQU  *              ;# of bytes to read from box
00C6:          0001   88 TYPEWRITE  DS   1,0            ;Write type code
00C7:          0002   89 TESTTMP    DS   2,0            ;Diag read from address

00C9:          0001   91 ADDR0      DS   1,0            ;Used as temp and restored
00CA:          0001   92 ADDR1      DS   1,0            ;Used as temp and restored

00CB:          000B   94 ZP2CUSELEN EQU  *-ZP2CUSE      ;# of bytes used in //c zpage

00CB:          0002   96 BOOTIT     DS   2,0            ;Boot prog strt adr

0000:                 98           DEND

0000:                100 * AppleTalk //c non zero page usage

0000:          03F2  102 SOFTEV     EQU  $3F2           ;Reset vector
0000:          07FE  103 APTALKUNIT EQU  $7FE           ;Unit # screen hole
0000:          047F  104 SCRNHOLE0  EQU  $47F           ;DRIVERFLAG placed here
0000:          077F  105 SCRNHOLE1  EQU  $77F           ;Print drvr #>start-1
0000:          07FF  106 SCRNHOLE2  EQU  $7FF           ;Print drvr #<start-1
0000:          0478  107 SCRNTMP0   EQU  $478           ;Temp use only
0000:          07F8  108 MSLOT      EQU  $7F8           ;Card slot # ($Cn) for ints
0000:          C7D3  109 ALTROMSW   EQU  $C7D3          ;Switch to alt ROM vector
0000:          C784  110 MAINROMSW  EQU  $C784          ;Switch to main ROM vector
0000:          C883  111 ALTPRCNVENTRY EQU $C883        ;Alt ROM prot conv entry point
0000:          FB2F  112 INIT       EQU  $FB2F          ;Get in Text mode
```

```
02 APTALK.2CVARS   AppleTalk //c Variables.            22-APR-85  16:01 PAGE 4

0000:          FC58  113 HOME       EQU   $FC58        ;Clear screen
0000:          FE84  114 SETNORM    EQU   $FE84        ;Normal char display
0000:          FE89  115 SETKBD     EQU   $FE89        ;Keyboard is input device
0000:          FE93  116 SETVID     EQU   $FE93        ;Video is output device

---- NEXT OBJECT FILE NAME IS APTALK.2CROM.0
C700:          C700   24           ORG   $C700        ;Cn00 page for //c goes here
C700:                 25           INCLUDE APTALK.C700
```

```
C700:                 4 ********************************************************
C700:                 5 *                                                      *
C700:                 6 *                    AppleTalk //c                     *
C700:                 7 *                                                      *
C700:                 8 *                    $C700 Routines                    *
C700:                 9 *                                                      *
C700:                10 *                         by                           *
C700:                11 *                    Fern Bachman                      *
C700:                12 *                                                      *
C700:                13 *         Copyright Apple Computer, Inc. 1985          *
C700:                14 *               All Rights Reserved.                   *
C700:                15 *                                                      *
C700:                16 ********************************************************

C700:                18 * Entry at $C700 means that the user wants
C700:                19 * to initialize the printer driver interface
C700:                20 * if one is loaded into main memory.
C700:                21 * To determine whether a driver is available
C700:                22 * or not we must perform the following steps;
C700:                23 *
C700:                24 *  1. Determine which slot we are in to get $Cn.
C700:                25 *  2. Test the 1st screen hole $3B8+$Cn to verify
C700:                26 *     that it is $Cn ($Cn is the flag indicating
C700:                27 *     a driver has been installed.)
C700:                28 *
C700:                29 * If a driver is not available the monitor ROM
C700:                30 * is mapped in and a JMP to the monitor RESET
C700:                31 * routine is executed.
C700:                32 *
C700:                33 * If a driver is available we pass data to is
C700:                34 * in the following form;
C700:                35 *
C700:                36 *        Y = user Y
C700:                37 *        X = user X
C700:                38 *        A = user A
C700:                39 *        P = Print character status
C700:                40 *            V=1 if init printer driver requested
C700:                41 *            C=1 if input to printer
C700:                42 *            C=0 if output to printer
C700:                43 * The driver can test the input/output hooks hi
C700:                44 * bytes to determine if the call is from BASIC or
C700:                45 * from machine language.  If $37 is $Cn then the
C700:                46 * user did a PR#n.  If $39 is $Cn then the user
C700:                47 * did a IN#n.  If the hooks do not have $Cn as
C700:                48 * the high byte then the user entered from
C700:                49 * machine language.  It is up to the driver to
C700:                50 * correctly observe this protocol.

C700:2C 03 C7        52           BIT   TOSETV        ;
C703:        C703    53 TOSETV    EQU   *             ;Bit here to set 'V'
C703:70 1B   C720    54           BVS   BASICENT      ;
C705:                55 *BASICINPUT EQU *             ;BASIC wants char if here
C705:38              56           SEC                  ;Identifier byte #1 ($38)
C706:90              57           DFB   $90            ;BCC opcode
C707:                58 *BASICOUTPUT EQU *            ;BASIC sends char if here
C707:18              59           CLC                  ;Identifier byte #2 ($18)
```

```
C708:B8              60           CLV                  ;Clear V if entered near here
C709:50 15   C720    61           BVC   BASICENT       ;Skip PASCAL protocol stuff

C70B:                63 *GENERIC  EQU *                ;PASCAL generic sig byte
C70B:01              64           DFB   $01            ;
C70C:                65 *DEVSIG   EQU *                ;9=bus card//B=Apple tech ID
C70C:9B              66           DFB   $9B            ;
C70D:1C              67           DFB   >PASERR        ;Offset to PASCAL err rtne
C70E:1C              68           DFB   >PASERR        ;Offset to PASCAL err rtne
C70F:1C              69           DFB   >PASERR        ;Offset to PASCAL err rtne
C710:1C              70           DFB   >PASERR        ;Offset to PASCAL err rtne
C711:88              71           DFB   $88            ;<>0 if no offsets follow

C712:                73 * The entry point APPLETALK must appear at
C712:                74 * $Cn12 in this and all future AppleTalk cards
C712:                75 * for the Apple // product line.

C712:                77 *AppleTalk Call

C712:                79 *  LDY #<PARAMLST
C712:                80 *     ;Y must contain hi byte of parameter list
C712:                81 *  LDX $>PARAMLST :
C712:                82 *     ;X must contain lo byte of parameter list
C712:                83 *  LDA #$Cn
C712:                84 *     ;A must contain the slt # of the AppleTalk card+$C0
C712:                85 *  JSR APPLETALK
C712:                86 *     ;Call the interface (in ROM in //c and in RAM
C712:                87 *      section of peripheral card in //e)
C712:                88 *  BNE ERRROUTINE
C712:                89 *    ;<>0 then an err occurred

C712:                91 *APPLETALK EQU *               ;FIXED entry point!!!!!!!!!!!
C712:18              92           CLC                  ;Vector to actual routine
C713:80 2A   C73F    93           BRA   APPLETALK1     ;Go to AppleTalk entry ptr


C715:                95 * REBOOT is accessed by a JMP/JSR to $Cn15.
C715:                96 * This causes boot code to be transferred from the
C715:                97 * AppleTalk//c converter box ROM to the //c RAM and
C715:                98 * causes the execution of that code.

C715:        C715   100 REBOOTAPTALK EQU *             ;Jmp here to reboot AppleTalk
C715:38              101          SEC                  ;Set carry means reboot
C716:78              102          SEI                  ;No interrupts during boot
C717:A2 FF           103          LDX   #$FF           ;Reset stack ptr for boot
C719:9A              104          TXS                  ;
C71A:80 26   C742    105          BRA   APPLETALK2     ;


C71C:        C71C   107 PASERR    EQU   *              ;PASCAL error entry point
C71C:38              108          SEC                  ;Set carry for error
C71D:A2 03           109          LDX   #$03           ;Error code for PASCAL
C71F:60              110          RTS                  ;Back to PASCAL


C720:        0000   112 CN20FILL  EQU   $C720-*
C720:        0000   113           DS    CN20FILL,$00   ;Fill to $Cn20 for BASICENT
```

```
C720:          C720  115 BASICENT  EQU   *              ;MUST start at $Cn20
C720:8D 78 04        116            STA   SCRNTMP0       ;Save user's output byte
C723:A9 C7           117            LDA   #$C7           ;Say we're in slot 7
C725:8D F8 07        118            STA   MSLOT          ;>>>>>> REQUIRED <<<<<<
C728:08              119            PHP                   ;Save V/C status
C729:C5 39           120            CMP   KSWH           ;If=KSWH then IN#n was done
C72B:F0 E8    C715   121            BEQ   REBOOTAPTALK   ;If so then must reboot
C72D:28              122            PLP                   ;Restore V/C status
C72E:4D 7F 04        123            EOR   SCRNHOLE0      ;Test for driver installed
C731:D0 1A    C74D   124            BNE   APTALKOFFLN    ;<>= to flag then error
C733:AD FF 07        125            LDA   SCRNHOLE2      ;Hi byte of prntr drv prg
C736:48              126            PHA                   ;To stack for RTS type jump
C737:AD 7F 07        127            LDA   SCRNHOLE1      ;Lo byte-1 of prntr drv prg
C73A:48              128            PHA                   ;
C73B:AD 78 04        129            LDA   SCRNTMP0       ;Restore user's output byte
C73E:60              130            RTS                   ;Exit to printer driver


C73F:          C73F  132 APPLETALK1 EQU   *
C73F:8D F8 07        133            STA   MSLOT          ;Save $Cn in case of interrupt
C742:          C742  134 APPLETALK2 EQU   *
C742:20 D3 C7        135            JSR   ALTROMSW       ;Continue in alt ROM
C745:70 01    C748   136            BVS   FOS            ;V=1 if from boot code
C747:60              137            RTS                   ;V=0 then return to user

C748:          C748  139 FOS        EQU   *              ;From Other Side (alt ROM)
C748:B0 03    C74D   140            BCS   APTALKOFFLN    ;Error so display message
C74A:6C CB 00        141            JMP   (BOOTIT)       ;Start of boot code

C74D:          C74D  143 APTALKOFFLN EQU  *
C74D:AD 81 C0        144            LDA   $C081          ;Switch in LC ROM
C750:AD 81 C0        145            LDA   $C081          ;
C753:20 84 FE        146            JSR   SETNORM        ;No inverse stuff
C756:20 2F FB        147            JSR   INIT           ;Fix up some stuff
C759:20 58 FC        148            JSR   HOME           ;Clear screen for message
C75C:20 93 FE        149            JSR   SETVID         ;Screen is output device
C75F:20 89 FE        150            JSR   SETKBD         ;Keyboard is input device

C762:A0 10           152            LDY   #APOFFMSGLN-1  ;Length of error message
C764:          C764  153 APOFFLOOP  EQU   *
C764:B9 6F C7        154            LDA   APOFFMSG,Y     ;Get character to show
C767:99 DB 07        155            STA   $7DB,Y         ;Display on screen
C76A:88              156            DEY                   ;
C76B:10 F7    C764   157            BPL   APOFFLOOP      ;Loop til done
C76D:          C76D  158 BRAHANGLOOP EQU  *              ;Hang til user presses reset
C76D:80 FE    C76D   159            BRA   BRAHANGLOOP    ;Loop forever

C76F:                161            MSB   ON

C76F:          C76F  163 APOFFMSG   EQU   *
C76F:C1 F0 F0 EC     164            ASC   "AppleTalk    Offline"

C780:          0011  166 APOFFMSGLN EQU   *-APOFFMSG     ;Length of error message


C780:          0000  168 C7FILL80   EQU   $C780-*        ;
C780:          0000  169            DS    C7FILL80,$FF   ;Fill to version number
```

```
0000:                171           DSECT
C7FF:          C7FF  172           ORG   $C7FF          ;Version # goes at $C7FF

C7FF:                174 *RELVERSION EQU *              ;Release version number
C7FF:00              175           DFB   RELVERNUM      ;


C780:                177           DEND


---- NEXT OBJECT FILE NAME IS APTALK.2CROM.1
C580:          C580   26           ORG   $C580          ;$C580-$C77F in aux ROM
C580:                 27           INCLUDE APTALK.ROMSTUFF
```

```
C580:                  4 ********************************************************
C580:                  5 *                                                      *
C580:                  6 *            AppleTalk //c Protocol Converter          *
C580:                  7 *                                                      *
C580:                  8 *              Alternate ROM Stuff Routines            *
C580:                  9 *                                                      *
C580:                 10 *                          by                          *
C580:                 11 *                      Fern Bachman                    *
C580:                 12 *                                                      *
C580:                 13 *          Copyright Apple Computer, Inc. 1985         *
C580:                 14 *                All Rights Reserved.                  *
C580:                 15 *                                                      *
C580:                 16 ********************************************************

C580:                 18 *ARAPPLETALK EQU *              ;Alternate ROM entry point
C580:90 59    C5DB    19            BCC   ARAPPLETALK2   ;C=0 then regular ApTalk call

C582:                 21 *DOBOOTCODE EQU *              ;Alt ROM ApTalk Reboot entry
C582:A9 C7            22            LDA   #$C7          ;Put $Cn at $8 for boot program
C584:85 08            23            STA   ZP8           ;
C586:85 C7            24            STA   TESTTMP        ;<>0 then indicates from here
C588:20 06 C6         25            JSR   ARINIT1       ;Verify AppleTalk online
C58B:B0 38    C5C5    26            BCS   GETCODE4      ;C=1 then offline

C58D:64 C3            28            STZ   PTRBUFF+1     ;Response buffer is same
C58F:A9 C2            29            LDA   #>PTRBUFF     ;   as send buffer.
C591:85 C2            30            STA   PTRBUFF       ;
C593:A9 09            31            LDA   #CMDCREBOOT   ;Reboot command
C595:20 6A C7         32            JSR   CALLSETUP     ;Setup some stuff before JSR
C598:20 83 C8         33            JSR   ALTPRCNVENTRY ;Call the prot conv
C59B:00               34            DFB   PCSTATUSCMD   ;Prot Conv status command
C59C:C0 00            35            DW    ZP2CUSE       ;Parameter buffer
C59E:D0 25    C5C5    36            BNE   GETCODE4      ;<>= then errors

C5A0:A5 C2            38            LDA   PTRBUFF       ;Save start for later
C5A2:85 CB            39            STA   BOOTIT        ;
C5A4:A5 C3            40            LDA   PTRBUFF+1     ;
C5A6:85 CC            41            STA   BOOTIT+1      ;

C5A8:         C5A8    43 GETCODE2   EQU   *
C5A8:20 83 C8         44            JSR   ALTPRCNVENTRY ;Call the prot conv
C5AB:00               45            DFB   PCSTATUSCMD   ;Prot Conv status command
C5AC:C0 00            46            DW    ZP2CUSE       ;Parameter buffer
C5AE:F0 25    C5D5    47            BEQ   GETCODE5      ;= then no errors
C5B0:C9 3F            48            CMP   #LASTPACKET-$80 ;Last packet read yet?
C5B2:D0 11    C5C5    49            BNE   GETCODE4      ;<> last pkt then error
C5B4:18               50            CLC                  ;C=0 if last pkt received

C5B5:                 52 * ROM boot program received. Now enable ACIA
C5B5:                 53 * interrupt capability for AppleTalk boot
C5B5:                 54 * program.

C5B5:A9 C0            56            LDA   #$C0          ;
C5B7:8D F9 05         57            STA   $5F9          ;Enable firmware to pass int
```

```
C5BA:A9 0F          58            LDA   #$F           ;Set up ACIA
C5BC:0C 9A C0       59            TSB   $C09A         ;

C5BF:       C5BF    61 GETCODE3   EQU   *
C5BF:2C 74 C6       62            BIT   FF            ;V=1 to indicate from here
C5C2:4C 84 C7       63            JMP   MAINROMSW     ;Return to main ROM


C5C5:       C5C5    65 GETCODE4   EQU   *             ;Error exit for reboot rtne
C5C5:38             66            SEC                  ;C=1 on error
C5C6:9C F2 03       67            STZ   SOFTEV        ;RESET vctrs to basic
C5C9:A9 E0          68            LDA   #<$E000       ;BASIC coldstart location
C5CB:8D F3 03       69            STA   SOFTEV+1      ;
C5CE:49 A5          70            EOR   #$A5          ;
C5D0:8D F4 03       71            STA   SOFTEV+2      ;Power up byte
C5D3:80 EA   C5BF   72            BRA   GETCODE3      ;Exit this half of ROM


C5D5:       C5D5    74 GETCODE5   EQU   *
C5D5:E6 C3          75            INC   PTRBUFF+1     ;Inc for next block
C5D7:E6 C3          76            INC   PTRBUFF+1     ;
C5D9:80 CD   C5A8   77            BRA   GETCODE2      ;
```

```
C5DB:          C5DB    81 ARAPPLETALK2 EQU *              ;Reg AppleTalk call conts here
C5DB:08                82             PHP                  ;Make sure no ints in here
C5DC:D8                83             CLD                  ;MUST enter with Dec mode clear
C5DD:78                84             SEI                  ;Force off int ability
C5DE:8C 78 04          85             STY   SCRNTMP0       ;Save Y temporarily

C5E1:A0 0B             87             LDY   #ZP2CUSELEN    ;# of bytes to save on stk
C5E3:          C5E3    88 ARSTKSVE    EQU   *
C5E3:B9 BF 00          89             LDA   ZP2CUSE-1,Y    ;Get value to save
C5E6:48                90             PHA                  ;Save it
C5E7:88                91             DEY                  ;Test for more
C5E8:D0 F9    C5E3     92             BNE   ARSTKSVE       ;<>= go for more

C5EA:86 C9             94             STX   ADDR0          ;User data buffer ptr
C5EC:AE 78 04          95             LDX   SCRNTMP0       ;Recall 'Y'
C5EF:86 CA             96             STX   ADDR1          ;Hi byte of data buff ptr
C5F1:B1 C9             97             LDA   (ADDR0),Y      ;Get command #
C5F3:F0 7A    C66F     98             BEQ   ARAPTALK2      ;0 is invalid command
C5F5:30 78    C66F     99             BMI   ARAPTALK2      ;- then test for DIAG call
C5F7:C9 06            100             CMP   #6             ; else test for valid #
C5F9:B0 78    C673    101             BCS   CMDEXITE       ;>=6 is illegal
C5FB:0A               102             ASL                  ;Make command # into index
C5FC:AA               103             TAX                   ;
C5FD:C8               104             INY                   ;Inc to 2nd byte in user buff
C5FE:B1 C9            105             LDA   (ADDR0),Y      ;Pick up the data there
C600:C8               106             INY                   ;Inc index for later
C601:7C 74 C7         107             JMP   (APTALKCMDS-2,X) ;Jump to routine
```

```
C604:          C604  110 ARINIT    EQU   *               ;AppleTalk init call entry
C604:64 C7           111           STZ   TESTTMP         ;=0 indicates from here

C606:          C606  113 ARINIT1   EQU   *               ;AppleTalk init call entry
C606:9C FE 07        114           STZ   APTALKUNIT      ;
C609:          C609  115 ARINIT2   EQU   *
C609:EE FE 07        116           INC   APTALKUNIT      ;Save in screen hole
C60C:A9 0A           117           LDA   #CMDCID1        ;Commands code #
C60E:20 6A C7        118           JSR   CALLSETUP       ;Set up some stuff
C611:A9 C7           119           LDA   #$C7            ;Point to ROM in case
C613:85 C3           120           STA   PTRBUFF+1       ; some device sends us
C615:64 C2           121           STZ   PTRBUFF         ;Lo byte is zero

C617:20 83 C8        123           JSR   ALTPRCNVENTRY   ;Call the protocol converter
C61A:00              124           DFB   PCSTATUSCMD     ;Prot Conv status command
C61B:C0 00           125           DW    ZP2CUSE         ;Pointer to call buffer
C61D:C9 46           126           CMP   #ID1-$80        ;ApTalk ID status code 1??
C61F:D0 0F    C630   127           BNE   NOTHISUNIT      ;Not ID1 then maybe last unit #

C621:E6 C4           129           INC   CODECMDS        ;Commands code now CMDCID2
C623:20 83 C8        130           JSR   ALTPRCNVENTRY   ;Call the protocol converter
C626:00              131           DFB   PCSTATUSCMD     ;Prot Conv status command
C627:C0 00           132           DW    ZP2CUSE         ;Pointer to call buffer
C629:C9 42           133           CMP   #ID2-$80        ;ApTalk ID status code 2??
C62B:D0 03    C630   134           BNE   NOTHISUNIT      ;Not ID2 then maybe last unit #
C62D:18              135           CLC                    ;If here then we've got it
C62E:80 04    C634   136           BRA   MAYBCONTINIT    ;See it we should cont INIT call

C630:          C630  138 NOTHISUNIT EQU  *
C630:C9 28           139           CMP   #NODEVCON-$80   ;Test for dev not connected
C632:D0 D5    C609   140           BNE   ARINIT2         ;C=1 if bad unit # tried
C634:          C634  141 MAYBCONTINIT EQU *              ;
C634:A5 C7           142           LDA   TESTTMP         ;0=init call—<>0=reboot call
C636:F0 01    C639   143           BEQ   ARINIT4         ;=0 then cont init call
C638:60              144           RTS                    ;V=1 return to reboot call

C639:          C639  146 ARINIT4   EQU   *               ;Continue init call here
C639:B0 28    C663   147           BCS   DOEXIT1         ;C=1 then AppleTalk unit not avail
C63B:A0 04           148           LDY   #CMDCINIT       ;Commands code for init
C63D:80 02    C641   149           BRA   ARINIT6         ;Skip ARSTATUS entry point


C63F:          C63F  151 ARSTATUS  EQU   *               ;Alt ROM status entry point
C63F:A0 05           152           LDY   #CMDCSTATUS     ;Command code for status


C641:          C641  154 ARINIT6   EQU   *               ;
C641:E6 C9           155           INC   ADDR0           ;Calculate user buffer
C643:D0 02    C647   156           BNE   STUPPTRS        ;C=0 then leave hi byte alone
C645:E6 CA           157           INC   ADDR1           ;
C647:          C647  158 STUPPTRS  EQU   *
C647:B2 C9           159           LDA   (ADDR0)         ;Get user byte
C649:85 C5           160           STA   BYTEUSER        ;Put in buffer
C64B:A5 C9           161           LDA   ADDR0           ;
C64D:85 C2           162           STA   PTRBUFF         ;Put in buffer
C64F:A5 CA           163           LDA   ADDR1           ;
C651:85 C3           164           STA   PTRBUFF+1       ;
C653:98              165           TYA                    ;Move commands code to 'A'
```

```
C654:          C654  167 CALLBOX   EQU   *
C654:20 6A C7        168           JSR   CALLSETUP      ;Setup some stuff for JSR
C657:20 83 C8        169           JSR   ALTPRCNVENTRY  ;Go to prot conv
C65A:00              170           DFB   PCSTATUSCMD    ;Prot Conv status command
C65B:C0 00           171           DW    ZP2CUSE        ;Pointer to buffer
```

```
C65D:          C65D  174 DOEXIT     EQU   *
C65D:F0 0B     C66A  175            BEQ   NECMDEXIT      ;=0 then no errors to report
C65F:C9 30           176            CMP   #$30           ;Less than $30 then make err4
C661:B0 02     C665  177            BCS   DOEXIT2        ;
C663:          C663  178 DOEXIT1    EQU   *
C663:A9 04           179            LDA   #NOUNIQUEID-$80-$30 ;Make no unique id error
C665:          C665  180 DOEXIT2    EQU   *
C665:29 0F           181            AND   #$F            ;Lo nibble has correct error code
C667:AA              182            TAX                   ;Error code must be in X
C668:D0 0B     C675  183            BNE   ECMDEXIT       ;<>= if errors
C66A:          C66A  184 NECMDEXIT  EQU   *
C66A:A2 00           185            LDX   #0             ;0= no error
C66C:18              186            CLC                   ;C=0 = no error
C66D:80 07     C676  187            BRA   CMDEXIT        ;Exit now

C66F:          C66F  189 ARAPTALK2  EQU   *
C66F:C9 81           190            CMP   #DIAGCMD       ;Is it a DIAG call?
C671:F0 17     C68A  191            BEQ   ARDIAG         ;If so go do it

C673:          C673  193 CMDEXITE   EQU   *
C673:          C674  194 FF         EQU   *+1
C673:A2 FF           195            LDX   #$FF           ;Illegal command error
C675:          C675  196 ECMDEXIT   EQU   *              ;Error command exit
C675:38              197            SEC                   ;Set carry for error
C676:          C676  198 CMDEXIT    EQU   *
C676:A0 F5           199            LDY   #$100-ZP2CUSELEN ;# of bytes to restore
C678:          C678  200 ARSTKRST   EQU   *
C678:68              201            PLA                   ;Recall value from stack
C679:99 CB FF        202            STA   ZP2CUSE-$100+ZP2CUSELEN,Y ;Store value back in zpage
C67C:C8              203            INY                   ;Next
C67D:D0 F9     C678  204            BNE   ARSTKRST       ;Loop til done

C67F:B8              206            CLV                   ;V=0 if from here
C680:68              207            PLA                   ;Modify entry status to reflect
C681:29 04           208            AND   #$04           ; correct exit status
C683:D0 01     C686  209            BNE   NOTACTIVE      ;<>0 =ints were off at entry

C685:58              211            CLI                   ;If here, ints were on at entry
C686:          C686  212 NOTACTIVE  EQU   *              ;
C686:8A              213            TXA                   ;Put error command in A
C687:4C 84 C7        214            JMP   MAINROMSW      ;Exit back to main ROM
```

```
C68A:          C68A  216 ARDIAG     EQU   *
C68A:C8              217             INY                    ;Move data to param buffer
C68B:B1 C9           218             LDA   (ADDRØ),Y        ;User buffer ptr lo byte
C68D:85 C2           219             STA   PTRBUFF          ;
C68F:C8              22Ø             INY                    ;
C69Ø:B1 C9           221             LDA   (ADDRØ),Y        ;User buffer ptr hi byte
C692:85 C3           222             STA   PTRBUFF+1        ;
C694:          C694  223 ARDIAG1    EQU   *
C694:C8              224             INY                    ;
C695:B1 C9           225             LDA   (ADDRØ),Y        ;User data
C697:99 C2 ØØ        226             STA   BYTELONUM-3,Y    ;Put in param buffer
C69A:C Ø Ø6          227             CPY   #6               ;Y=6 then done
C69C:DØ F6     C694  228             BNE   ARDIAG1          ;Loop til 4 moved

C69E:A9 Ø8           23Ø             LDA   #CMDCDIAG        ;Diag command code
C6AØ:2Ø 6A C7        231             JSR   CALLSETUP        ;Set some stuff for JSR


C6A3:          C6A3  233 ARDIAG2    EQU   *
C6A3:2Ø 83 C8        234             JSR   ALTPRCNVENTRY    ;Alt ROM prot conv entry point
C6A6:ØØ              235             DFB   PCSTATUSCMD      ;Prot Conv status command
C6A7:C Ø ØØ          236             DW    ZP2CUSE          ;Ptr to commands buffer
C6A9:F Ø Ø6    C6B1  237             BEQ   ARDIAG4          ;=Ø then not last packet sent
C6AB:C9 3F           238             CMP   #LASTPACKET-$8Ø  ;Hi bit not fed to us
C6AD:F Ø BB    C66A  239             BEQ   NECMDEXIT        ;=Ø then last packet received
C6AF:8Ø B2     C663  24Ø             BRA   DOEXIT1          ;Error command exit


C6B1:          C6B1  242 ARDIAG4    EQU   *                ;If here then more to come
C6B1:18              243             CLC                    ;Add # bytes read to buff ptr
C6B2:8A              244             TXA                    ;Lo byte to 'A'
C6B3:65 C2           245             ADC   PTRBUFF          ;
C6B5:85 C2           246             STA   PTRBUFF          ;Save for next call
C6B7:98              247             TYA                    ;Hi byte to add
C6B8:65 C3           248             ADC   PTRBUFF+1        ;
C6BA:85 C3           249             STA   PTRBUFF+1        ;Save for next call
C6BC:8Ø E5     C6A3  25Ø             BRA   ARDIAG2          ;Back for more
```

```
C6BE:          C6BE  253 ARREADPROT EQU  *              ;Read protocol entry point
C6BE:B8               254            CLV                ;Clear V if from here
C6BF:80 07     C6C8   255            BRA   ARREADREST2  ;Start into readrest routine

C6C1:          C6C1   257 ARREADPROT2 EQU *             ;
C6C1:A9 07            258            LDA   #CMDCREADPROT ;Readprot command code
C6C3:80 8F     C654   259            BRA   CALLBOX      ;Call box for execution


C6C5:          C6C5   261 ARREADREST EQU  *             ;AppleTalk readrest call entry
C6C5:2C 74 C6         262            BIT   FF           ;Sets V flag
C6C8:          C6C8   263 ARREADREST2 EQU *             ;AppleTalk readrest call entry
C6C8:85 C2            264            STA   PTRBUFF      ;'A' has lo byte of RAM ptr
C6CA:B1 C9            265            LDA   (ADDR0),Y    ;'A' has hi byte of RAM ptr
C6CC:85 C3            266            STA   PTRBUFF+1    ;
C6CE:C8               267            INY                ;Get/move 2 more bytes
C6CF:B1 C9            268            LDA   (ADDR0),Y    ;
C6D1:85 C5            269            STA   BYTEUSER     ;
C6D3:C8               270            INY                ;
C6D4:B1 C9            271            LDA   (ADDR0),Y    ;
C6D6:85 C6            272            STA   BYTEHINUM    ;
C6D8:50 E7     C6C1   273            BVC   ARREADPROT2  ;V=0 then exit here
C6DA:A9 06            274            LDA   #CMDCREADREST ;Readrest command code
C6DC:20 6A C7         275            JSR   CALLSETUP    ;Set up some stuff for JSR
C6DF:20 83 C8         276            JSR   ALTPRCNVENTRY ;Call the prot conv
C6E2:00               277            DFB   PCSTATUSCMD  ;Prot Conv status command
C6E3:C0 00            278            DW    ZP2CUSE      ;Buffer pointer
C6E5:48               279            PHA                ;Save for awhile
C6E6:5A               280            PHY                ;Save for awhile
C6E7:A0 04            281            LDY   #4           ;Put # of bytes read in user buff
C6E9:8A               282            TXA                ;Move to A to save
C6EA:91 C9            283            STA   (ADDR0),Y    ;
C6EC:68               284            PLA                ;Move hi byte too
C6ED:C8               285            INY                ;Next loc in user buff
C6EE:91 C9            286            STA   (ADDR0),Y    ;
C6F0:68               287            PLA                ;Restore error byte

C6F1:          C6F1   289 ARREXIT    EQU   *
C6F1:4C 5D C6         290            JMP   DOEXIT       ;
```

```
C6F4:                293 * Now move data to write into card.  The
C6F4:                294 * data is obtained from the table pointed
C6F4:                295 * to in the WRITE parameter list.
C6F4:                296 * The WRITE parameter list is set up as
C6F4:                297 * follows:

C6F4:                299 * WRITETBL EQU *
C6F4:                300 *   DW 1                  ;Length in bytes
C6F4:                301 *   DW addr of dest addr  ;Ptr to dest address
C6F4:                302 *   DW 1                  ;Length in bytes
C6F4:                303 *   DW addr of src addr   ;Ptr to Src address
C6F4:                304 *   DW 1                  ;Length in bytes
C6F4:                305 *   DW addr of LAP type   ;Ptr to LAP type
C6F4:                306 *   DW $bbbb              ;Length in bytes
C6F4:                307 *   DW addr of DDP data   ;Ptr to DDP data
C6F4:                308 *   DW $bbbb              ;Length in bytes
C6F4:                309 *   DW addr of ATP data   ;Ptr to ATP data
C6F4:                310 *   DW $bbbb              ;Length in bytes
C6F4:                311 *   DW addr of misc data  ;Ptr to misc data
C6F4:                312 *   DW $FFxx              ;Terminator <- REQUIRED

C6F4:         C6F4   314 ARWRITE   EQU   *              ;AppleTalk write call entry
C6F4:AA              315             TAX                ;Save in X
C6F5:B1 C9           316             LDA   (ADDR0),Y    ;Hi byte of user WRITETBL
C6F7:85 CA           317             STA   ADDR1        ;
C6F9:86 C9           318             STX   ADDR0        ;
C6FB:AA              319             TAX                ;Must save for later
C6FC:64 C7           320             STZ   TESTTMP      ;Sum of # bytes to send
C6FE:64 C8           321             STZ   TESTTMP+1    ;Hi byte of above
C700:A0 00           322             LDY   #0           ;Add together # bytes to
C702:         C702   323 SEND2MANYLP EQU *              ; send to see if too many
C702:B1 C9           324             LDA   (ADDR0),Y    ;Lo byte of # of bytes in buff
C704:65 C7           325             ADC   TESTTMP      ;Add to total
C706:85 C7           326             STA   TESTTMP      ;Update total
C708:C8              327             INY                ;
C709:B1 C9           328             LDA   (ADDR0),Y    ;Hi byte of # of bytes in buff
C70B:1A              329             INC                ;Sets Z if A was $FF (end)
C70C:F0 0E    C71C   330             BEQ   FOUNDEND     ;Off to brighter things
C70E:3A              331             DEC                ;Restore original number
C70F:65 C8           332             ADC   TESTTMP+1    ;Add to total
C711:85 C8           333             STA   TESTTMP+1    ;Update total
C713:C8              334             INY                ;Inc past buffer pointers
C714:C8              335             INY                ;
C715:C8              336             INY                ;Inc to lo byte of # bytes in buff
C716:D0 EA    C702   337             BNE   SEND2MANYLP  ;Loop if here
C718:E6 CA           338             INC   ADDR1        ;> then 255 buffers if here
C71A:80 E6    C702   339             BRA   SEND2MANYLP  ;

C71C:         C71C   341 FOUNDEND  EQU   *
C71C:86 CA           342             STX   ADDR1        ;Restore to it's orig value
C71E:A5 C8           343             LDA   TESTTMP+1    ;Do a quick chk for too many
C720:C9 03           344             CMP   #3           ;If hi byte is >=3 then too many
C722:90 05    C729   345             BCC   ITSHORTENUF  ;<3 then it's short enough
C724:A2 05           346             LDX   #BYTEGTR03-$80-$30 ;Error code
C726:4C 75 C6        347             JMP   ECMDEXIT     ;Error command exit

C729:         C729   349 ITSHORTENUF EQU *              ;If pkt len is OK come here
C729:A0 00           350             LDY   #0           ;Start back at 1st buffer
```

```
C72B:A9 04          351            LDA   #PCOUNTW      ;# of parameters for write call
C72D:20 6E C7       352            JSR   CALLSETUP2    ;Set up some stuff for JSR
C730:64 C6          353            STZ   TYPEWRITE     ;Data packet type is 0

C732:         C732  355 ARWRITE2   EQU   *
C732:B1 C9          356            LDA   (ADDR0),Y     ;Get lo byte of # bytes to send
C734:85 C4          357            STA   NUMLOWRITE    ;Put in buffer
C736:C8             358            INY                  ;
C737:B1 C9          359            LDA   (ADDR0),Y     ;
C739:C9 FF          360            CMP   #$FF          ;Terminator reached yet?
C73B:F0 1F    C75C  361            BEQ   SAYSENDIT     ;Yes then send 'send it' req
C73D:85 C5          362            STA   NUMHIWRITE    ;
C73F:C8             363            INY                  ;Put buffer ptrs in buff now
C740:B1 C9          364            LDA   (ADDR0),Y     ;
C742:85 C2          365            STA   PTRBUFF       ;
C744:C8             366            INY                  ;
C745:B1 C9          367            LDA   (ADDR0),Y     ;
C747:85 C3          368            STA   PTRBUFF+1     ;
C749:C8             369            INY                  ;Ready for next loop
C74A:D0 02    C74E  370            BNE   ARWRITE4      ;Skip inc
C74C:E6 CA          371            INC   ADDR1         ;For page cross
C74E:         C74E  372 ARWRITE4   EQU   *
C74E:84 C7          373            STY   TESTTMP       ;MUST preserve 'Y'
C750:20 83 C8       374            JSR   ALTPRCNVENTRY ;Call prot conv
C753:09             375            DFB   PCWRITECMD    ;Prot Conv write command
C754:C0 00          376            DW    ZP2CUSE       ;Buffer
C756:D0 99    C6F1  377            BNE   ARREXIT       ;Error then exit
C758:A4 C7          378            LDY   TESTTMP       ;Restore Y
C75A:80 D6    C732  379            BRA   ARWRITE2      ;Loop til done

C75C:         C75C  381 SAYSENDIT  EQU   *             ;If here last packet was sent
C75C:64 C4          382            STZ   NUMLOWRITE    ;0 out # of bytes this packet
C75E:64 C5          383            STZ   NUMHIWRITE    ;
C760:85 C6          384            STA   TYPEWRITE     ;<>0 means send Aptalk pkt
C762:20 83 C8       385            JSR   ALTPRCNVENTRY ;Call prot conv
C765:09             386            DFB   PCWRITECMD    ;Prot Conv write command
C766:C0 00          387            DW    ZP2CUSE       ;Buffer
C768:80 87    C6F1  388            BRA   ARREXIT       ;Return to user
```

```
C76A:          C76A  392 CALLSETUP EQU   *              ;Setup some stuff for Prot Conv
C76A:85 C4           393           STA   CODECMDS       ;Save cmd code for Prot Conv
C76C:A9 06           394           LDA   #PCOUNTS4.B    ;# of parameters for call

C76E:          C76E  396 CALLSETUP2 EQU  *              ;Alternate entry point
C76E:85 C0           397           STA   PARAMNUM       ;
C770:AD FE 07        398           LDA   APTALKUNIT     ;Move unit number to buff
C773:85 C1           399           STA   NUMUNIT        ;Put in buffer
C775:60              400           RTS                   ;Back to caller


C776:          C776  402 APTALKCMDS EQU  *
C776:04 C6           403           DW    ARINIT         ;AppleTalk init call
C778:C5 C6           404           DW    ARREADREST     ;AppleTalk readrest call
C77A:F4 C6           405           DW    ARWRITE        ;AppleTalk write call
C77C:3F C6           406           DW    ARSTATUS       ;AppleTalk status call
C77E:BE C6           407           DW    ARREADPROT     ;AppleTalk readprot call


C780:          0000  409 ROMSTUFFFILL EQU $C780-*
C780:          0000  410           DS    ROMSTUFFFILL,$FF ;Fill character


C780:                 28           LST   ASYM,VSYM      ;List by symbol and address
```

```
   C9 ADDR0          CA ADDR1          C883 ALTPRCNVENTRY C7D3 ALTROMSW
 C764 APOFFLOOP      C76F APOFFMSG     0011 APOFFMSGLN    C73F APPLETALK1
 C742 APPLETALK2     C776 APTALKCMDS   C74D APTALKOFFLN   07FE APTALKUNIT
 C5DB ARAPPLETALK2   C66F ARAPTALK2    C68A ARDIAG        C694 ARDIAG1
 C6A3 ARDIAG2        C6B1 ARDIAG4      C609 ARINIT2       C639 ARINIT4
 C604 ARINIT         C606 ARINIT1      C641 ARINIT6       C6BE ARREADPROT
 C6C1 ARREADPROT2    C6C8 ARREADREST2  C6C5 ARREADREST    C6F1 ARREXIT
 C63F ARSTATUS       C678 ARSTKRST     C5E3 ARSTKSVE      C732 ARWRITE2
 C74E ARWRITE4       C6F4 ARWRITE      C720 BASICENT        CB BOOTIT
 C76D BRAHANGLOOP      B5 BYTEGTR603     C6 BYTEHINUM       C5 BYTELONUM
   C5 BYTEUSER         00 C7FILL80     C654 CALLBOX       C76E CALLSETUP2
 C76A CALLSETUP        08 CMDCDIAG       0A CMDCID1         04 CMDCINIT
   07 CMDCREADPROT     06 CMDCREADREST   09 CMDCREBOOT      05 CMDCSTATUS
 C676 CMDEXIT        C673 CMDEXITE       00 CN20FILL        C4 CODECMDS
   81 DIAGCMD        C65D DOEXIT       C663 DOEXIT1       C665 DOEXIT2
 C675 ECMDEXIT       C674 FF           C748 FOS           C71C FOUNDEND
 C5A8 GETCODE2       C5BF GETCODE3     C5C5 GETCODE4      C5D5 GETCODE5
 FC58 HOME             C6 ID1            C2 ID2           FB2F INIT
 C729 ITSHORTENUF      39 KSWH           BF LASTPACKET    C784 MAINROMSW
 C634 MAYBCONTINIT   07F8 MSLOT        C66A NECMDEXIT       A8 NODEVCON
 C686 NOTACTIVE      C630 NOTHISUNIT     B4 NOUNIQUEID      C5 NUMHIWRITE
   C4 NUMLOWRITE       C1 NUMUNIT        C0 PARAMNUM      C71C PASERR
   06 PCOUNTS4.B       04 PCOUNTW        00 PCSTATUSCMD     09 PCWRITECMD
   C2 PTRBUFF        C715 REBOOTAPTALK   00 RELVERNUM       00 ROMSTUFFFILL
 C75C SAYSENDIT      047F SCRNHOLE0    077F SCRNHOLE1     07FF SCRNHOLE2
 0478 SCRNTMP0       C702 SEND2MANYLP  FE89 SETKBD        FE84 SETNORM
 FE93 SETVID         03F2 SOFTEV       C647 STUPPTRS        C7 TESTTMP
 C703 TOSETV           C6 TYPEWRITE      0B ZP2CUSELEN      C0 ZP2CUSE
   08 ZP8
```

```
   00 ROMSTUFFFILL     00 RELVERNUM        00 CN20FILL         00 PCSTATUSCMD
   00 C7FILL80         04 PCOUNTW          04 CMDCINIT         05 CMDCSTATUS
   06 CMDCREADREST     06 PCOUNTS4.B       07 CMDCREADPROT     08 ZP8
   08 CMDCDIAG         09 PCWRITECMD       09 CMDCREBOOT       0A CMDCID1
   0B ZP2CUSELEN     0011 APOFFMSGLN       39 KSWH             81 DIAGCMD
   A8 NODEVCON         B4 NOUNIQUEID       B5 BYTEGTR603       BF LASTPACKET
   C0 PARAMNUM         C0 ZP2CUSE          C1 NUMUNIT          C2 PTRBUFF
   C2 ID2              C4 CODECMDS         C4 NUMLOWRITE       C5 BYTEUSER
   C5 BYTELONUM        C5 NUMHIWRITE       C6 TYPEWRITE        C6 ID1
   C6 BYTEHINUM        C7 TESTTMP          C9 ADDR0            CA ADDR1
   CB BOOTIT         03F2 SOFTEV         0478 SCRNTMP0       047F SCRNHOLE0
 077F SCRNHOLE1      07F8 MSLOT          07FE APTALKUNIT     07FF SCRNHOLE2
 C5A8 GETCODE2       C5BF GETCODE3       C5C5 GETCODE4       C5D5 GETCODE5
 C5DB ARAPPLETALK2   C5E3 ARSTKSVE       C604 ARINIT         C606 ARINIT1
 C609 ARINIT2        C630 NOTHISUNIT     C634 MAYBCONTINIT   C639 ARINIT4
 C63F ARSTATUS       C641 ARINIT6        C647 STUPPTRS       C654 CALLBOX
 C65D DOEXIT         C663 DOEXIT1        C665 DOEXIT2        C66A NECMDEXIT
 C66F ARAPTALK2      C673 CMDEXITE       C674 FF             C675 ECMDEXIT
 C676 CMDEXIT        C678 ARSTKRST       C686 NOTACTIVE      C68A ARDIAG
 C694 ARDIAG1        C6A3 ARDIAG2        C6B1 ARDIAG4        C6BE ARREADPROT
 C6C1 ARREADPROT2    C6C5 ARREADREST     C6C8 ARREADREST2    C6F1 ARREXIT
 C6F4 ARWRITE        C702 SEND2MANYLP    C703 TOSETV         C715 REBOOTAPTALK
 C71C FOUNDEND       C71C PASERR         C720 BASICENT       C729 ITSHORTENUF
 C732 ARWRITE2       C73F APPLETALK1     C742 APPLETALK2     C748 FOS
 C74D APTALKOFFLN    C74E ARWRITE4       C75C SAYSENDIT      C764 APOFFLOOP
 C76A CALLSETUP      C76D BRAHANGLOOP    C76E CALLSETUP2     C76F APOFFMSG
 C776 APTALKCMDS     C784 MAINROMSW      C7D3 ALTROMSW       C883 ALTPRCNVENTRY
 FB2F INIT           FC58 HOME           FE84 SETNORM        FE89 SETKBD
 FE93 SETVID
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 15-JAN-84 21:28
** TOTAL LINES ASSEMBLED   738
** FREE SPACE PAGE COUNT   79
```

Index