

# United States Patent [19]

[11] 4,383,296

Sander

Best Available Copy

[45] May 10, 1983

[54] **COMPUTER WITH A MEMORY SYSTEM FOR REMAPPING A MEMORY HAVING TWO MEMORY OUTPUT BUSES FOR HIGH RESOLUTION DISPLAY WITH SCROLLING OF THE DISPLAYED CHARACTERS**

[75] Inventor: Wendell B. Sander, San Jose, Calif.

[73] Assignee: Apple Computer, Inc., Cupertino, Calif.

[21] Appl. No.: 150,630

[22] Filed: May 16, 1980

[51] Int. Cl.<sup>3</sup> ..... G06F 13/06; G06F 3/14

[52] U.S. Cl. .... 364/200; 340/726; 340/799

[58] Field of Search ... 364/200 MS File, 900 MS File; 340/726, 798, 799; 358/17

## [56] References Cited

### U.S. PATENT DOCUMENTS

3,821,730	6/1974	Carey et al. ....	340/799 X
3,893,075	7/1975	Orban et al. ....	340/799 X
3,903,510	9/1975	Zobel ....	340/726 X
3,980,992	9/1976	Levy et al. ....	364/200
4,136,359	1/1979	Wozniak ....	358/17
4,150,364	4/1979	Baltzer ....	364/900 X

## FOREIGN PATENT DOCUMENTS

1351590	5/1974	United Kingdom .
1482819	8/1977	United Kingdom .
1496563	12/1977	United Kingdom .
1524873	9/1978	United Kingdom .

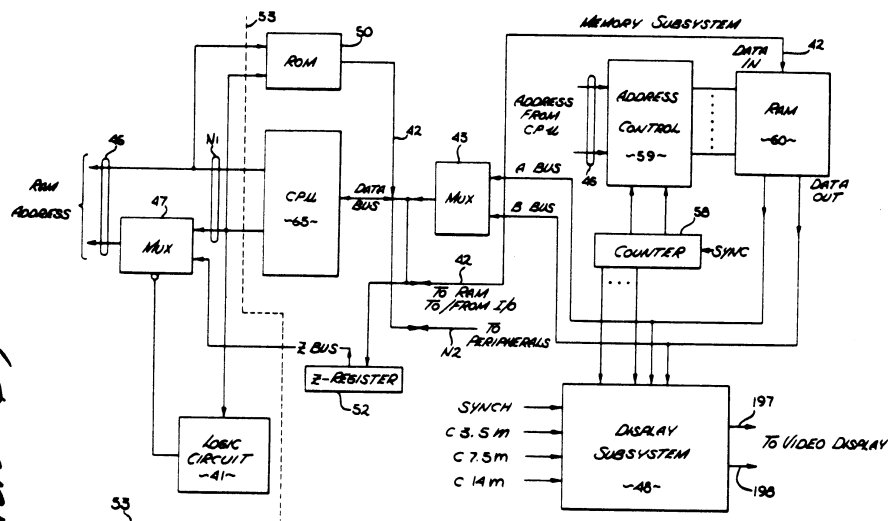
Primary Examiner—Raulfe B. Zache  
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

## [57] ABSTRACT

A microcomputer system with video display capability, particularly suited for small business applications and home use is described. The CPU performance is enhanced by permitting zero page data to be stored throughout the memory. The circuitry permitting this capability also provides a pointer for improved direct memory access. Through unique circuitry resembling "bank switching" improved memory mapping is obtained. Four-bit digital signals are converted to an AC chroma signal and a separate luminance signal for display modes. Display modes include high resolution modes, one of which displays 80 characters per line.

22 Claims, 9 Drawing Figures

CONTAINS BOOT ROM LISTING  
(REVISION D)



# Apple /// Computer

U.S. Patent May 10, 1983

Sheet 1 of 8

4,383,296

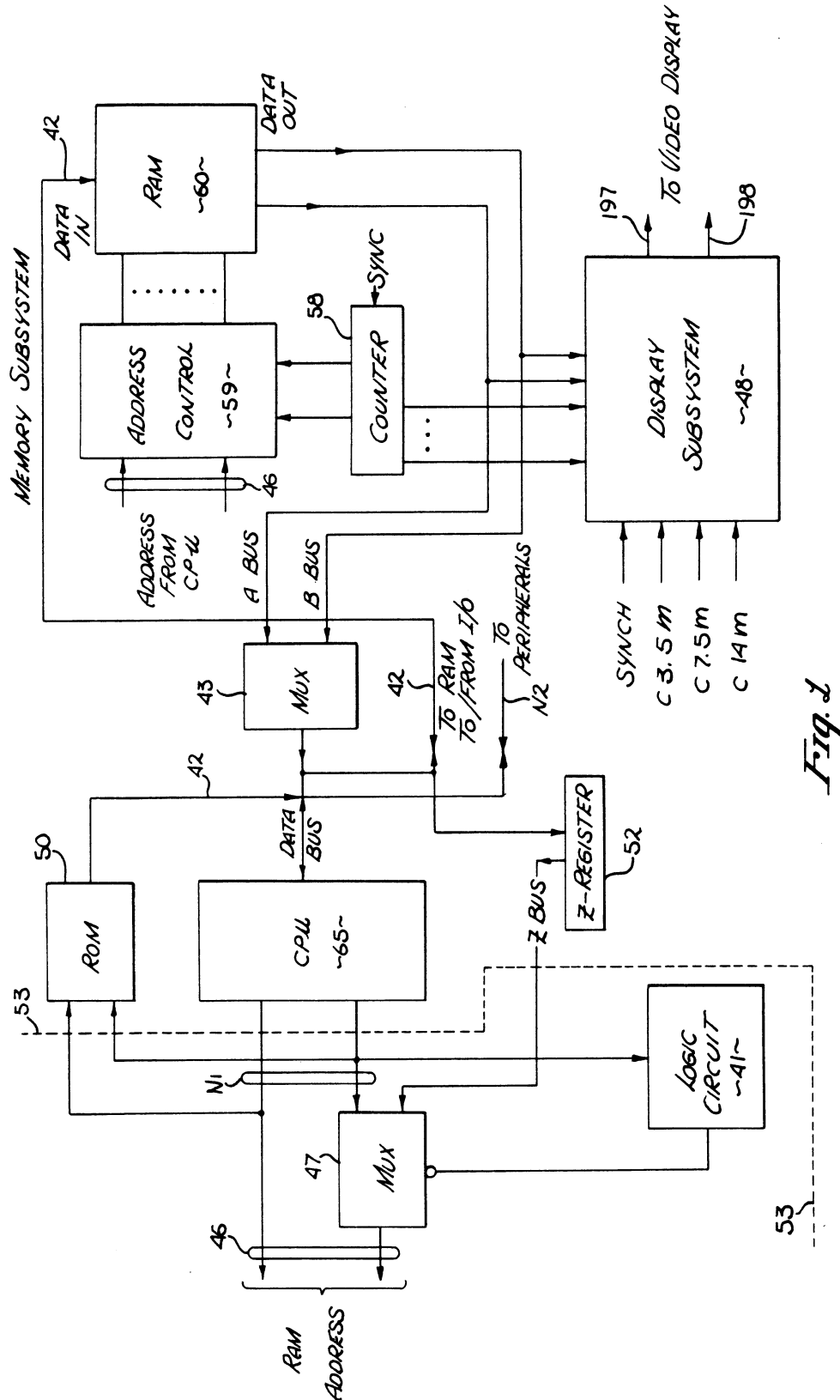
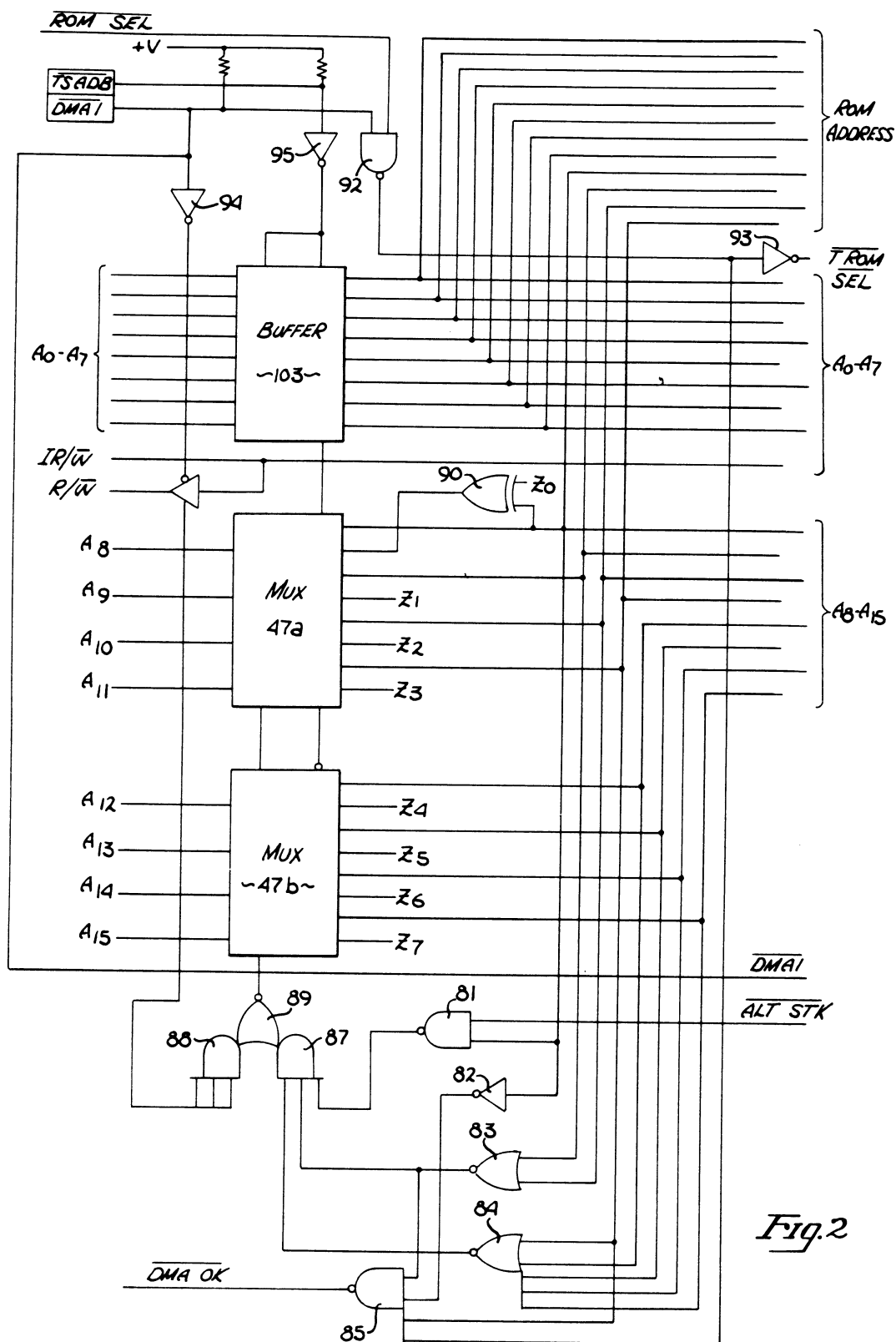


Fig. 1

**U.S. Patent**      **May 10, 1983**

Sheet 2 of 8

4,383,296



*Fig.2*

U.S. Patent May 10, 1983

Sheet 3 of 8

4,383,296

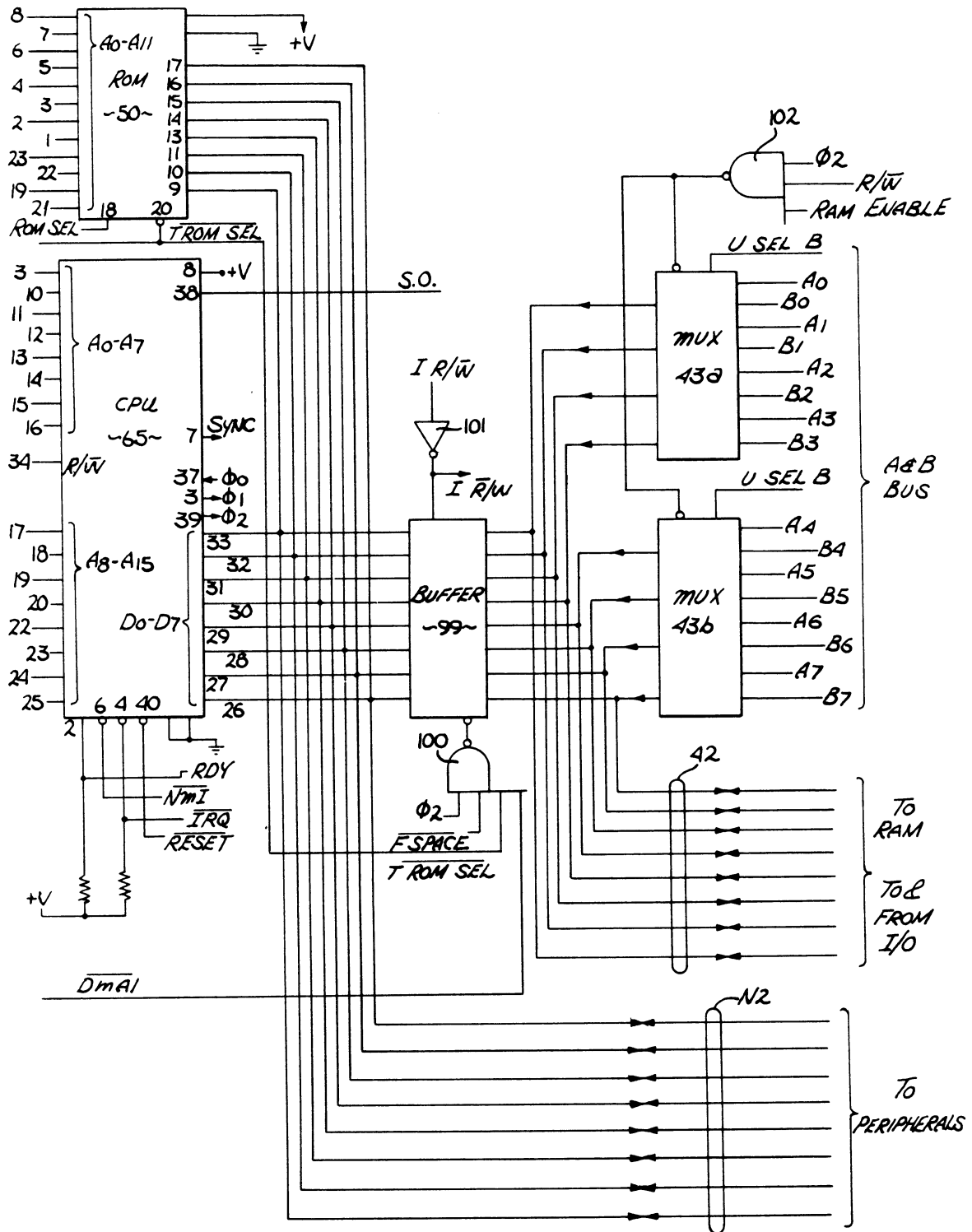


Fig. 3



U.S. Patent May 10, 1983

Sheet 4 of 8

4,383,296

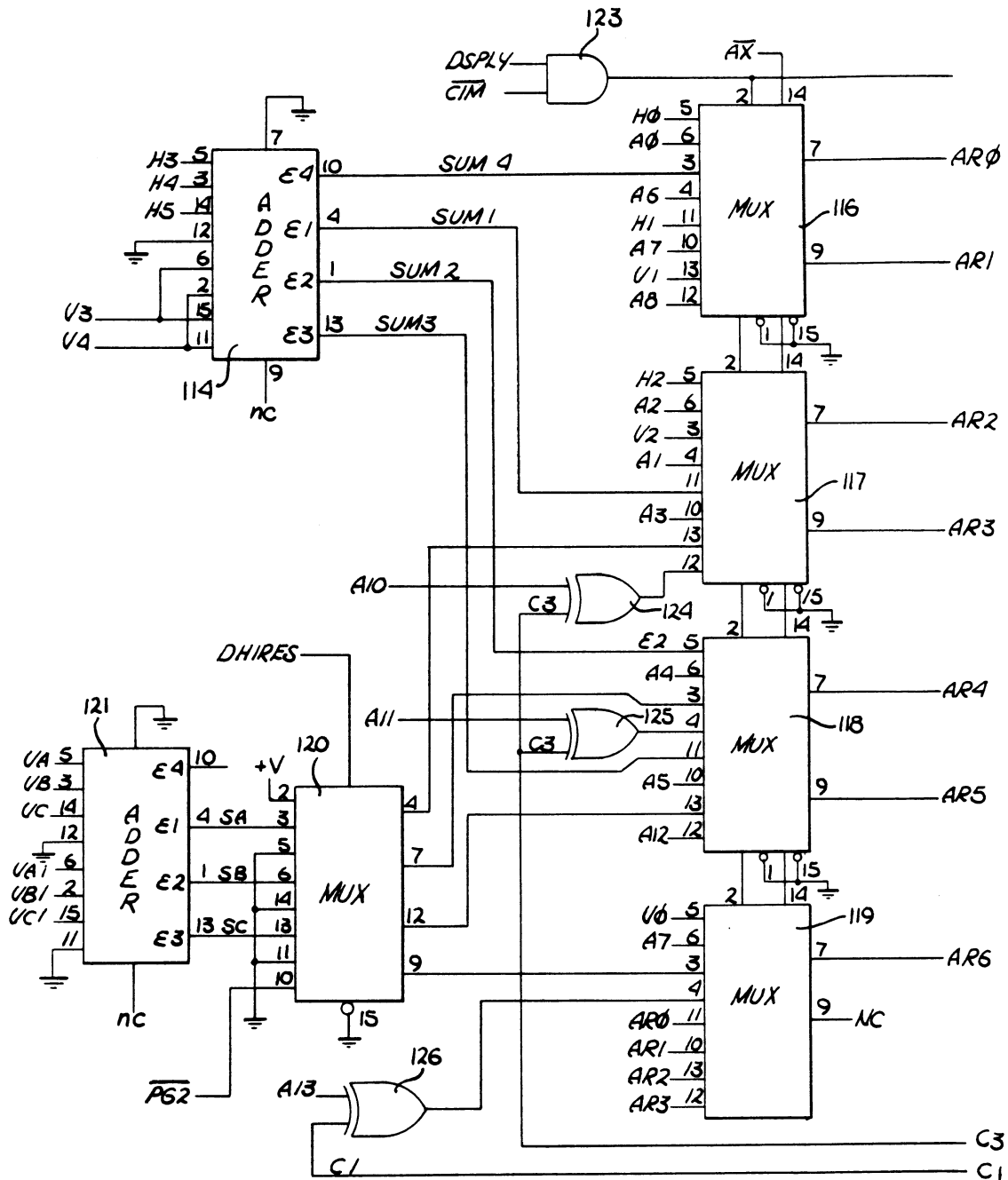


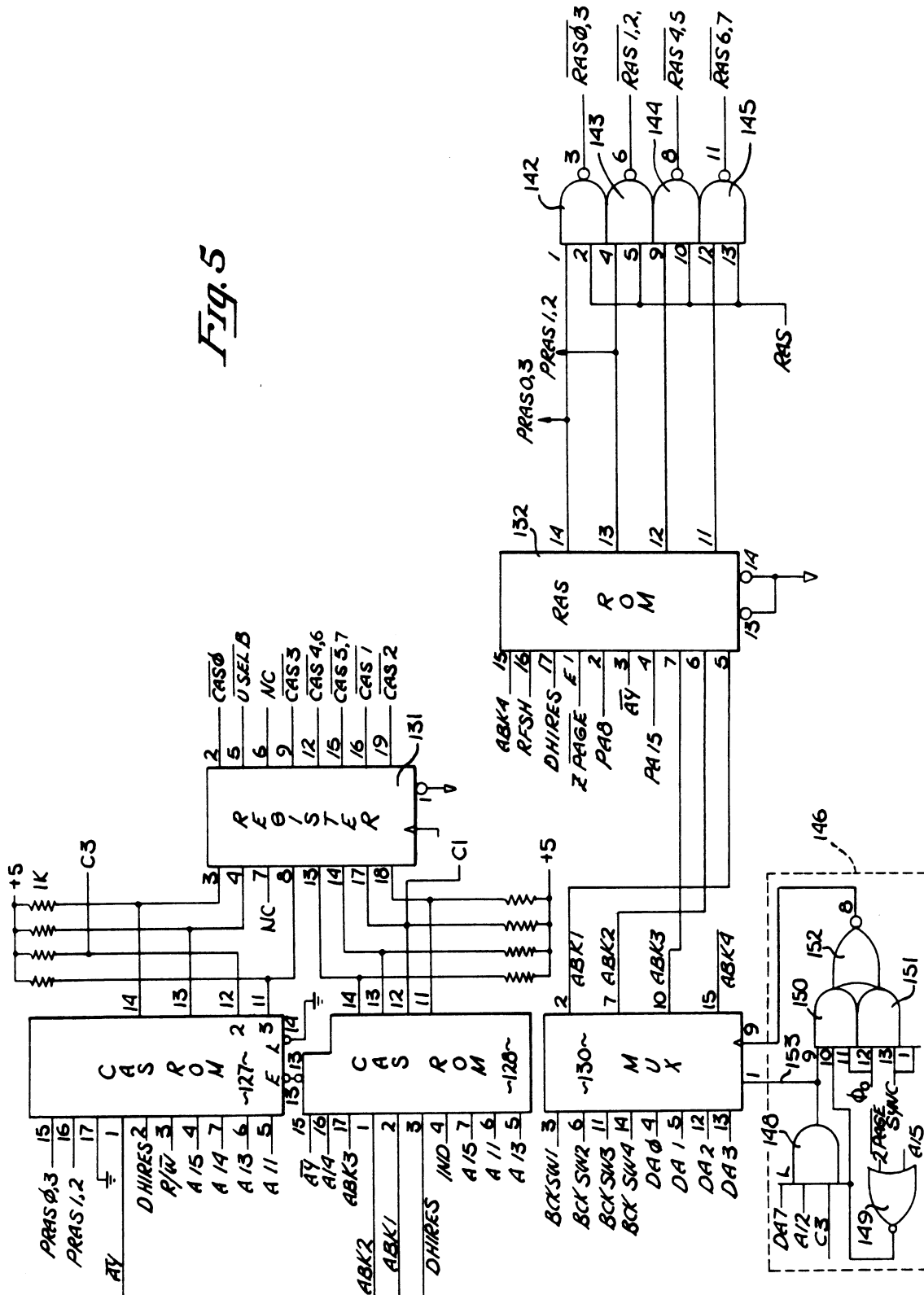
Fig. 4

U.S. Patent May 10, 1983

Sheet 5 of 8

4,383,296

Fig. 5



## U.S. Patent

Sheet 6 of 8

4,383,296

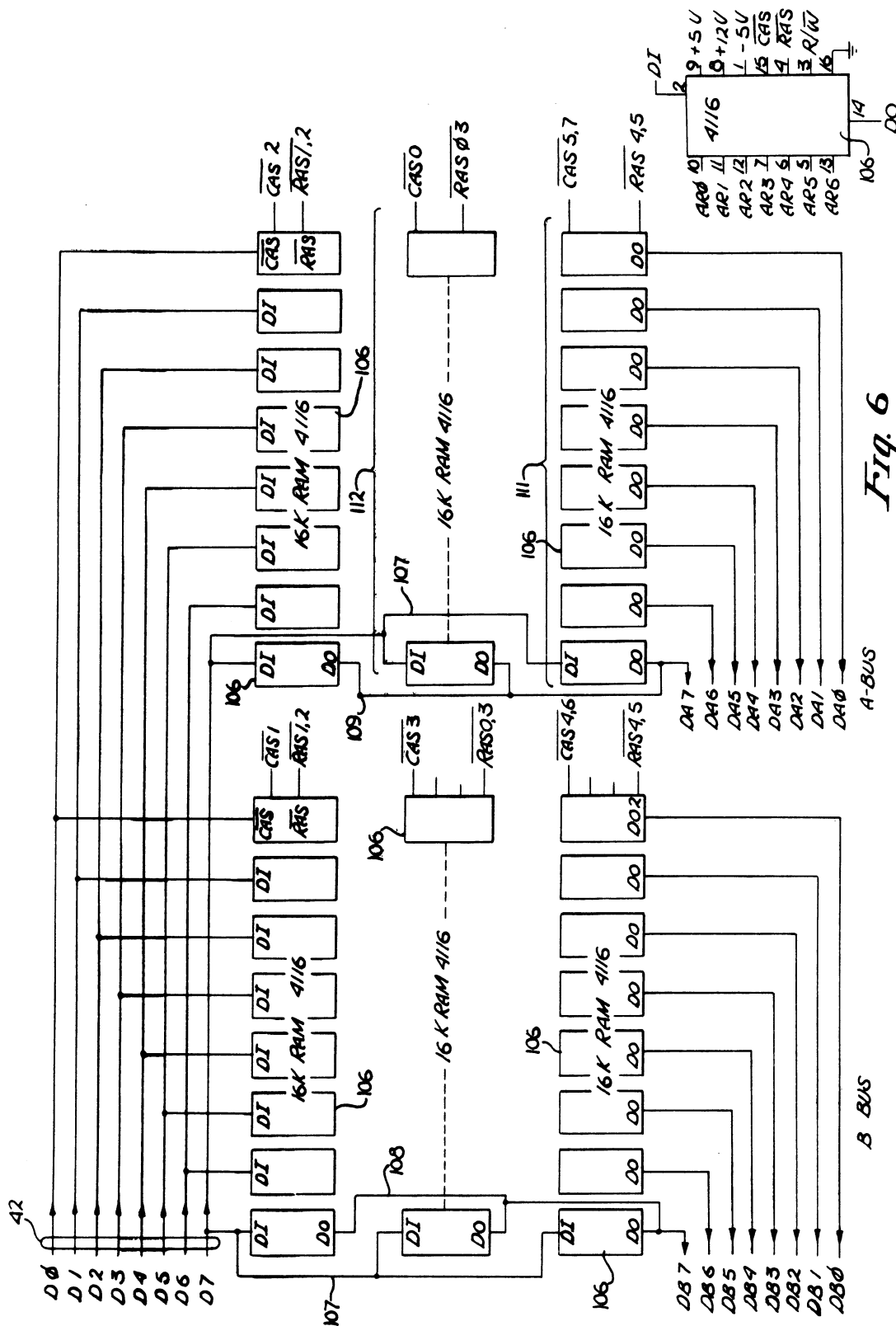


Fig. 6

**U.S. Patent**      **May 10, 1983**

Sheet 7 of 8

4,383,296

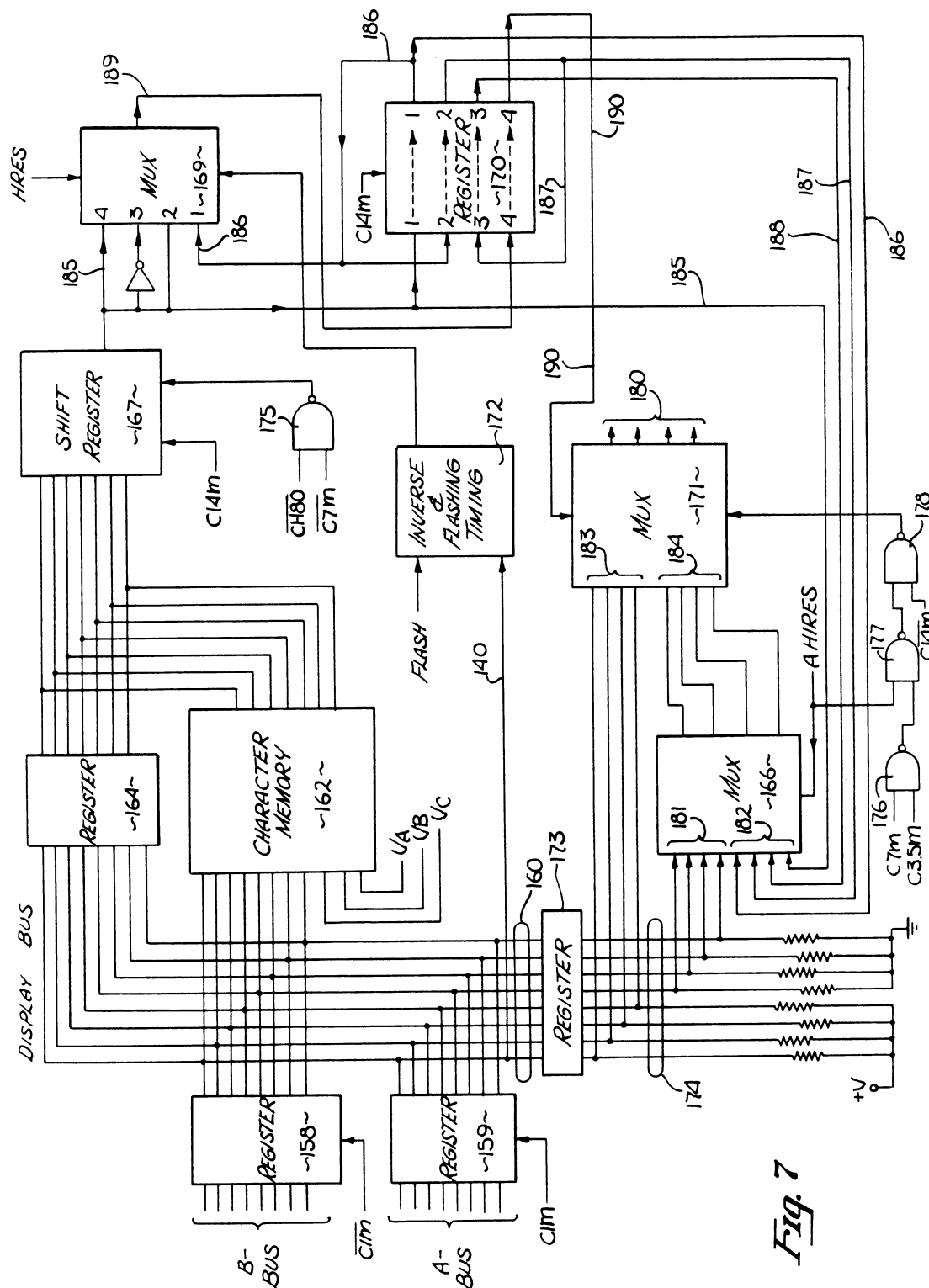


Fig. 2

U.S. Patent May 10, 1983

Sheet 8 of 8

4,383,296

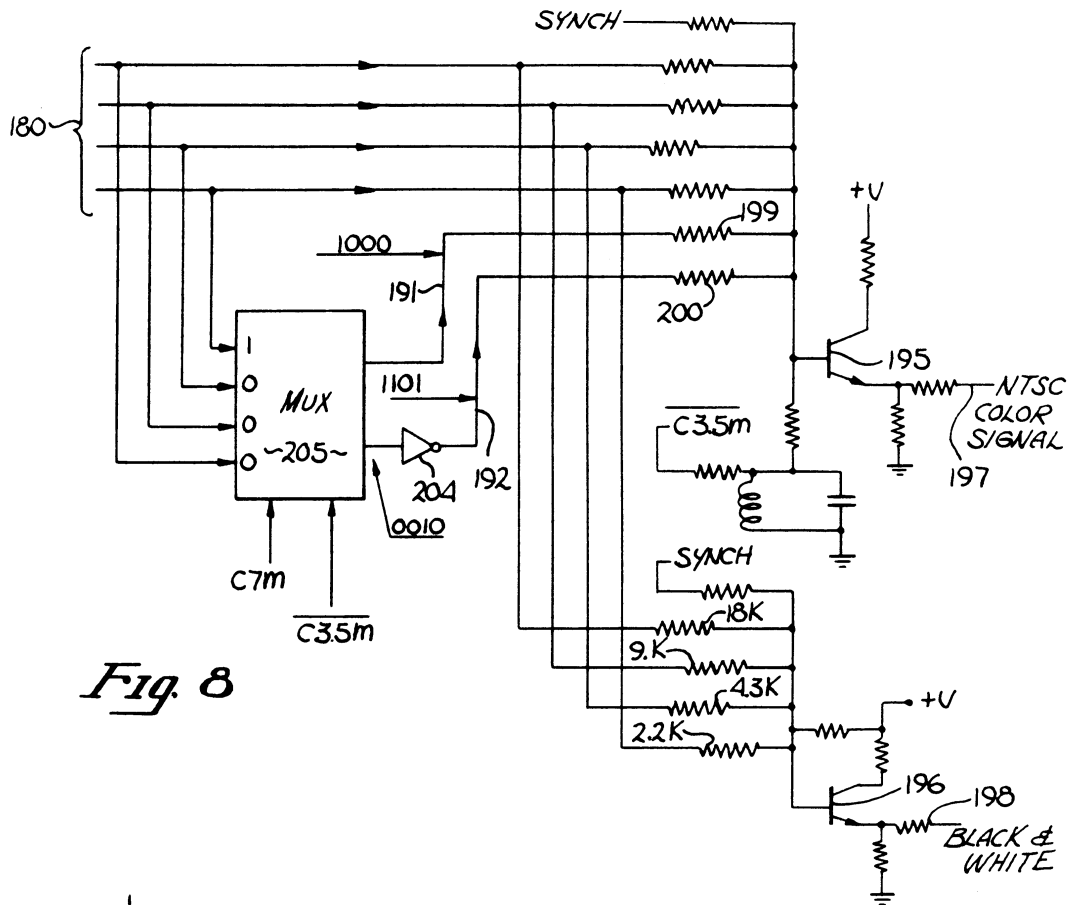


Fig. 8

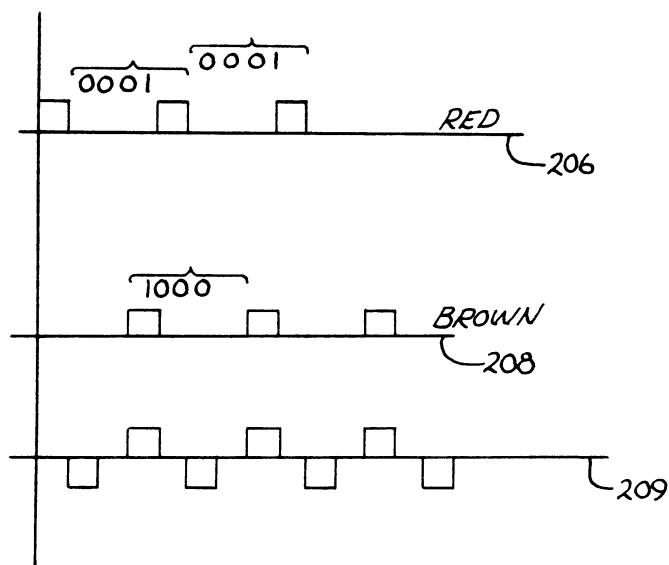


Fig. 9

4,383,296

1

# COMPUTER WITH A MEMORY SYSTEM FOR REMAPPING A MEMORY HAVING TWO MEMORY OUTPUT BUSES FOR HIGH RESOLUTION DISPLAY WITH SCROLLING OF THE DISPLAYED CHARACTERS

## BACKGROUND OF THE INVENTION

The invention relates to the field of digital computers, particularly microcomputers, having video display capabilities.

### Prior Art

In the last few years, there has been rapid growth in the use of digital computers in homes by hobbyists, for small business and for routine engineering and scientific application. For the most part, these needs have been met with self-contained, relatively inexpensive microcomputers or microprocessors with essential peripherals, including disc drives and with relatively easy to manage computer programs. The design for computers for these needs requires considerable ingenuity since each computer must meet a wide range of applications and because this market is particularly cost conscious.

A home or small business computer must, for example, operate with a number of different program languages, including those requiring relatively large memories, such as Pascal. The computer should interface with a standard raster scanned display and provide a wide range of display capabilities, such as high density alpha-numeric character displays needed for word processing in addition to high resolution graphics displays.

To meet these specialize computer needs, generally requires that a relatively inexpensive microprocessor be used and that the capability of the processor be enhanced through circuit techniques. This reduces the overall cost of the computer by reducing, for example, power needs, bus structures, etc. Another important consideration is that the new computers be capable of using programs developed for earlier models.

As will be seen, the presently described microcomputer is ideally suited for home and small business applications. It provides a wide range of capabilities including advanced display capabilities not found in comparable prior art computers.

The closest prior art computer known to applicant is commercially available under the trademark, Apple-II. Portions of that computer are described in U.S. Pat. No. 4,136,359.

## SUMMARY OF THE INVENTION

A digital computer which includes a central processing unit (CPU) and a random-access memory (RAM) with interconnecting address bus and data bus is described. One aspect of the present invention involves the increased capability of the CPU by allowing base page or zero page data to be stored throughout the memory. Alternate stack locations and an improved direct memory access capability are also provided by the same circuitry. Detection means are used for detecting a predetermined address range such as the zero page. This detection means causes a special register (Z-register) to be coupled into the address bus. The contents of this Z-register provide, for example, a pointer during direct memory access, or alternate stack locations for storing data normally stored on page one.

The memory of the invented computer is organized in an unusual manner to provide compatibility with the

2

8-bit data bus and yet provide high data rates (16-bits/MHz) needed for high resolution displays. A first plurality of memory devices are connected to a first memory output bus; these memory devices are also connected to the data bus. The memory includes a second plurality of memory devices which are also connected to the data bus; however, the outputs of these second devices are coupled to a second output memory bus. First switching means permit the first and second memory buses to be connected to the display for high data rate transfers. Second switching means permit either one of the memory buses to be connected to the data bus during non-display modes.

The addressing capability of the memory is greatly enhanced not only through bank switching, but through a novel remapping which does not require the CPU control associated with bank switching. In effect, the "unused" bits from one of the first and second memory buses are used for remapping purposes. This mode of operation is particularly useful for providing toggling between two separate portions of the memory.

The display subsystem of the described computer generates video color signal in a unique manner. A 4-bit color code as used in the prior art, is also used with the described display subsystem. However, this code is used to generate an AC chrominance signal and a separate DC luminance signal. This provides enhanced color capability over similar prior art color displays.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram showing the major components and subsystems of the invented and described microcomputer system.

FIGS. 2 and 3 together show the central processing unit (CPU) and the architecture associated with this CPU, particularly the address bus and data bus.

FIG. 2 is a circuit diagram primarily showing the address bus and the logic means associated with this bus.

FIG. 3 is a circuit diagram primarily showing the data bus and its interconnection with the memory buses (A bus and B bus), bootstrap read-only memory, and input/output ports.

FIGS. 4, 5 and 6 show the memory subsystem.

FIG. 4 is a circuit diagram primarily showing the circuitry for selecting between address signals from the address bus and display counter signals.

FIG. 5 is a circuit diagram primarily showing the generation of various "select" signals for the memory devices.

FIG. 6 is a circuit diagram showing the organization of the random-access memory and its interconnection with the data bus and memory output buses.

FIGS. 7 and 8 illustrate the display subsystem of the invented computer.

FIG. 7 is a circuit diagram showing the circuitry for generating the digital signals used for the video display.

FIG. 8 is a circuit diagram of the circuitry used to convert the digital signals to analog video signals.

FIG. 9 is a graph of several waveforms used to describe a prior art circuit and the circuit of FIG. 8.

## DETAILED DESCRIPTION OF THE INVENTION

A microcomputer system capable of driving a raster scanned video display is disclosed. In the following description, numerous specific details such as specific

4,383,296

3

part numbers, clock rates, etc., are set forth to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the inventive concepts described in this patent may be practiced without these specific details. In other instances, well-known circuits have been shown in block diagram form in order not to obscure the present invention in unnecessary detail.

Referring first to FIG. 1, in general the described computer includes a central processing unit (CPU) 65, its associated data bus 42, address bus 46, a memory subsystem and a display subsystem 58.

The address bus 46 from the CPU is coupled to the memory subsystem to permit the selection of locations in memory. Some of the address signals pass through a multiplexer 47. For some modes of operation, signals from a register 52 are coupled through the multiplexer 47 onto the bus 46. The register 52 is identified as the Z-register and is coupled to the multiplexer 47 by the Z bus. The general description of the multiplexer 47 and its control by the logic circuit 41 are described in detail in conjunction with FIG. 2. In general, the circuitry shown to the left of the dotted line 53 is included in FIG. 2 while the CPU 65, memory 50, data bus 42 and multiplexer 43 are shown in detail in FIG. 3.

The address bus N1 is coupled to the read-only memory 50. The output of this memory is coupled to the computer's data bus 42. The read-only memory (ROM) 50, as will be described, stores test routines, and other data of a general bootstrap nature for system initialization.

The data bus 42 couples data to the random-access memory (RAM) 60 and to and from I/O ports. This bus also couples data to the Z-register 52 and other commonly used registers not illustrated. The data bus 42 receives data from the RAM 60 through the A bus and B bus which are selected by multiplexer 43. The peripheral Bus N2 is used, as is better illustrated in FIG. 3, for coupling to peripherals.

The memory subsystem is shown in detail in FIGS. 4, 5 and 6. The address control means which receives addresses on bus 46, makes the final selection of memory locations within the RAM 60. Bank switching, addressing for display purposes, scrolling and other memory mapping is controlled by the address control means 59 as will be described in greater detail in conjunction with FIGS. 4 and 5. The PAM 60 is shown in detail in FIG. 6. The counter 58 which is synchronized with the horizontal and vertical display signals, provides signals both to the address control means 59 and to the display subsystem 48.

The display subsystem receives data from the RAM 60 on the A bus and B bus and converts these digital signals to video signals which control a standard raster scanned display. A standard NTSC color signal is generated on line 197 and a black and white video signal on line 198. The same signals used to generate these video signals can be used to generate separate red, green, blue (RGB) video signals. The display subsystem 48 receives numerous timing signals including the standard color reference signal shown as 3.5 MHz (C3.5M). This subsystem is described in detail in FIGS. 7 and 8.

#### COMPUTER ARCHITECTURE

In the presently preferred embodiment, the CPU 65 (microprocessor) employed with the described computer is a commercially available component, the 6502A. This 8-bit processor (8-bit data bus) which has a

4

16-bit address bus is shown in FIG. 3 with its interconnections to the remainder of the computer. The pin number for each interconnection is shown adjacent to the corresponding line. In many cases, the nomenclature associated with the 6502A (CPU 65) is used in this application. For example, pin 6 receives the nonmaskable interrupt signal (NMI), and pin 4 is coupled to receive the interrupt request signal (IRQ). Some of the signals employed with the CPU 65, which are well-known in the art, and which are not necessary for the understanding of the present invention are not described in detail in this application, such as the various synchronization signals and clocking signals. The address signals from the CPU 65 are identified as A<sub>0</sub>-A<sub>7</sub> and A<sub>8</sub>-A<sub>15</sub>. The data signals associated with the CPU 65 are shown as D<sub>0</sub>-D<sub>7</sub>. As will be apparent to one skilled in the art, the inventive concepts described in this application may be employed with other microprocessors.

Referring now to FIGS. 2 and 3, the general architecture, particularly the architecture associated with the CPU 65 can best be seen. The address signals A<sub>0</sub>-A<sub>7</sub> are coupled to a buffer 103 by the bus shown primarily in FIG. 2. These address signals are also coupled to the ROM 50. The signals A<sub>0</sub>-A<sub>7</sub> after passing through the buffer 103 are coupled to the memory subsystem. The address signals A<sub>8</sub>-A<sub>15</sub> (higher order address bits) are coupled through lines shown in FIG. 2 to the multiplexers 47a and 47b. The contents of the Z-register 52 of FIG. 1 is also connected to the multiplexers 47a and 47b through the Z-bus (Z<sub>1</sub>-Z<sub>7</sub>). The multiplexers 47a and 47b allow the selection of either the signals A<sub>8</sub>-A<sub>15</sub> from the CPU 65 or the contents of the Z-register (Z<sub>1</sub>-Z<sub>7</sub>) for addressing the RAM 60. The output of these multiplexers are shown as A<sub>8</sub>-A<sub>15</sub>; this designation is used even when the Z-bus is selected. Note in the case of the Z<sub>0</sub> signal, this signal is coupled to the multiplexer 47a through the exclusive OR gate 90 for reasons which are explained later. The address signals A<sub>8</sub>-A<sub>11</sub> are also coupled to the ROM 50, thus the signals A<sub>0</sub>-A<sub>11</sub> are used for addressing the ROM 50. The signals A<sub>8</sub>-A<sub>15</sub> are connected to the logic circuit shown in the lower left-hand corner of FIG. 2; this logic circuit corresponds to the logic circuit 41 of FIG. 1.

The input and output data signals from the CPU 65 are coupled by a bidirectional bus to the bidirectional buffer 99 (FIG. 3). This buffer is selectively disabled by gate 100 to allow the output of ROM 50 to be communicated to CPU 65 and during other times not pertinent to the present discussion. The direction of flow through the buffer 99 is controlled by a read/write signal coupled to the buffer through inverter 101. Data from the CPU 65 is coupled through the buffer 99 and bus 42 to the RAM 60 or to I/O ports. Data from the RAM 60 is communicated to CPU 65 or bus N2 from the A bus and B bus through the buffer 99. The 4 lines of the A bus and 4 lines of the B bus are coupled to the multiplexer 43a. Similarly, the other 4 lines of the A and B buses are coupled to the multiplexer 43b. Multiplexers 43a and 43b select the 8 lines of the A bus or B bus and communicate the data through to buffer 99 and bus 42. These multiplexers are selectively disabled (for example, during writing) by gate 102. As will be described later, the 16 lines of the A bus and B bus permits the reading of 16-bits from the RAM at one time. This provides a data rate of 16-bits/MHz which is necessary, for example, for an 80 character per line display. The data is loaded into the RAM 60, 8-bits at a time.

4,383,296

5

The ROM 50, as mentioned, stores test programs, data needed to initialize various registers, character generation data (for RAM 162 of FIG. 7) and other related data. Specific programs employed in the presently preferred embodiment of the computer are set forth in Table 1. The ROM 50 is selected by control signals coupled to its pins 18 and 20, identified as signals ROM SEL and TROM SEL. Any one of a plurality of commercially available read-only memories may be used for the ROM 50. In the presently preferred embodiment, commercially available Part No. SY2333 is used.

Referring now to this logic circuit (lower left-hand corner of FIG. 2), the NAND gate 81 receives the address signal A<sub>8</sub> and also the alternate stack signal identified as ALT STK. The output of this gate provides one input to the AND gate 87. The A<sub>8</sub> signal is also coupled through the inverter 82 to one input terminal of the NAND gates 85 and 86. The address signals A<sub>9</sub> and A<sub>10</sub> are coupled to the input terminals of the NOR gate 83. The output of this gate is coupled to one input terminal of the NAND gates 85 and 86 and the AND gate 87. The address signals A<sub>11</sub>-A<sub>15</sub> are coupled to the input terminals of the NOR gate 84. The signal A<sub>11</sub> is also coupled to an input terminal of the NAND gate 85.

The outputs of the AND gates 87 and 88 (through NOR gate 89), controls the multiplexers 47a and 47b. When the output of gate 89 is low the Z-bus is selected, otherwise the address signals from the CPU 65 are selected.

The logic circuit above-described, along with the Z-bus and Z-register provide enhanced performance for the computer. First, this circuit permits the zero page or base page data to be stored throughout the RAM 60 rather than just on zero page. Secondly, this circuit enables addressing of alternate stack locations (other than page one). Lastly, this circuit through the Z-register provides a RAM pointer for direct memory access (DMA).

Assume for purposes of discussion that the CPU 65 is addressing the zero page of memory. That is, the higher order address bits A<sub>8</sub>-A<sub>15</sub> are all zeros. The zeros for A<sub>9</sub>-A<sub>15</sub> are detected by the gates 83 and 84. If all the inputs to these gates are zeros, the outputs of these gates are high which condition is communicated to the gate 87. A<sub>8</sub> which is also low, insures that the output of gate 81 will be high. Thus, all the inputs to gate 87 are high, causing the signal at the output of the gate 89 to drop. When this occurs, the Z-bus is selected. Instead of all the binary zeros from the CPU being coupled to the main memory (RAM 60), the contents of the Z-register form part of the address for the memory. Therefore, even though the CPU 65 has selected the zero page, nonetheless data may be written into or from any location of RAM 60 (including the zero page). This enhances the performance of the CPU, since for example, the time consumed in shifting data to and from a single zero page is minimized.

Normally, the CPU 65 selects page one for stack locations. This occurs when A<sub>8</sub> is high and A<sub>9</sub>-A<sub>15</sub> are low. Assume first that the alternate stack locations have not been selected. Both inputs to gate 81 are high and its output is low. The low input to the gate 87 prevents the selection of the Z-bus. Thus, for these conditions the address signals A<sub>0</sub>-A<sub>7</sub> select stack locations on page one.

6

Next assume that page one has been selected by the CPU and that the ALT STK signal is low, indicating the alternate stack locations are to be selected. (A flag is set by the CPU to change the ALT STK signal). Since the ALT STK signal is low and A<sub>8</sub> is high, a high output occurs from the gate 81. All the inputs to gates 83 and 84 are low, therefore, high outputs occur from both these gates. The conditions of gate 87 are met, causing a high output from this gate and lowering the output from the gate 89. The Z-bus is thus selected by the multiplexers 47a and 47b. This allows the contents of the Z-register to be used as alternate locations. Non-zero page locations are assured by inverting A<sub>8</sub>. The exclusive OR gate 90 acts as a selective inverter. If A<sub>8</sub> is high and Z<sub>0</sub> is low, then A<sub>8</sub> at the output of the multiplexer 47a will be low. Note that during zero page selection when A<sub>8</sub> is low, the Z<sub>0</sub> signal is directly communicated through gate 90 to the output of multiplexer 47a.

Thus, the logic circuits along with the ALT STK signal allows alternate stack locations to be selected through the Z-bus. This further enhances the performance of the CPU which would otherwise be limited to page one for stack locations.

The logic circuit of FIG. 2 is also used along with the Z-register to provide a pointer during direct memory access (DMA). Assume that direct access to the computer's memory is required by a peripheral apparatus. To initiate the DMA mode the CPU provides an address between F800 and F8FF. Through a logic circuit not illustrated in FIGS. 2 and 3, the ROM SEL signal is brought low for addresses between F000 and FFFF. This signal is communicated to gate 93 and causes the output of gate 92 to rise (DMA I is high at this time). This rise in potential is communicated to one input of the gate 85. Additionally, gate 85 senses that the address bits A<sub>8</sub>, A<sub>9</sub> and A<sub>10</sub> are low. This information is coupled to gate 85 through the inverter 82 and the NOR gate 83 as high signals. Also the fact that A<sub>11</sub> is high is directly communicated to gate 85. Thus, with the address between F800 and F8FF the DMA OK signal drops in potential. This is sensed by the peripheral apparatus which in turn causes the DMA I signal to drop and provides a ready signal to the CPU 65. With the completion of this handshake, data may begin to be transferred to the RAM.

The DMA I signal through gate 93 and inverter 93 forces the TROM SEL signal low. This signal in addition to being communicated to the ROM 50, is coupled to the buffer 99 through gate 100, disabling this buffer (during the reading of ROM 50). Also, the ready signal causes the CPU to come to a hard stop. Importantly, the DMA I signal, after passing through the inverter 94 and the gates 88 and 89, assures the selection of the Z-register. The contents of the Z-register are fixed and provide a pointer to a page in the RAM.

Under the above conditions, the CPU increments the lower 8-bits of the address signal. The ROM 50 furnishes the instructions for incrementing the address, specifically SBC #1 and BEQ. The peripheral apparatus provides the data or receives the data in synchronization with the CPU operation. The peripheral also furnishes a read/write signal to indicate which operation is to occur. Data is then written into RAM via bus N2 and bus 42, or read from RAM via the A and B buses and bus N2.

Importantly, with the above DMA arrangement, addresses from the peripheral apparatus are not neces-



4,383,296

7

sary and the Z-register is used to provide a pointer to a page in RAM 60.

#### MEMORY SUBSYSTEM

The memory subsystem shown in FIG. 1 as the address control means 59 and RAM 60 is illustrated in detail in FIGS. 4, 5 and 6 as mentioned. In FIGS. 4 and 5, the memory control means is shown, while in FIG. 6 the memory devices and their organization are illustrated. The address control means of FIGS. 4 and 5 receives the address signals from the CPU 65 (A<sub>0</sub>-A<sub>15</sub>), the count in the vertical and horizontal counters (counter 58 of FIG. 1) which are used during display modes, control signals from the CPU and other signals. In general, this control means develops the address signals which are coupled to the RAM of FIG. 6 including the column address and row address signals, commonly referred to as  $\overline{\text{CAS}}$  and  $\overline{\text{RAS}}$ . Other related functions are also shown in FIGS. 4 and 5, such as the circuitry which provides display scrolling, indirect RAM addressing and memory mapping.

The CPU of FIG. 3 provides a 16-bit address for addressing the memory. Under ordinary circumstances this address limits the memory capacity to 64K bytes. This size memory is insufficient in many applications, as for example, to effectively use the Pascal program language. As will be described in greater detail, the address control means of FIGS. 4 and 5 enable the use of a memory having a 96K byte or 128K byte capacity. One well-known technique which is used with the present invention for increasing this capacity is bank switching; this switching occurs under the control of the CPU. In addition, the address control means uses a unique indirect addressing mode which provides the benefits of bank switching, however, this mode does not require CPU control. This greatly enhances CPU operation with the larger memory (as will be described) when compared to the CPU controlled bank switching.

Referring first to FIG. 6, the RAM configuration is illustrated for a capacity of 96K bytes. The memory is organized into six rows, each of which includes eight 16K memory devices such as rows 111 and 112. In the presently preferred embodiment, Part No. 4116 MOS dynamic RAMs are used. (The pin designations and signal designations refer to this memory device.) Obviously, other memory devices may be employed.

Input data to these memory devices 106 is provided from the bus 42. Each line in the bus 42 is connected to the data input terminal of one device 106 in each row. The interconnection of this bus with each of the memory devices is not shown in FIG. 6 in order to overcomplicate this drawing. By way of example, however, line 107 connects the data bit D7 to the data input terminal of one of the memory devices in each of the six rows.

Three rows of devices 106 have their output terminals coupled to the A bus, and three rows are similarly coupled to the B bus. By way of example, line 108 connects three output terminals of devices 106 to the DB7 line of the B bus while line 109 connects three output terminals of the devices 106 to the DA7 line of the A bus.

The described memory devices 106 are each organized as a 16KX1 memory. Thus, each device receives a 14-bit address which is time multiplexed into two, 7-bit addresses. This multiplexing occurs under the control of the  $\overline{\text{CAS}}$  and  $\overline{\text{RAS}}$  signals as is well-known. The lines coupling the address signals to each of the devices in FIG. 6 are not illustrated. However, in the

8

lower right-hand corner of FIG. 6, the various signals applied to each device (including the address signals), along with the corresponding pin numbers are shown. Other circuitry not illustrated is the refresh control circuitry which operates in a well-known manner in conjunction with the  $\overline{\text{CAS}}$ ,  $\overline{\text{RAS}}$  and address signals to refresh the dynamic devices.

Each row of memory devices 106 receives a unique combination of  $\overline{\text{CAS}}$  and  $\overline{\text{RAS}}$  signals. For example, row 111 receives  $\overline{\text{CAS}}$  5, 7 and  $\overline{\text{RAS}}$  4, 5; similarly, row 112 receives  $\overline{\text{CAS}}$  0 and  $\overline{\text{RAS}}$  0, 3. The generation of these  $\overline{\text{CAS}}$  and  $\overline{\text{RAS}}$  signals is described in conjunction with FIG. 5. These signals (along with the 14-bit address signals) permit the selection of a single 8-bit location in the 96K byte memory (for writing) and also the selection (for reading) of 16-bit locations.

The memory of FIG. 6 may be expanded to a 128K byte memory by using 32K memory devices, such as Part No. 4132. In this case, four rows of eight, 32K memory devices are used with each row receiving two  $\overline{\text{CAS}}$  and  $\overline{\text{RAS}}$  signals.

Before reviewing FIG. 4, a general understanding of the organization of the display is helpful. The display, during certain modes, is organized into 80 horizontal segments and 24 vertical segments for a total of 1920 blocks. 11-bits of the counter 58 of FIG. 1 are used as part of the address signals for the memory to access data for displaying during these modes. These counter signals are shown in FIG. 4 as H<sub>0</sub>-H<sub>5</sub> and V<sub>0</sub>-V<sub>4</sub>. During other display modes each horizontal segment is further divided into 8 segments (e.g. for displaying 80 alpha numeric characters per line). This requires 3 additional vertical timing signals shown as V<sub>A</sub>, V<sub>B</sub> and V<sub>C</sub> in FIGS. 4 and 7.

Often in the prior art, two separate counters are used to supply the timing/address signals for accessing a memory when the data in the memory is displayed. The count in one counter represents the horizontal lines of the screen (vertical count) and the other the position along each line, (horizontal or dot count). In many prior art displays the most significant bit of the dot counter is used to increment the line counter. Data in memory intended for display is mapped with a one-to-one correlation to the counts in these counters. In another prior art system (implemented in the Apple-II computer sold by Apple Computer, Inc.) this one-to-one correlation is not used. Rather, to conserve on circuitry, a single counter is employed and a more dispersed mapping is used in the memory. (Note that where a maximum horizontal count of 80 is used, this number cannot be represented by all ones in a digital counter and thus the vertical counter cannot easily be incremented by the most significant bit in the horizontal counter.) Since this more dispersed mapping technique is part of the prior art and not critical to an understanding of the present invention, it shall not be described in detail. However, the manner in which it is implemented shall be discussed in conjunction with the adder 114 of FIG. 4. For purposes of discussion, the signals from the counter 58 of FIG. 1 are designated as either vertical (V) or horizontal (H).

Referring now to FIG. 4, the selection of either the counter signals on the address signals from the CPU is made by the multiplexers 116, 117, 118 and 119. Each of these commercially available multiplexers (Part No. 153) couples one of four input lines to an output line. There are eight inputs to multiplexers 116, 117 and 118 and the outputs of these multiplexers provide the ad-

9

dress signals for the memories (AR0 through AR5). The multiplexer 119 has four inputs on its pins 3, 4, 5, 6 and provides a single output on pin 7, the AR6 address signal. (The signals supplied to pins 11, 12 and 13 of multiplexer 119 are for clamping purposes only.)

The AX signal is applied to the pin 14 of each of the multiplexers. The signal on this line and the signal applied to pin 2, determines which of the four inputs is coupled to each of the outputs of the multiplexers. The AX signal is a RAM timing signal for clocking the first 7 bits and second 7 bits of the multiplexed 14-bit address applied to each of the memory devices 106. The other control signal to the multiplexers is developed through the AND gate 123. The inputs to this gate are the display signal (DSPLY) which indicates that the computer is in a display mode and a clocking signal, specifically a 1MHz timing signal (C1M). The output of the AND gate 123 determines whether the address signals from the CPU or the signals associated with the counter 58 of FIG. 1 are selected.

Assume for purposes of discussion that the display has not been selected, and thus, the output of gate 123 is low. The AX signal then selects for pin 7 of multiplexer 116 first the address signal A<sub>0</sub> and then A<sub>6</sub>. Likewise, each of the multiplexers selects an address signal (except for those associated with exclusive OR gates 124 and 125 which shall be discussed). If the display signal is high and an output is present from the gate 123, then, by way of example, the AX signal first causes the H<sub>1</sub> signal and then the V<sub>1</sub> signal to be connected to the AR1 address line. Similarly, signals corresponding to the vertical and horizontal count are coupled to the other address lines during display modes.

The adder 114 is an ordinary digital adder for adding two 4-bit digital nibbles and for providing a digital sum signal. A commercially available adder (Part No. 283) is employed. The carry-in terminal (pin 7) is grounded and no carry-outs occur since one of the inputs (pin 12) is grounded. The adder sums the digital signal corresponding to H<sub>3</sub>, H<sub>4</sub> and H<sub>5</sub> with the digital signal corresponding to V<sub>3</sub>, V<sub>4</sub>, V<sub>5</sub>, V<sub>6</sub>. The resultant sum signal is coupled to the multiplexers 116, 117 and 118 as illustrated. the summing of these horizontal and vertical counter signals is used to provide the more dispersed mapping as previously discussed.

The adder 121 is identical to adder 114 and is coupled to sum the three least significant vertical counter bits from the counter 58 (FIG. 2) with the signals VA1, VB1 and VC1. The sum is selected by the multiplexer 120 during the high resolution display modes and also during scrolling as will be described. These sum signals are coupled to the multiplexers 117, 118 and 119. During the low resolution display modes, the multiplexer 120 couples ground signals or the page 2 signal (PG2) to the multiplexers 117, 118 and 119. (The PG2 signal is used for special mapping purposes, not pertinent to the present invention.) During the high resolution modes when the display is not being scrolled, the VA1, VB2 and VB3 signals are at ground potential and thus no summing occurs within adder 121 and the VA, VB and VC signals are coupled directly to the multiplexers 117, 118 and 119.

The address signals A<sub>10</sub>, A<sub>11</sub>, and A<sub>13</sub> from the CPU are coupled to the multiplexers 117, 118 and 119, respectively, through exclusive OR gates 124, 125, and 126, respectively. The other input terminals to gates 124 and 125 receive the C<sub>3</sub> signal, while the other input terminal of the gate 126 receives the C<sub>1</sub> signal. (The

4,383,296

10

development of the C<sub>1</sub> and C<sub>3</sub> signals is illustrated in FIG. 5.) The gates 124, 125 and 126 provide mapping compensation within the memory. As the computer and memory are presently implemented, the sequence in which the various portions of the display are generated is not the same as the sequence in which the data is removed from memory for display. These gates provide compensating addresses and, in effect, cause a remapping so that the proper sequence is maintained when data is read from the memory for the display. These gates are shown to provide a complete disclosure of the presently preferred embodiment, however, they are not critical to the present invention.

In operation, the circuitry of FIG. 4, as mentioned, selects the address signals which are applied to each of the memory devices, either from the CPU or counter if the display mode is selected. It should be noted that not all of the address bits from the CPU are coupled to the multiplexers 116 through 119. Some of these address bits, as will be described in conjunction with FIG. 5, are used to develop the various CAS and RAS signals and thus select different rows within the memory of FIG. 6.

The scrolling operation which is used is somewhat unusual in that each line of the display is separately moved up (line-by-line) with one line of data in memory being moved for each frame. This technique provides a uniform, esthetically pleasing, scroll. Scrolling the screen one line per frame can be achieved by moving all the data in the memory into a new position for each frame. This would be very time consuming and impractical. With the described technique, only one-eighth of the data in the memory is moved for each new frame.

Referring to the adder 121, as mentioned, the signals V<sub>A</sub>, V<sub>B</sub>, V<sub>C</sub> are the three least significant vertical counter bits from the counter 58. These bits or counts, by way of example, represent the 8 horizontal lines of each character. In adder 12, a 3-bit digital signal, VA1, VB1 and VC1, is added to the count from counter 58. This 3-bit signal is constant during each frame, however, it is incremented for each new frame.

During a first frame, 000 is added to the vertical count. During a second frame, 001 is added; and during a third frame, 010 is added, and so on. By adding this digital signal to the count from counter 58, the addresses to the memory are changed in the vertical sense. During the first frame when 000 is added, the display remains unaffected. During the next frame, when 001 is added to the vertical count, instead of first displaying the first line of a character, the second line of each character is displayed at the top of each character space and each subsequent line of the character is likewise moved up one line. If data in memory is not moved, the first line of the character would appear at the bottom of each character. Note when 001 is added to 111 from the counter, 000 results. Thus, the first line of characters would be addressed when the beam is scanning the eighth line of characters. To prevent this, the data corresponding to the first line of each character is moved in memory for this frame. The first line of one character is moved up and becomes the bottom line of the character directly above it. When 010 is added, the process is again repeated. For example, the third line of each character is first displayed in each character space and the second line of each character is moved up to become the bottom line of the character directly above it. This process is repeated to scroll the data. The movement of data in memory is controlled by the CPU in a well-known manner.

4,383,296

11

Thus, through use of adder 121, an even, continuous scroll is obtained without moving all the data in memory for each frame. Rather, only 1/8th of the data is moved for each frame.

Referring now to FIG. 5, the circuitry used to extend the addressing from the CPU is illustrated. In general, the CAS signals are generated by the ROMs 127 and 128. The RAS signals are generated by the ROM 132. The multiplexer 130 allows the selection of either the bank switching signals, or the unique indirect addressing mode when "bank switching" occurs without direct commands from the CPU.

The CAS ROM 127 receives as an address the following signals: PRAS $\phi$ 3, PRAS 1,2 AY, DHIREs, R/W, A<sub>11</sub>, A<sub>13</sub>, A<sub>14</sub>, and A<sub>15</sub>. As the PRAS $\phi$ 3 and PRAS 1, 2 represent the RAS signals being used. These signals are high when the respective RAS signal is active.

As previously mentioned, the AY signal is high for display modes and the DHIREs signal is high for high resolution display modes. The CAS ROM 128 receives as address signals the ABK1, ABK2, and ABK3 signals and also DHIREs, AY, IND, A<sub>11</sub>, A<sub>13</sub>, A<sub>14</sub>, and A<sub>15</sub>.

The ROMs 127 and 128 are programmed to implement the following equations.

$$\overline{\text{PCAS0}} = (\text{PRAS0.3} \cdot (\text{DHIREs} \cdot \overline{\text{AY}} + \text{AY} \cdot (\overline{\text{A15}} \cdot \overline{\text{A14}} \cdot \overline{\text{A13}} \cdot \text{R/WN} + \overline{\text{A15}} \cdot \overline{\text{A14}} \cdot \text{A13} \cdot \text{R/WN} + \overline{\text{A15}} \cdot \text{A14} \cdot \overline{\text{A13}} \cdot \overline{\text{A11}}))) \quad (1)$$

$$\overline{\text{PCAS2}} = (\text{DHIREs} \cdot \overline{\text{AY}} + \text{AY} \cdot (\overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}} \cdot \overline{\text{IND}} + \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \text{ABK3} \cdot (\overline{\text{A15}} \cdot \overline{\text{A14}} + \text{AY} \cdot \text{IND} \cdot \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}} \cdot \overline{\text{A15}} \cdot (\overline{\text{A14}} \cdot \overline{\text{A13}} + \text{A14} \cdot \overline{\text{A13}}))) \quad (2)$$

$$\overline{\text{PCAS3}} = \text{PRAS0.3} \cdot (\text{DHIREs} \cdot \overline{\text{AY}} + \text{AY} \cdot (\overline{\text{A15}} \cdot \overline{\text{A14}} \cdot \overline{\text{A13}} \cdot \text{A11} + \overline{\text{A15}} \cdot \overline{\text{A14}} \cdot \overline{\text{A13}} \cdot \overline{\text{A11}} + \overline{\text{A15}} \cdot \text{A14} \cdot \overline{\text{A13}} \cdot \overline{\text{A11}}))) \quad (3)$$

$$\overline{\text{PCAS4.6}} = (\text{AY} \cdot \text{IND} \cdot \overline{\text{ABK3}} \cdot \overline{\text{A15}} \cdot (\overline{\text{ABK1}} \cdot \overline{\text{ABK2}} + \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{A14}} \cdot \overline{\text{A13}} + \text{A14} \cdot \overline{\text{A13}}) + \text{AY} \cdot \text{IND} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{ABK2}} \cdot \overline{\text{ABK1}} \cdot \text{A15} + \overline{\text{ABK2}} \cdot \overline{\text{ABK1}} \cdot \overline{\text{A15}} \cdot \overline{\text{A14}} + \text{AY} \cdot \text{IND} \cdot \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{A15}} \cdot \overline{\text{A14}} + \text{AY} \cdot \text{IND} \cdot \overline{\text{ABK3}} \cdot \overline{\text{ABK2}} \cdot (\overline{\text{A15}} \cdot \overline{\text{ABK1}} + \text{A15} \cdot \overline{\text{ABK1}}) \cdot (\overline{\text{A14}} \cdot \overline{\text{A13}} + \text{A14} \cdot \overline{\text{A13}}))) \quad (4)$$

$$\overline{\text{PCAS5.7}} = (\text{AY} \cdot \text{IND} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{ABK1}} \cdot \overline{\text{ABK2}} + \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{A15}} \cdot \overline{\text{A14}} \cdot \overline{\text{A13}} + \text{A15} \cdot \overline{\text{A14}} \cdot \overline{\text{A13}}) + \text{AY} \cdot \text{IND} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{ABK2}} \cdot \overline{\text{ABK1}} \cdot \text{A15} + \overline{\text{ABK2}} \cdot \overline{\text{ABK1}} \cdot \overline{\text{A15}} \cdot \overline{\text{A14}} + \text{AY} \cdot \text{IND} \cdot \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{A15}} \cdot \overline{\text{A14}} + \text{AY} \cdot \text{IND} \cdot \overline{\text{ABK3}} \cdot \overline{\text{ABK2}} \cdot (\overline{\text{A15}} \cdot \overline{\text{ABK1}} + \text{A15} \cdot \overline{\text{ABK1}}) \cdot (\overline{\text{A14}} \cdot \overline{\text{A13}} + \text{A14} \cdot \overline{\text{A13}}))) \quad (5)$$

In effect, these ROMs are programmed to allow selection of predetermined rows in the memory, based on the address signals A<sub>10</sub>, A<sub>13</sub>, A<sub>14</sub> and A<sub>15</sub>, (ignoring for a moment the contribution of the RAS signals and the other signals appearing in the equations).

The outputs of the CAS ROMs 127 and 128 are coupled to the register 131. Register 131 is a commercially available register which permits the enabling of output signals (Part No. 374). During accessing of the memory the various CAS signals (CAS 0 through CAS 7) are coupled to the memory of FIG. 6 to permit selection of the appropriate memory devices. The signal USELB from CAS ROM 127 through register 131 selects either the A bus or B bus. This signal is coupled to the multiplexers 43a and 43b of FIG. 3.

During normal operation, the multiplexer 130 selects the bank switching signals BCKSW 1 through BCKSW

12

4. These four signals (or alternatively four signals from the A bus) provide four of the inputs (address signals) to the ROM 132. The other inputs to this ROM are the DHIREs, Z PAGE, PA8, PA15, RFSH (refresh), and AY signals. These address signals select the RAS 0, 3; RAS 1, 2; RAS 4, 5 and RAS 6, 7 signals. The ROM 132 is programmed to implement the following four equations.

$$\text{PRAS0.3} = \overline{\text{AY}} \cdot (\text{DHIREs} + \text{RFSH}) + (\text{ABK4} \cdot (\text{Z PAGE} \cdot \overline{\text{PA8}}) + \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}}) \cdot \text{AY} \quad (6)$$

$$\text{PRAS1.2} = \overline{\text{AY}} \cdot (\text{DHIREs} + \text{RFSH}) + \text{AY} \cdot (\overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}} \cdot (\text{ABK4} \cdot (\text{Z PAGE} \cdot \overline{\text{PA8}}) \cdot \overline{\text{PA15}} + \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}}) + \text{AY} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK4}} \cdot (\text{Z PAGE} \cdot \overline{\text{PA8}}) \cdot \overline{\text{PA15}} + \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK4}} \cdot (\text{Z PAGE} \cdot \overline{\text{PA8}}) \cdot \overline{\text{PA15}})) \quad (7)$$

$$\text{PRAS4.5} = \text{RFSH} \cdot \overline{\text{AY}} + \text{AY} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{ABK1}} \cdot \overline{\text{ABK4}} \cdot (\text{Z PAGE} \cdot \overline{\text{PA8}}) \cdot \overline{\text{PA15}} + \overline{\text{ABK1}} \cdot \overline{\text{ABK4}} \cdot (\text{Z PAGE} \cdot \overline{\text{PA8}}) \cdot \overline{\text{PA15}})) \quad (8)$$

$$\text{PRAS6.7} = \text{RFSH} \cdot \overline{\text{AY}} + \text{AY} \cdot \overline{\text{ABK3}} \cdot (\overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK4}} \cdot (\text{Z PAGE} \cdot \overline{\text{PA8}}) \cdot \overline{\text{PA15}} + \overline{\text{ABK1}} \cdot \overline{\text{ABK2}} \cdot \overline{\text{ABK4}} \cdot (\text{Z PAGE} \cdot \overline{\text{PA8}}) \cdot \overline{\text{PA15}}))$$

Thus, the bank switching signals (along with the other input signals to ROM 132) select predetermined rows in memory in conjunction with the CAS signals.

The output signals of the ROM 132 are coupled through the NAND gates 142, 143, 144 and 145 to the memory. The other input terminals of these gates receive the RAS timing signal. In this manner, the output signals of the ROM 132 are clocked through the gates 142 through 145 to provide the RAS signals shown in FIGS. 5 and 6.

An important feature to the presently described computer is provided by the circuitry shown within the dotted line 146. The AND gate 148 receives, at its input terminals, the DA7, A<sub>12</sub>, and C<sub>3</sub> signals. The NOR gate 149 receives the zero page and A<sub>15</sub> signal. The output of gate 149 provides one input to the gate 148 and also one input to the AND gate 150. The output of gate 148 provides another input signal to gate 150 and this signal (line 153) is one of the two control signals coupled to the multiplexer 130. The AND gates 150 and 151 also receive a SYNC signal and the  $\phi_0$  signal. The output of the gates 150 and 151 are coupled to a NOR gate 152 with the output of the gate 152 (line 154) coupled to the other control terminal of the multiplexer 130.

The gates 150, 151 and 152 effectively form a clock for multiplexer/register 130 (multiplexer 130 is a commercial part, Part No. 399, which effectively is a register/multiplexer). This selects the lower four input lines to the multiplexer 130. However, because of the synchronization signal applied to gate 151, the multiplexer 130 selects the bank switching signals each time an OP code is fetched by the CPU.

To understand the operation of the circuit shown within the dotted line 146 it should be recalled that the memory of FIG. 6 provides a 16-bit output. As mentioned, during certain display modes, 16-bits/msec. are needed for display purposes. In nondisplay modes, only 8-bits are required, particularly for interaction with the CPU. When the memory is addressed by the CPU during the indirect addressing modes the data on the A bus is not ordinarily used. However, with the circuitry shown within the dotted line 146, this otherwise "un-

4,383,296

13

used" data is put to use to provide the equivalent of the bank switching signals through multiplexer 130.

Whenever the CPU selects a predetermined range of addresses, the multiplexer 130 selects the equivalent of the bank switching signals from the A bus provided DA7 is high. (This occurs when addressing as zero page 5 the address space -1800 through 1FFF.) Once the signal on line 153 is high it is latched through gates 150, 151 and 152 causing the multiplexer 130 to select the four bits from the A bus (assuming the timing signals are high). Even if the next reference from the CPU is not to this special address range, the multiplexer 130 nonetheless remains latched with the four bits from the data bus. Once the SYN pulse drops, however, which is an indication that an OP code is being fetched, the signal on line 154 rises in potential, causing the multiplexer to switch back to the bank switching signals.

Effectively, what occurs is that when the CPU selects this special address range, (and provided DA7 is high) the bits DA0 through DA3 which are stored in memory, cause a remapping, that is, the address from the CPU accesses a different part of the memory. With the fetching of each OP code, the mapping automatically returns to the bank switching signals. Importantly, the remapping, which occurs is controlled by the bits stored in the RAM (DA0 through DA3). Thus, with the remapping information stored in RAM, toggling can occur between different portions of the memory without requiring bank switching signals, or the like from the CPU. This enhances the CPU's performance since CPU time is not used for remapping. Additionally, it provides an easy tool for programming.

For some program languages it is desirable to separate data and the program into separate portions of the memory. For example, the 128K memory can be divided into two 64K memories, one for program and one for data. Switching can occur between these memory portions without the generation of bank switching signals by the CPU with the above described circuit. This arrangement is particularly useful when using the Pascal program language.

#### DISPLAY SUBSYSTEM

The display subsystem 48 of FIG. 1 receives data from the A bus and B bus and converts the data into video signals which may be used for displaying alphanumeric characters or other images on a standard raster scanned cathode ray tube display. The display subsystem 48 specifically generates on line 197, a standard NTSC color video signal and a video black and white video signal on line 198 (FIG. 8). This display subsystem, in addition to other inputs, receives a synchronization signal, and several clocking signals. For sake of simplicity, the standard color reference signal of 3.579545 MHz is shown as C3.5M. Twice this frequency and four times this frequency are shown as C7M and C14M, respectively.

Before describing the details of the display subsystem 48, a discussion of a prior art display system will be helpful in understanding the present display subsystem. In U.S. Pat. No. 4,136,359, a video display system is described which is implemented in a commercially available computer, Apple-II, sold by Apple Computer, Inc., of Cupertino, Calif. In this system, 4-bit digital words are shifted in parallel into a shift register. These words are then circulated in the shift register at 14 MHz to define a waveform having components at 3.5 MHz. Referring to FIG. 9, line 206, assume that the digital

14

word 0001 is placed in the shift register and circulated at a rate of 14 MHz. The resultant signal which has a component of 3.5 MHz is shown on line 206. The phase relationship of this component to the 3.5 MHz reference signal determines the color of the resultant video signal. This relationship is changed by changing the 4-bit word placed in the shift register. As explained in the above-referenced patent, if the signal 1000 is placed in the register and circulated, the resultant phase relationship of the 3.5 MHz component results in the color brown, this signal is shown on line 208. With this prior art technique, the luminance was determined by the DC component of the signals such as shown on lines 206 and 208.

The display subsystem 48 of FIG. 1 also uses 4-bit words to generate the various color signals in a manner somewhat similar to the above-described system. Referring to FIG. 8, 4-bit words representative of colors (16 possible colors) are coupled to the bus 180. (The generation of these words shall be described in detail in conjunction with FIG. 7.) Instead of using a shift register which circulates the 4-bit work, the same result is achieved by using a multiplexer 205 which sequentially selects each of the lines of the bus 180. The signals on bus 180 also provide a luminance signal and a black and white video signal with a gray scale.

The 4 lines of the bus 180 are coupled to multiplexer 205; this multiplexer also receives the C7M and the C3.5M timing signals. These two timing signals cause each of the four lines to be sequentially selected and coupled to line 191. (Note that the order in which each of the lines of the bus 180 is selected does not change.)

In effect, the multiplexer operates to serialize the parallel signal from bus 180. Assume for sake of explanation that the digital signals on bus 180 are 1000 as indicated in FIG. 8. The signal on line 191 will then be 10001000 . . . . The output of the multiplexer 205 coupled to the input of the inverter 204 also receives in a sequential order, the signals from bus 180, however, in a different order. For the example shown, the input to inverter 204 is 00100010 . . . . After inversion, this results in the signal 11011101 . . . on line 192. Effectively, the signals on lines 191 and 192 are added by resistors 199 and 200. The resultant waveform is an AC signal (no DC component) shown in FIG. 9 on line 209. Thus, with the described circuit, a chroma signal is generated, having a predetermined phase relationship to the 3.5 MHz color reference signal. This phase relationship which is varied by changing the signals on bus 180 determines the color of the video signal on line 197.

In the prior art display discussed above, the DC component of the color signal determines the luminance. In the present invention, the signals on bus 180 are coupled to the base of transistor 195, consists of an AC signal from resistors 199 and 200, and the luminance level also determined by the signals on bus 180. These inputs to transistor 195, along with the C3.5M signal, generate a NTSC color signal on line 197 of improved quality when compared to the discussed prior art system.

In some cases, the signals on bus 180 are all binary ones or all binary zeros. When this occurs, there is no AC component from resistors 199 and 200 (no color signal) and the resultant signal on line 197 is either "black" or "white."

The lines of bus 180 are also coupled through resistors to the base of a transistor 196. Each of these resistors have a different value to provide a "weighting" to the binary signal.

4,383,296

15

This weighting is used for non-color displays to provide "gray" shades as opposed to having a display with only black and white. The binary signals on bus 180 drive the transistor 196 to provide a video signal on line 198. RGB is generated with weighted sums of these same five signals.

Referring now to FIG. 7, data from memory is coupled from the A bus and B bus to registers 159 and 158, respectively. These registers are clocked by the 1 MHz clocking signal and its complement, thus permitting the sequential transfer of 8-bit words every 0.5 msec. As will be described, in some display modes the data is transferred at the 2 MHz rate, and in other display modes, at a 1 MHz rate.

The registers 158 and 159 are coupled to an 8 line display bus 160. This display bus transfers data to registers 164 and 173, and also addresses to a memory 162. The registers 164 and 173 and memory 162 are enabled during specific display modes as will be apparent.

The character memory 162, in the presently preferred embodiment, is a random-access memory which stores patterns representative of alpha-numeric characters. Each time the computer is powered up, the character information is transferred from the ROM 50 into the character memory 162 during an initialization period. During character display modes, the signals from the display bus 160 are addresses, identifying particular alpha-numeric characters stored within the character memory 160. The vertical counter signals  $V_A$ ,  $V_B$ , and  $V_C$  (previously discussed in conjunction with adder 121 of FIG. 4) identify the particular line in each character which is to be displayed. Thus, the generation of the digital signals representative of each of the characters occurs in an ordinary manner. The 7-bit signal representative of each line of each character (memory output) is coupled to the shift register 167. Through timing signals not shown, either the register 164 or the character memory 162 is selected to allow the shift register 167 to receive either data directly from the A bus or B bus, or alpha-numeric character information from the memory 162.

The 7-bits of information from either memory 162 or register 164 are serialized by the shift register 167 either at a 7 MHz rate or 14 MHz rate, depending upon the display mode. The serialized data is coupled by line 185 to the multiplexer 169, pins 1 and 4. The inverse of this data is also coupled to multiplexer 169, pin 3. Line 185 is also coupled as one input to the multiplexer 166 and to the register 170 (input 1).

The output 1 of register 170 (line 186) is coupled to the multiplexer 169, pin 1; to register 170 (input 2); and to multiplexer 166. Output 2 of register 170 (line 187) is coupled to input 3 of register 170 and also to multiplexer 166. Output 3 of register 170 (line 187) provides a third input to the multiplexer 166. Input 4 of the register 170 receives the output of the multiplexer 169 (line 189). Output 4 of register 120 (line 190) provides one control signal for the multiplexer 171.

The multiplexer 171 selects either the four lines of bus 183 or the four lines of bus 184. The output of multiplexer 171, bus 180, provides the 4-bit signal discussed in conjunction with FIG. 8. During one of the high resolution display modes (AHIRES), the multiplexer 171 is controlled by a timing signal from the output of the gate 178.

The multiplexer 166 selects either the lines of bus 181 or bus 182. The output of this multiplexer provides the signals for the bus 184. In all but the AHIREs display

16

mode, multiplexer 166 selects bus 181. Thus, typically, the multiplexer 171 receives the signals from bus 174.

For purposes of description above, and also for purposes of explaining for some of the display modes below a simplifying assumption has been made. The signals coupled to the bus 180 by multiplexer 171, for most modes, are controlled by the serialized signal on line 190. This serialized signal is in synchronization with the C7M or C14M clocking signals. The multiplexer 205 of FIG. 8, which as described above, does the "spinning" for the parallel digital signal on bus 180, operates in synchronization with the multiplexer 171. In the description above, and except when otherwise noted below, it is assumed that, by way of example, if the multiplexer 171 is coupling all binary ones and zeros onto bus 180, the signal on line 191 will be either ones or zeros. Also for this condition the signal on line 192 will be all binary zeros or ones, and thus, no AC signal is generated at the base of transistor 195. However, as actually implemented, there is a "phase" difference between the clocking of the multiplexer 171 when compared to the sampling of the signals from bus 180 by the multiplexer 205. This results in a first constant AC signal on the gate of transistor 195 even when it appears that all binary ones are on bus 180, and a second constant AC signal when all binary zeros are on the bus 180. Thus, in this specification, when it states that "black" or "white" signals are being generated, instead, as currently implemented, two constant colors are generated on a color display. Where a true black and white is desired, color suppression is introduced such as through the color burst signal.

The circuit of FIG. 7, along with the circuit of FIG. 8, provides the capability for several distinct display modes. The first of these modes provides a display consisting of 40 characters (or spaces) per horizontal line. This requires a data rate of 8-bits/MHz or half the data rate the memory is capable of delivering. In this mode, data is loaded from the A bus during every other 0.5  $\mu$ sec period. (B bus is not used during this mode.) This data addresses the character memory 162, and along with the signals  $V_A$ ,  $V_B$  and  $V_C$ , provides the appropriate character line (7-bits) to the shift register 167. During this mode, registers 164 and 173 are disabled. The shift register 167 for this mode shifts the data at a data rate of 7 MHz (note  $\overline{CH80}$  is high, allowing the 7 MHz signal from gate 175 to control the shift register 167). Each 7-bit signal is shifted serially onto line 185 and then to line 189 since multiplexer 169 selects pin 4. The data is shifted through the register 170 onto line 190. The serial binary signal on line 190 causes the selection of buses 183 or 184.

The four lines of bus 183 during this mode are coupled to +V (register 173 is disabled); therefore the selection of bus 184 provides four binary ones. The selection of bus 184 provides four binary zeros through bus 181. Thus, the serial binary signal on line 190 provides either all binary ones or all binary zeros to bus 180. As discussed, the circuit of FIG. 8 will provide a black and white display with 40 characters per line.

If the inverse and flashing timing means 172 is selected, each time the shift register 167 is loaded, multiplexer 169 shifts between pins 3 and 4. This causes the characters to change from white characters on a black background to black characters on a white background, and so on.

During the 80 character per line display mode, the registers 158 and 159 are each loaded during sequential

4,383,296

17

0.5  $\mu$ sec periods (this utilizes the 2 MHz cycle rate previously discussed). The shift register 167 shifts the character data from memory 162 at a 14 MHz rate. The serialized data at the 14 MHz rate is shifted through the register 170 and again controls the multiplexer 171 as previously described. (Note that register 170 is always clocked at the 14 MHz rate.) Flashing again can be obtained as previously discussed.

In another alpha-numeric character display mode, the background of each character may be in one color and the character itself (foreground) in another color. This mode provides 40 characters per line. The character identification (address for RAM 162), is furnished on the A bus to register 159 at a frequency of 1 MHz. The color information (background color and foreground color) is furnished on the B bus as two 4-bit words to register 158. In the manner previously described, the address from register 159 selects the appropriate character from memory 162 and provides this information to shift register 167. The color information from the B bus is transferred to register 173. For purposes of explanation, assume that the 4-bits identifying the color red for the background are on bus 184 (from register 173 and multiplexer 166) and that 4-bits representing the color blue for the foreground are on bus 183. (Note that when register 173 is enabled, the signals from the register override the binary ones and zeros which otherwise appear on the lines of bus 174.) The serial binary signal representative of the character itself on line 190, selects either the color blue from bus 183 for the character itself or the color red from bus 184 for the background. The digital signals representative of these colors are transferred to bus 180 and provide the color data to the circuit of FIG. 8. For black and white displays, a "gray" scale is provided through the weighting circuit associated with transistor 196 of FIG. 8. Again, the multiplexer 169 may, through the timing means 172, alternate between the signal of line 185 and its inverse, which will have the effect of interchanging the foreground and background colors.

During the high resolution graphics modes, the character memory 162 is not used, but rather, data from the memory directly provides pattern information for display. This requires more mapping of data from within the main memory since new data is required for each line of the display. (Note that when characters are displayed, the character memory 162 provides the different signals required for the 8 lines of each character row.) During these high resolution modes, the register 164 is enabled and the character memory 162 is disabled. Thus, the data from the A bus and B bus is shifted into the shift register 167. In these modes, the "HRES" signal to multiplexer 169 causes this multiplexer to select between pins 1 and 2. Pin 2 provides the signal directly from the shift register 167 while the signal on pin 1 is effectively the signal on line 185 delayed by one period of the C14M signal. This delay occurs through the register 170 from input 2 to output 2 since register 170 is clocked at C14M.

18

During a first graphics mode, data from the display bus 160 is loaded into shift register 167 at the rate of 7-bits/MHz. The data is serialized on line 185 and in the manner previously described for displaying characters, controls the selection of all binary ones and all binary zeros through the multiplexer 171. Note, as mentioned before, in the presently preferred embodiment, unless color suppression is used, this will not result in a black and white display, but rather a two-color display. If a high bit is present on line 140 of the display bus, the inverse and flashing timing means 172 causes the multiplexer 169 to alternate between pins 1 and 2. This switching occurs at a 1 MHz rate and provides a phase shift for every other 7-bits of data coupled to the multiplexer 171 on line 190. This results in an additional color being generated on the display for every other 7-bits of data.

For the above-described graphics modes when shift register 161 is shifting at a 7 MHz rate, 8-bits may be coupled to the bus 160 during each period. Specifically, as in the case of the differing background and foreground colors for the 40 character per line display mode, two 4-bit color words are shifted into register 173 at a rate of 1 MHz. Then, the multiplexer 171 selects between two predetermined colors on buses 183 and 184. Note these colors can be changed at a 1 MHz rate.

In an additional color mode identified as "AHIREs," multiplexer 171 operates under the control of gates 176, 177 and 178. In effect, multiplexer 171 selects bus 184 and latches the signals on this bus every four cycles of the C14M clock. Data is shifted into the shift register 167 from the A bus and B bus every 0.5  $\mu$  sec the register 167 operates under the control of the C14M signal. Each data bit on line 185 is shifted first to line 186, then to line 187 and finally to line 188. These lines are coupled to the multiplexer 171 through multiplexer 166 which selects bus 182 since AHIREs is high. In effect, what occurs is that 4-bit color words are serialized onto line 185 and then brought back into parallel on bus 182. Since multiplexer 171 latches the signals on bus 184 every four cycles of the C14M signal, a new color word is generated at a 3.5 MHz rate on the bus 180. The resultant display is 140 by 192 colored blocks wherein each block can be any one of 16 colors.

In the last display mode, typically used with color suppression, data is shifted into the shift register 167 from the display bus at the rate of 14-bits/MHz. The data is serialized onto line 185 and controls the selection of either all binary ones or all zeros through multiplexer 171. This provides the highest resolution graphics display for the system.

Thus, a microcomputer with video display capability has been described. The computer is fabricated from commercially available parts and provides high utilization of these parts. Numerous existing programs including many of those which operate on the Apple-II computer, may be employed in the above-described computer.

60

65

19

4,383,296

20

T A B L E I

```

F000: 13 *****
F000: 14 *   CRITICAL TIMING   *
F000: 15 *   REQUIRES PAGE BOUND   *
F000: 16 *   CONSIDERATIONS FOR   *
F000: 17 *   CODE AND DATA   *
F000: 18 *   -----CODE----- *
F000: 19 *   VIRTUALLY THE ENTIRE *
F000: 20 *   'WRITE' ROUTINE   *
F000: 21 *   MUST NOT CROSS   *
F000: 22 *   PAGE BOUNDARIES *
F000: 23 *   CRITICAL BRANCHES IN *
F000: 24 *   THE 'WRITE', 'READ', *
F000: 25 *   AND READ ADDR SUBRS *
F000: 26 *   WHICH MUST NOT CROSS *
F000: 27 *   PAGE BOUNDARIES ARE *
F000: 28 *   NOTED IN COMMENTS *
F000: 29 *
F000: 30 *****
F000: 31 *
F000: 32 *   EQUATES   *
F000: 33 *
0200: 34 NBUF1 EQU $200
0302: 35 NBUF2 EQU $302 ; (ZERO PAGE AT $300)
F000: 36 *
00B0: 37 HRDERRS EQU $B0
00E0: 38 DVMOT EQU $E0
F000: 39 *
00B1: 40 IBSL0T EQU $B1
00B2: 41 IBTRVN EQU IBSL0T+1
00B3: 42 IBTRK EQU IBSL0T+2
00B4: 43 IBSECT EQU IBSL0T+3
00B5: 44 IDBUFP EQU IBSL0T+4 ; &5
00B7: 45 IBCMD EQU IBSL0T+6
00B8: 46 IBSTAT EQU IBSL0T+7
00B9: 47 IBSMOD EQU IBSL0T+8
00B9: 48 CSUM EQU IBSMOD ; USED ALSO FOR ADDRESS HEADER CKSUM
00BA: 49 IOBPDN EQU IBSL0T+9
00BB: 50 IMASK EQU IBSL0T+$A
00BC: 51 CURTRK EQU IBSL0T+$B
00B5: 52 DRVOTRK EQU CURTRK-7
F000: 53 ;SLOT 4, DRIVE 1
F000: 54 ;SLOT 4, DRIVE 2
F000: 55 ;SLOT 5, DRIVE 1
F000: 56 ;SLOT 5, DRIVE 2
F000: 57 ;SLOT 6, DRIVE 1
F000: 58 ;SLOT 6, DRIVE 2
0093: 59 RETRYCNT EQU IBSL0T+$12
0094: 60 SEEKCNT EQU IBSL0T+$13
009B: 61 BUF EQU IBSL0T+$1A
009F: 62 ENVTEMP EQU IBSL0T+$1E
F000: 63 *IBSL0T+$1F NOT USED
F000: 64 *
F000: 66 *****
F000: 67 *
F000: 68 *   -----READADR----- *
F000: 69 *
F000: 70 *****
0095: 71 COUNT EQU IBSL0T+$14 ; 'MUST FIND' COUNT.
0095: 72 LAST EQU IBSL0T+$14 ; 'ODD BIT' NIBLS.
0096: 73 CKSUM EQU IBSL0T+$15 ; CHECKSUM BYTE.
0097: 74 CSSTV EQU IBSL0T+$16 ; FOUR BYTES,
F000: 75 *   CHECKSUM, SECTOR, TRACK, AND VOLUME.
F000: 76 *
F000: 77 *****
F000: 78 *

```

APPLE III BOOT ROM LISTING  
 REVISION D ROM (See addr F1B9)

	21	4,383,296	22
F000:	79 *	-----WRITE-----	*
F000:	80 *		*
F000:	81 *	USES ALL NBUFS	*
F000:	82 *	AND 32-BYTE	*
F000:	83 *	DATA TABLE 'NIBL'	*
F000:	84 *		*
F000:	85 *	*****	
F000:	86 *		
F000:	87 *	*****	
F000:	88 *		*
F000:	89 *	-----READ-----	*
F000:	90 *		*
F000:	91 *	USES ALL NBUFS	*
F000:	92 *	USES LAST 54 BYTES	*
F000:	93 *	OF A CODE PAGE FOR	*
F000:	94 *	SIGNIFICANT BYTES	*
F000:	95 *	OF DNIBL TABLE.	*
F000:	96 *		*
F000:	97 *	*****	
F000:	98 *		
F000:	99 *	*****	
F000:	100 *		*
F000:	101 *	---- SEEK ----	*
F000:	102 *		*
F000:	103 *	*****	
0095:	104	TRKCNT EQU COUNT ; HALFTRKS MOVED COUNT.	
009D:	105	PRIOR EQU IBSLOT+\$1C	
009E:	106	TRKN EQU IBSLOT+\$1D	
F000:	107 *		
F000:	108 *	*****	
F000:	109 *		*
F000:	110 *	---- MSWAIT ----	*
F000:	111 *		*
F000:	112 *	*****	
0099:	113	MONTIMEL EQU CSSTV+2 ; MOTOR-ON TIME	
009A:	114	MONTIMEH EQU MONTIMEL+1 ; COUNTERS.	
F000:	115 *		
F000:	117 *	*****	
F000:	118 *		*
F000:	119 *	DEVICE ADDRESS	*
F000:	120 *	ASSIGNMENTS	*
F000:	121 *		*
F000:	122 *	*****	
C080:	123	PHASEOFF EQU \$C080 ; STEPPER PHASE OFF.	
C081:	124	PHASEON EQU \$C081 ; STEPPER PHASE ON.	
C08C:	125	Q6L EQU \$C08C ; Q7L, Q6L=READ	
C08D:	126	Q6H EQU \$C08D ; Q7L, Q6H=SENSE WPROT	
C08E:	127	Q7L EQU \$C08E ; Q7H, Q6L=WRITE	
C08F:	128	Q7H EQU \$C08F ; Q7H, Q6H=WRITE STORE	
FFEF:	129	INTERUPT EQU \$FFEF	
FFDF:	130	ENVIRON EQU \$FFDF	
0080:	131	ONEMEG EQU \$80	
007F:	132	TWOMEG EQU \$7F	
F000:	133 *	*****	
F000:	134 *		
F000:	135 *	EQUATES FOR RWTS AND BLOCK	
F000:	136 *		
F000:	137 *	*****	
C088:	138	MOTOROFF EQU \$C088	



4,383,296

23

24

```

COB9: 139 MOTORON EQU $COB9
COBA: 140 DRVOEN EQU $COBA
COBB: 141 DRV1EN EQU $COBB
COB1: 142 PHASON EQU $COB1
COB0: 143 PHSOFF EQU $COB0
0097: 144 TEMP EQU CSSTV ; PUT ADDRESS INFO HERE
0097: 145 CSUM1 EQU TEMP
0098: 146 SECT EQU CSUM1+1
0099: 147 TRACK EQU SECT+1
0099: 148 TRKN1 EQU TRACK
009A: 149 VOLUME EQU TRACK+1
0083: 150 IBRERR EQU HRDERRS+3
0082: 151 IDERR EQU HRDERRS+2
0081: 152 IBWPER EQU HRDERRS+1
0080: 153 IBNODRV EQU HRDERRS
F000: 155 *****
F000: 156 *
F000: 157 * READ WRITE A *
F000: 158 * TRACK AND SECTOR *
F000: 159 *
F000: 160 *****
F000: 161 *
F000 A0 01 162 REGRWTS LDY #1 ;RETRY COUNT
F002 A6 81 163 LDX IBSL0T ;GET SLOT # FOR THIS OPERATION
F004 84 94 164 STY SPEK0NT ;ONLY ONE RECALIBRATE PER CALL
F006 08 165 PHP ;DETERMINE INTERRUPT STATUS
F007 68 166 PLA
F008 6A 167 ROR A
F009 6A 168 ROR A ;GET INTERRUPT FLAG INTO BIT 7
F00A 6A 169 ROR A
F00B 6A 170 ROR A
F00C 85 8B 171 STA IMASK
F00E AD DF FF 172 LDA ENVIRON ;PRESERVE ENVIRONMENT
F011 85 9F 173 STA ENVTEMP
F013: 174 *
F013: 175 * NOW CHECK IF THE MOTOR IS ON. THEN START IT
F013: 176 *
F013 20 2B F1 177 USP CHDRV ;GET ZERO FLAG IF MOTOR STOPPED
F016 08 178 PHP ;SAVE TEST RESULTS
F017 A5 85 179 LDA IBBUFP ;MOVE OUT POINTER TO BUFFER INTO ZPAGE
F019 85 9B 180 STA BUF
F01B A5 8A 181 LDA IBBUFP+1
F01D 85 9C 182 STA BUF+1
F01F A5 E0 183 LDA #DVMOT
F021 85 9A 184 STA MONTIMEH
F023 A5 82 185 LDA IBDRVN ;DETERMINE DRIVE ONE OR TWO
F025 C5 8A 186 CMP IOBPDN ;SAME DRIVE USED BEFORE?
F027 85 8A 187 STA IOBPDN ;SAVE IT FOR NEXT TIME
F029 03 188 PHP ;KEEP RESULTS OF COMPARE
F02A 6A 189 ROR A ;GET DRIVE NUMBER INTO CARRY
F02B 8D 89 00 190 LDA MOTORON,X ;TURN ON THE DRIVE
F02E 90 01 191 BCC DRIVSEL ;BRANCH IF DRIVE 1 SELECTED
F030 E6 192 INX ;SELECT DRIVE 2
F031 3D 8A 00 193 DRIVSEL LDA DRVOEN,X
F034 20 4C FF 194 JSR SETIMEG ;INSURE ONE MEGAHERTZ OPERATION
F037 28 195 PLP ;WAS IT SAME DRIVE?
F038 F0 0A 196 BEQ OK
F03A 28 197 PLP
F03B A0 07 198 LDY #7 ;MUST INDICATE DRIVE OFF BY SETTING ZERO
F03D 20 56 F1 199 DRUWAIT JSR MSWAIT ;DELAY 150 MS BEFORE STEPPING (FLAG)
F040 86 200 DEY ;(ON RETURN A=0)
F041 00 FA 201 BNE DRUWAIT
F043 08 202 PHP ;NOW ZERO FLAG SET
F044 A5 83 203 OK LDA IBTRK ;GET DESTINATION TRACK
F046 A6 81 204 LDX IBSL0T ;RESTORE PROPER X (SLOT*16)
F048 20 05 F1 205 JSR MYSEEK ;AND GO TO IT
F04B: 206 *NOW AT THE DESIRED TRACK WAS THE MOTOR
F04B: 207 * ON TO START WITH?
F04B 28 208 PLP ;WAS MOTOR ON?
F04C D0 17 209 BNE TRYTRK ;IF SO, DON'T DELAY, GET IT TODAY!
F04E: 210 *

```

4,383,296

25

26

```

F04E      211 * MOTOR WAS OFF, WAIT FOR IT TO SPEED UP
F04E      212 *
F04E:A0 12 213 MOTOF LDY  #12      ; WAIT EXACTLY 100 US FOR EACH COUNT
F050:88    214 CONWAIT DEY      ; IN MONTIME
F051:D0 F1 215 BNE CONWAIT
F053:E4 99 216 INC MONTIMEH ; COUNT UP TO 0000
F055:D0 F1 217 BNE MOTOF
F057:D0 9A 218 INC MONTIMEH
F059:D0 F0 219 BNE MOTOF
F05B:      221 *****
F05B:      222 *
F05B:      223 * MOTOR SHOULD BE UP TO SPEED
F05B:      224 * IF IT STILL LOOKS STOPPED THEN
F05B:      225 * THE DRIVE IS NOT PRESENT
F05B:      226 *
F05B:      227 *****
F05B:20 2B F1 228 JSR CHKDRV ; IS DRIVE PRESENT?
F05E:D0 05 229 BNE TRYTRK ; YES, CONTINUE
F060:A9 80 230 NOHDIVEPR LDA #1BNODRV ; NO, GET TELL EM NO DRIVE!
F062:4C EB F1 231 JMP HNDLERR
F065:      232 *
F065:      233 * NOW CHECK IF IT IS NOT THE FORMAT DISK COMMAND
F065:      234 * LOCATE THE CORRECT SECTOR FOR THIS OPERATION
F065:      235 *
F065:A5 87 236 TRYTRK LDA IBCHMD ; GET COMMAND CODE #
F067:F0 77 237 BEQ ALLDONE ; IF NULL COMMAND, GO HOME TO BED
F069:C9 03 238 CMP #3 ; COMMAND IN RANGE?
F06B:8C 73 239 BCS ALLDONE ; NO, DO NOTHING!
F06D:6A 240 ROR A ; SET CARRY=1 FOR READ, 0 FOR WRITE
F06E:8C 0B 241 BCS TRYTRK2 ; MUST PREINIBLIZE FOR WRITE
F070:A0 DF F1 242 LLA ENVIRON
F072:29 7F 243 AND #1WMEG ; SHIFT TO HIGH SPEED!
F074:8D DF F1 244 STA ENVIRON
F076:20 C6 F1 245 JSR PREINIB1
F078:A0 7F 246 TRYTRK2 LDY #1FF ; ONLY 127 RETRIES OF ANY KIND
F07A:84 93 247 BNE TRYTRK2
F07C:A6 B1 248 LDA #ADR16 ; GET SLOT NUM INTO X-REG
F07E:20 BD F1 249 BNE #ADR16 ; READ NEXT ADDRESS FIELD
F080:90 21 250 BCC RDRIGHT ; IF READ IT RIGHT, HURRAH!
F082:24 B8 251 TRYADRI BIT IMASK ; SHOULD INTERRUPTS BE ALLOWED?
F084:3C 01 252 DMI NOINTR1 ; NO, DON'T ALLOW THEM
F086:58 253 CLI ; RE-ENABLED AFTER READ/READADR 'WRIT'
F088:58 254 BNE RETRYCNT ; ANOTHER MISTAKE!! FAILURE
F08A:C4 93 255 BNE RETRYCNT
F08C:10 F0 256 BPL TRYADR ; WELL, LET IT GO THIS TIME
F08E:A5 8C 257 LDA CURTRK
F090:48 258 PHA ; SAVE TRACK WE REALLY WANT
F092:C6 94 259 DEC SEEKCNT ; ONLY RECALIBRATE ONCE!
F094:D0 4F 260 BNE DRVERR ; TRIED TO RECALIBRATE A SECOND TIME, -
F096:A4 00 261 LDA #160 ; RECALIBRATE ALL OVER AGAIN! ERROR!
F098:20 27 F1 262 JSR SETTRK ; PRETEND TO BE ON TRACK 80
F09A:A9 00 263 LDA #100
F09C:20 05 F1 264 JSR MYSEEK ; MOVE TO TRACK 00
FOA0:68 265 GDBAL1 PLA
FOA2:20 05 F1 266 GDBAL JSR MYSEEK ; GO TO CORRECT TRACK THIS TIME!
FOA4:4C 7B F1 267 JMP TRYTRK2 ; LOOP BACK, TRY AGAIN ON THIS TRACK
FOA7:      268 * HAVE NOW READ AN ADDRESS FIELD CORRECTLY.
FOA7:      269 * MAKE SURE THIS IS THE TRACK, SECTOR, AND VOLUME DESIRED.
FOA7:A4 99 270 RDRIGHT LDY TRACK ; ON THE RIGHT TRACK?
FOA9:C4 8C 271 CPY CURTRK
FOAB:F0 0E 272 BEQ RETTRK ; IF SO, GOOD
FOAD:      273 * RECALIBRATING FROM THIS TRACK
FOAD:A5 8C 274 LDA CURTRK ; PRESERVE DESTINATION TRACK
FOAF:48 275 PHA
FOB0:98 276 TYA
FOB2:20 25 F1 277 JSR SETTRK
FOB4:68 278 PLA
FOB5:20 05 F1 279 JSR MYSEEK
FOB8:4C 84 F0 280 JMP TRYADR2 ; GO AHEAD AND RECALIBRATE
FOBB:      282 *
FOBB:      283 * DRIVE IS ON RIGHT TRACK, CHECK VOLUME MISMATCH
FOBB:      284 *
FOBB:A5 9A 285 RTTRK LDA VOLUME ; GET ACTUAL VOLUME HERE
FOBD:85 89 286 STA IBMOD ; TELL OPSYS WHAT VOLUME WAS THERE

```

4,383,296

27

28

```

FOBF A5 98      287 CORRECTVOL LDA SECT      ;CHECK IF THIS IS THE RIGHT SECTOR
FOCI C5 84      288          CMP IBSECT
FOCI FO 02      289          BEQ CORRECTSECT ;IF SO, DO WHATEVER WANTED
FOCI D0 BF      290          BNE TRYADR2      ;NO, TRY ANOTHER SECTOR
FOCI A5 87      291 CORRECTSECT LDA IBCMD      ;READ OR WRITE?
FOCI 4A         292          LSR A          ;THE CARRY WILL TELL
FOCI 90 2D      293          BCC WRIT      ;CARRY WAS SET FOR READ OPERATION,
FOCI 20 4B F1   294          JSR READ16     ;CARRIED FOR WRITE
FOCI 80 B5      295          BCS TRYADR2   ;CARRY SET UPON RETURN IF BAD READ
FOCI AD DF FF   296          LDA ENVIRON
FOCI 29 7F      297          AND #TWOMEG
FOCI 95 DF FF   298          STA ENVIRON    ;SET TWO MEGAHERTZ MODE
FOCI 20 11 F3   299          JSR POSTNIB16 ;DO PARTIAL POSTNIBBLE CONVERSION
FOCI 80 A8      300          BCS TRYADR2   ;CHECKSUM ERROR
FOCI A5 81      301          LDX IB SLOT   ;RESTORE SLOTNUM INTO X
FOCI 18         302 ALLDONE CLC
FOCI A9 00      303          LDA #0          ;NO ERROR
FOCI 90 04      304          BCC ALLDONE1  ;SKIP OVER NEXT BYTE WITH R11 OPCODE
FOCI 68         305 DRVERR PLA          ;REMOVE CURTRK
FOCI A9 82      306          LDA #IBDERR   ;BAD DRIVE
FOCI 38         307 HNDLERR SEC          ;INDICATE AN ERROR
FOCI 85 88      308 ALLDONE1 STA IBSTAT   ;GIVE HIM ERROR#
FOCI 8D 88 00   309          LDA MOTOROFF,X ;TURN IT OFF
FOCI 24 88      310          BIT IMASK     ;SHOULD INTERRUPTS BE ENABLED?
FOCI 30 01      311          BMI NOINTR2   ;BRANCH IF NOT
FOCI 58         312          CLI
FOCI A5 9F      313 NOINTR2 LDA ENVTEMP   ;RESTORE ORIGINAL ENVIRONMENT
FOCI 8D DF FF   314          STA ENVIRON
FOCI 50         315          RTS
FOCI 20 19 F1   316 WRIT JSR WRITE16    ;WRITE NYBBLES NOW
FOCI 90 E2      317          BCC ALLDONE   ;IF NO ERRORS
FOCI A9 81      318          LDA #IBWPER   ;DISK IS WRITE PROTECTED!!
FOCI 50 E6      319          BVC HNDLERR   ;TAKEN IF TRULY WRITE PROTECT ERROR
FOCI 40 86 FF   320          JMP TRYADR2   ;OTHERWISE ASSUME AN INTERRUPT MESSED
FOCI 105        321 *                                     THINGS UP
FOCI 105        322 * THIS IS THE 'SEEK' ROUTINE
FOCI 105        323 * SEEKS TRACK 'N' IN SLOT #X/*10
FOCI 105        324 * IF DRIVNO IS NEGATIVE, ON DRIVE 0
FOCI 105        325 * IF DRIVNO IS POSITIVE, ON DRIVE 1
FOCI 105        326 *
FOCI 0A         327 MYSEEK ASL A          ;ASSUME TWO PHASE STEPPER
FOCI 85 99      328 SEV1 STA TRKN1     ;SAVE DESTINATION TRACK(*2)
FOCI 20 19 F1   329          JSR ALLOFF    ;TURN ALL PHASES OFF TO BE SURE
FOCI 20 3E F1   330          JSR DRVINDX    ;GET INDEX TO PREVIOUS TRACK FOR CURRENT
FOCI 85 85      331          LDA DRVOTRK,X ;DRIVE
FOCI 85 80      332          STA CURTRK   ;THIS IS WHERE I AM
FOCI A5 99      333          LDA TRKN1    ;AND WHERE I'M GOING TO
FOCI 95 85      334          STA DRVOTRK,X
FOCI 20 00 F4   335 GOSEEK JSR SEEK      ;GO THERE!
FOCI A0 03      336 ALLOFF LDY #3        ;TURN OFF ALL PHASES BEFORE RETURNING
FOCI 98         337 NXOFF TYA          ;(SEND PHASE IN ACC.)
FOCI 20 4A F4   338          JSR CLRPHASE  ;CARRY IS CLEAR, PHASES SHOULD BE TURNED
FOCI 86         339          DEY          OFF
FOCI 10 F9      340          BPL NXOFF
FOCI 46 BC      341          LSR CURTRK   ;DIVIDE BACK DOWN
FOCI 60         342          RTS          ;ALL OFF... NOW IT'S DARK
FOCI 125        344 *
FOCI 125        345 * THIS SUBROUTINE SETS THE SLOT DEPENDENT TRACK
FOCI 125        346 * LOCATION
FOCI 125        347 *
FOCI 20 3E F1   348 SETTRK JSR DRVINDX    ;GET INDEX TO DRIVE NUMBER.
FOCI 95 85      349          STA DRVOTRK,X
FOCI 60         350          RTS
FOCI 12B        351 *****
FOCI 12B        352 *
FOCI 12B        353 * SUBR TO TELL IF MOTOR IS STOPPED
FOCI 12B        354 *
FOCI 12B        355 * IF MOTOR IS STOPPED, CONTROLLER'S
FOCI 12B        356 * SHIFT REG WILL NOT BE CHANGING.
FOCI 12B        357 *
FOCI 12B        358 * RETURN Y=0 AND ZERO FLAG SET IF IT IS STOPPED.

```

4,383,296

29

30

```

F12B: 359 *
F12B: 360 *****
F12B: A0 00 361 CHKDRV LDY #0 INIT LOOP COUNTER
F12D: BD BC CO 362 CHKDRV1 LDA Q6L, X READ THE SHIFT REG
F130: 20 3D F1 363 JSR CKDRTS DELAY
F132: 48 364 PHA
F134: 68 365 PLA MORE DELAY
F136: 0E 80 366 CMP Q6L, X HAS SHIFT REG CHANGED?
F138: D0 03 367 BNE CKDRTS YES, MOTOR IS MOVING
F13A: 88 368 DEY NO, DEC RETRY COUNTER
F13B: D0 F0 369 BNE CHKDRV1 AND TRY 256 TIMES
F13D: 80 370 CKDRTS RTS THEN RETURN
F13E: 371 *
F13F: 48 372 DRVINDX PHA PRESERVE ACC.
F13F: 8A 373 TXA GET SLOT(**10)/8
F140: 4A 374 LSR A
F141: 4A 375 LSR A
F142: 4A 376 LSR A
F143: 05 8D 377 ORA IBDRVN FOR DRIVE 0 OR 1
F145: AA 378 TAX INTO X FOR INDEX TO TABLE
F146: 68 379 PLA RESTORE ACC.
F147: 60 380 RTS
F148: 381 *****
F148: 382 *
F148: 383 * NOTE: FORMATTING ROUTINES
F148: 384 * NOT INCLUDED FOR SOS
F148: 385 *
F148: 386 *****
F148: 387 *****
F148: 388 *****
F148: 389 *
F148: 390 * READ SUBROUTINE *
F148: 391 * (16-SECTOR FORMAT) *
F148: 392 *
F148: 393 *****
F148: 394 *
F148: 395 * READS ENCODED BYTES *
F148: 396 * INTO NBUF1 AND NBUF2 *
F148: 397 *
F148: 398 * FIRST READS NBUF2 *
F148: 399 * HIGH TO LOW, *
F148: 400 * THEN READS NBUF1 *
F148: 401 * LOW TO HIGH. *
F148: 402 *
F148: 403 * ---- ON ENTRY ---- *
F148: 404 *
F148: 405 * X-REG: SLOTNUM *
F148: 406 * TIMES *10. *
F148: 407 *
F148: 408 * READ MODE (Q6L, Q7L) *
F148: 409 *
F148: 410 * ---- ON EXIT ---- *
F148: 411 *
F148: 412 * CARRY SET IF ERROR. *
F148: 413 *
F148: 414 * IF NO ERROR: *
F148: 415 * A-REG HOLDS $AA. *
F148: 416 * X-REG UNCHANGED. *
F148: 417 * Y-REG HOLDS $00. *
F148: 418 * CARRY CLEAR. *
F148: 419 * ---- CAUTION ---- *
F148: 420 *

```

4,383,296

31

32

```

F148:      421 *      OBSERVE      *
F148:      422 *      'NO PAGE CROSS'      *
F148:      423 *      WARNINGS ON      *
F148:      424 *      SOME BRANCHES!!      *
F148:      425 *      *      *
F148:      426 *      ---- ASSUMES ----      *
F148:      427 *      *      *
F148:      428 *      1 USEC CYCLE TIME      *
F148:      429 *      *      *
F148:      430 *****
F148 AO 20      431 READ16 LDY ##20      'MUST FIND' COUNT.
F14A 8B      432 RSYNC DEY      IF CAN'T FIND MARKS.
F14B F0 6B      433 BEQ RDERR      THEN EXIT WITH CARRY SET
F14D BD 8C CO      434 RD1 LDA Q6L,X      READ NIBL.
F150 10 FB      435 BPL RD1      *** NO PAGE CROSS! ***
F152 49 D5      436 RSYNC1 EOR ##D5      DATA MARK 1?
F154 D0 F4      437 BNE RSYNC      LOOP IF NOT.
F156 EA      438 NOP      DELAY BETWEEN NIBLS.
F157 BD 8C CO      439 RD2 LDA Q6L,X
F15A 10 FB      440 BPL RD2      *** NO PAGE CROSS! ***
F15C C9 AA      441 CMP ##AA      DATA MARK 2?
F15E D0 F2      442 BNE RSYNC1      (IF NOT, IS IT DM1?)
F160 A0 55      443 LDY ##55      INIT NBUF2 INDEX.
F162      444 *      (ADDED NIBL DELAY)
F162 BD 8C CO      445 RD3 LDA Q6L,X
F165 10 FB      446 BPL RD3      *** NO PAGE CROSS! ***
F167 C9 AD      447 CMP ##AD      DATA MARK 3?
F169 D0 E7      448 BNE RSYNC1      (IF NOT, IS IT DM1?)
F16B      449 *      (CARRY SET IF DM3)
F16B BD 8C CO      450 RD4 LDA Q6L,X
F16E 10 FB      451 BPL RD4      *** NO PAGE CROSS! ***
F170 99 02 03      452 STA NBUF2,Y      STORE BYTES DIRECTLY
F173 AD EF FF      453 LDA INTERUPT      POLL INTERRUPT LINE
F176 05 8B      454 ORA IMASK      (THIS MAY BE USED TO INVALIDATE POLL)
F17B 10 40      455 BPL GOSERV
F17A 8B      456 DEY      INDEX TO NEXT
F17B 10 EE      457 BPL RD4
F17D C9      458 RD5 INY      (FIRST TIME Y=0)
F17E BD 8C CO      459 RD5A LDA Q6L,X      GET ENCODED BYTES OF NBUF1
F181 10 FB      460 BPL RD5A
F183 99 00 00      461 STA NBUF1,Y
F186 AD EF FF      462 LDA INTERUPT      POLL INTERRUPT LINE
F189 05 8B      463 ORA IMASK      (THIS MAY BE USED TO INVALIDATE POLL)
F18B 10 2D      464 BPL GOSERV
F18D C0 E4      465 CPY ##E4      WITHIN 1 MS OF COMPLETION?
F18F D0 EC      466 BNE RD5
F191 C9      467 INY
F192 BD 8C CO      468 RD6 LDA Q6L,X      NO POLL FROM NOW ON
F195 10 FB      469 BPL RD6
F197 99 00 00      470 STA NBUF1,Y
F19A C8      471 INY      FINISH OUT NBUF1 PAGE
F19B D0 F5      472 BNE RD6
F19D BD 8C CO      473 RDCKSUM LDA Q6L,X      GET CHECKSUM BYTE
F1A0 10 FB      474 BPL RDCKSUM
F1A2 85 96      475 STA CKSUM
F1A4 EA      476 NOP      EXTRA DELAY BETWEEN BYTES
F1A5 BD 8C CO      477 RD7 LDA Q6L,X
F1A8 10 FB      478 BPL RD7
F1AA C9 DE      479 CMP ##DE      *** NO PAGE CROSS! ***
F1AC D0 0A      480 BNE RDERR      FIRST BIT SLIP MARK?
F1AE EA      481 NOP      (ERR IF NOT)
F1AF BD 8C CO      482 RDB LDA Q6L,X      DELAY BETWEEN NIBLS.
F1B2 10 FB      483 BPL RDB      *** NO PAGE CROSS! ***
F1B4 C9 AA      484 CMP ##AA      SECOND BIT SLIP MARK?
F1B6 F0 5F      485 BEQ RDEXIT      (DONE IF IT IS)
F1B8 39      486 RDERR SEC      INDICATE 'ERROR EXIT'
F1B9 60      487 RTS      RETURN FROM READ16 OR RDADR16.
F1BA      488 *
F1BA 4C B3 F2      489 GOSERV JMP SERVICE      GO SERVICE INTERRUPT

```

ADDRESS F1B9 SPECIFICS  
 ROM REVISION: 60  
 REV 0 — 60  
 REV 1 — AD

	33	4,383,296	34
F1BD:	491	*****	
F1BD:	492	*	*
F1BD:	493	* READ ADDRESS FIELD	*
F1BD:	494	* SUBROUTINE	*
F1BD:	495	* (16-SECTOR FORMAT)	*
F1BD:	496	*	*
F1BD:	497	*****	
F1BD:	498	*	*
F1BD:	499	* READS VOLUME, TRACK	*
F1BD:	500	* AND SECTOR	*
F1BD:	501	*	*
F1BD:	502	* ---- ON ENTRY ----	*
F1BD:	503	*	*
F1BD:	504	* XREG: SLOTNUM TIMES \$10	*
F1BD:	505	*	*
F1BD:	506	* READ MODE (Q6L: Q7L	*
F1BD:	507	*	*
F1BD:	508	* ---- ON EXIT ----	*
F1BD:	509	*	*
F1BD:	510	* CARRY SET IF ERROR	*
F1BD:	511	*	*
F1BD:	512	* IF NO ERROR	*
F1BD:	513	* A-REG HOLDS \$AA	*
F1BD:	514	* Y-REG HOLDS \$00.	*
F1BD:	515	* X-REG UNCHANGED.	*
F1BD:	516	* CARRY CLEAR.	*
F1BD:	517	*	*
F1BD:	518	* CSSTV HOLDS CHKSUM.	*
F1BD:	519	* SECTOR, TRACK, AND	*
F1BD:	520	* VOLUME READ.	*
F1BD:	521	*	*
F1BD:	522	* USES TEMPS COUNT,	*
F1BD:	523	* LAST, CSUM, AND	*
F1BD:	524	* 4 BYTES AT CSSTV	*
F1BD:	525	*	*
F1BD:	526	* ---- EXPECTS ----	*
F1BD:	527	*	*
F1BD:	528	* ORIGINAL 10-SECTOR	*
F1BD:	529	* NORMAL DENSITY NIBLS	*
F1BD:	530	* (4-BIT), ODD BITS,	*
F1BD:	531	* THEN EVEN	*
F1BD:	532	*	*
F1BD:	533	* ---- CAUTION ----	*
F1BD:	534	*	*
F1BD:	535	* OBSERVE	*
F1BD:	536	* 'NO PAGE CROSSES'	*
F1BD:	537	* WARNINGS ON	*
F1BD:	538	* SOME BRANCHES!!	*
F1BD:	539	*	*
F1BD:	540	* ---- ASSUMES ----	*
F1BD:	541	*	*
F1BD:	542	* 1 USEC CYCLE TIME	*
F1BD:	543	*	*
F1BD:	544	*****	
F1BD: A0 FC	545	RDADR16 LDY ##FC	
F1BF: 84 95	546	STY COUNT ; 'MUST FIND' COUNT.	



4,383,296

37

38

F219	26 *	(W PROT VIOLATION) *	
F219	27 *		
F219	28 *	IF NO ERROR.	
F219	29 *		
F219	30 *	A-REG UNCERTAIN	
F219	31 *	X-REG UNCHANGED	
F219	32 *	Y-REG HOLDS \$00.	
F219	33 *	CARRY CLEAR	
F219	34 *		
F219	35 *	----- ASSUMES -----	
F219	36 *		
F219	37 *	1 USEC CYCLE TIME	
F219	38 *		
F219	39	*****	
F219 38	40	WRITE16 SEC	ANTICIPATE WPROT ERR.
F21A B8	41	CLV	TO INDICATE WRITE PROTECT ERROR INSTEAD OF
F21B BD BD CO	42	LDA Q6H,X	INTERUPT
F21E BD BE CO	43	LDA Q7L,X	SENSE WPROT FLAG
F221 30 F5	44	BMI WEXIT	BRANCH IF NOT WRITE PROTECTED
F223 A9 FF	45	WRT1 LDA #\$FF	SYNC DATA.
F225 9D BF CO	46	STA Q7H,X	(5) GOTO WRITE MODE
F228 1D BC CO	47	ORA Q6L,X	(4)
F22B A0 04	48	LDY #\$4	(2) FOR FIVE NIBLS
F22D EA	49	NOP	(2)
F22E 48	50	PHA	(4)
F22F 68	51	PLA	(3)
F230 48	52	WSYNC PHA	(4) EXACT TIMING
F231 68	53	PLA	(3) EXACT TIMING
F233 20 BD F2	54	JSR WNIBL7	(13,9,6) WRITE SYNC
F235 88	55	DEY	(2)
F236 D0 F5	56	BNE WSYNC	(2*) MUST NOT CROSS PAGE!
F238 A9 D5	57	LDA #\$D5	(2) 1ST DATA MARK
F23A 20 BC F2	58	JSR WNIBL9	(15,9,6)
F23D A9 AA	59	LDA #\$AA	(2) 2ND DATA MARK
F23F 20 DC F2	60	JSR WNIBL9	(15,9,6)
F242 A9 AD	61	LDA #\$AD	(2) 3RD DATA MARK.
F244 20 BC F2	62	JSR WNIBL9	(15,9,6)
F247 A0 55	63	LDY #\$55	(2) NBUF2 INDEX
F249 EA	64	NOP	(2) FOR TIMING
F24A EA	65	NOP	(2)
F24B EA	66	NOP	(2)
F24C D0 08	67	BNE VRYFRST	(3) BRANCH ALWAYS
F24E AD EF FF	68	WINTRPT LDA INTERRUPT	(4) POLL INTERRUPT LINE
F251 05 8B	69	ORA IMASK	(3)
F253 EA	70	NOP	(2)
F254 10 5D	71	BPL SERVICE	(2) BRANCH IF INTERRUPT HAS OCCURED
F256 30 00	72	VRYFRST BMI WRTFRST	(3) FOR TIMING
F258 B9 02 03	73	WRTFRST LDA NBUF2,Y	(4)
F25B 9D 8D CO	74	STA Q6H,X	(5) STORE ENCODED BYTE
F25E BD 8C CO	75	LDA Q6L,X	(4) TIME MUST = 32 US PER BYTE!
F261 88	76	DEY	(2)
F262 10 EA	77	BPL WINTRPT	(3) (2 IF BRANCH NOT TAKEN)
F264 98	78	TYA	(2) INSURE NO INTERRUPT THIS BYTE.
F265 30 03	79	BMI WMIDLE	(3) BRANCH ALWAYS.
F267 AD EF FF	80	WNTRPT1 LDA INTERRUPT	(4) POLL INTERRUPT LINE
F26A 05 8B	81	WMIDLE ORA IMASK	(3)
F26C EA	82	NOP	(2)
F26D 30 02	83	BMI WDATA2	(3) BRANCH IF NO INTERRUPT
F26F 10 42	84	BPL SERVICE	GO SERVICE INTERRUPT.
F271 C8	85	WDATA2 INY	(2)
F272 B9 00 02	86	LDA NBUF1,Y	(4)
F275 9D 8D CO	87	STA Q6H,X	(5) STORE ENCODED BYTE
F278 BD 8C CO	88	LDA Q6L,X	(4)
F27B C0 E4	89	CPY #\$E4	(2) WITHIN 1 MS OF COMPLETION?
F27D D0 E8	90	BNE WNTRPT1	(3) (2) NO KEEP WRITTING AND POLLING.
F27F EA	91	NOP	(2)
F280 C8	92	INY	(2)
F281 EA	93	WDATA3 NOP	(2)
F282 EA	94	NOP	(2)
F283 48	95	PHA	(4)
F284 68	96	PLA	(3)
F285 B9 00 02	97	LDA NBUF1,Y	(4) WRITE LAST OF ENCODED BYTES



4,383,296

39

40

```

F288 9D BD CO 98 STA Q6H,X (5) WITHOUT POLLING INTERRUPTS.
F28B BD BC CO 99 LDA Q6L,X (4)
F28E A5 96 100 LDA CKSUM (3) NORMALLY FOR TIMING
F290 C8 101 INY (2)
F291 D0 EE 102 BNE WDATA3 (3) (2)
F293 F0 00 103 BEQ WRCKSUM (3) BRANCH ALWAYS
F295 20 BD F2 104 WRCKSUM JSR WNIBL7 (13,9,6) GO WRITE CHECK SUM
F298 A9 DE 105 LDA #SDE (2) DM4, BIT SLIP MARK
F29A 20 BC F2 106 JSR WNIBL9 (15,9,6) WRITE IT
F29D A9 AA 107 LDA #SAA (2) DM5, BIT SLIP MARK
F29F 20 BC F2 108 JSR WNIBL9 (15,9,6) WRITE IT
F2A2 A9 EB 109 LDA #SEB (2) DM6, BIT SLIP MARK
F2A4 20 BC F2 110 JSR WNIBL9 (15,9,6) WRITE IT
F2A7 A9 FF 111 LDA #SFF (2) TURN-OFF BYTE
F2A9 20 BC F2 112 JSR WNIBL9 (15,9,9) WRITE IT
F2AC BD BE CO 113 NOWRITE LDA Q7L,X (OUT OF WRITE MODE)
F2AF BD BC CO 114 LDA Q6L,X (TO READ MODE)
F2B2 60 115 RTS (RETURN FROM WRITE)
F2B3 116 *
F2B3 38 117 SERVICE SEC (TREAT INTERRUPTION AS ERROR)
F2B4 20 54 F3 118 BIT SEV (SET VFLAG TO INDICATE INTERRUPT)
F2B7 20 AC F2 119 JSR NOWRITE (TAKE IT OUT OF WRITE MODE)
F2BA 58 120 CLI (COULD NOT HAVE GOT HERE WITHOUT CLI OK)
F2BB 60 121 RTS
F2BC 122 *****
F2BC 123 *
F2BC 124 * 7-BIT NIBL WRITE SUBRS *
F2BC 125 *
F2BC 126 * A-REG ORD PRIOR EXIT *
F2BC 127 * CARRY CLEARED *
F2BC 128 *
F2BC 129 *****
F2BC 18 130 WNIBL9 CLC (2) 9 CYCLES, THEN WRITE
F2BD 48 131 WNIBL7 PHA (3) 7 CYCLES, THEN WRITE
F2BE 60 132 PLA (4)
F2BF 9D BD CO 133 WNIBL STA Q6H,X (5) NIBL WRITE SUB
F2C0 10 9C CO 134 LRA Q6L,X (4) CLOBBERS ACC NOT CARR
F2C5 60 135 RTS
F2C6 136 *
F2C6 138 *****
F2C6 139 *
F2C6 140 * PRENIBLIZE SUBR *
F2C6 141 * (16-SECTOR FORMAT) *
F2C6 142 *
F2C6 143 *****
F2C6 144 *
F2C6 145 * CONVERTS 256 BYTES OF *
F2C6 146 * USER DATA IN (BUF) INTO *
F2C6 147 * ENCODED BYTES TO BE *
F2C6 148 * WRITTEN DIRECTLY TO DISK *
F2C6 149 * ENCODED CHECK SUM IN *
F2C6 150 * ZERO PAGE 'CKSUM' *
F2C6 151 *
F2C6 152 * ---- ON ENTRY ---- *
F2C6 153 *
F2C6 154 * BUF IS 2-BYTE POINTER *
F2C6 155 * TO 256 BYTES OF USER *
F2C6 156 * DATA *
F2C6 157 *
F2C6 158 * ---- ON EXIT ---- *
F2C6 159 *
F2C6 160 * A-REG CHECK SUM *
F2C6 161 * X-REG UNCERTAIN *
F2C6 162 * Y-REG HOLDS 0 *
F2C6 163 * CARRY SET *
F2C6 164 *
F2C6 165 *****
F2C6 A2 02 166 PRENIB16 LDX #S2 (START NBUF2 INDEX)
F2C8 A0 00 167 LDY #0 (START USER BUF INDEX)
F2CA 88 168 PRENIB1 DEY (NEXT USER BYTE)
F2CB B1 9B 169 LDA (BUF),Y
F2CD 4A 170 LSR A (SHIFT TWO BITS OF)
F2CE 3E 01 03 171 ROL NBUF2-1,X (CURRENT USER BYTE)

```

4,383,296

41		42	
F2D1 4A	172	LSR A	INTO CURRENT NBUF2
F2D2 3E 01 03	173	ROL NBUF2-1,X	BYTE
F2D3 99 01 02	174	STA NBUF1+1,Y	(6 BITS LEFT)
F2D8 E8	175	INX	FROM 0 TO \$55
F2D9 E0 56	176	CPX #\$56	
F2DB 90 ED	177	RCC PRENIB1	BR IF NO WRAPAROUND
F2DD A2 00	178	LDX #0	RESET NBUF2 INDEX
F2DF 98	179	TYA	USER BUF INDEX
F2E0 D0 E8	180	DNE PRENIB1	(DONE IF ZERO)
F2E2 A0 56	181	LDY #\$56	(ACC=0 FOR CHECK SUM)
F2E4 59 00 03	182	PRENIB3 EOR NBUF2-2,Y	COMBINE WITH PREVIOUS
F2E7 29 3F	183	PRENIB2 AND #\$3F	STRIP GARBAGE BITS
F2E9 AA	184	TAX	TO FORM RUNNING CHECK SUM
F2EA BD 55 F3	185	LDA NIBL,X	GET ENCODED EQUIV
F2ED 99 01 03	186	STA NBUF2-1,Y	REPLACE PREVIOUS
F2F0 B9 00 03	187	LDA NBUF2-2,Y	RESTORE ACTUAL PREVIOUS
F2F3 88	188	DEY	
F2F4 D0 EE	189	DNE PRENIB3	LOOP UNTIL ALL OF NBUF2 IS CONVERTED
F2F6 29 3F	190	AND #\$3F	
F2F8 59 01 02	191	PRENIB4 EOR NBUF1+1,Y	NOW DO THE SAME FOR
F2FB AA	192	TAX	NIBBLE BUFFER 1
F2FC BD 55 F3	193	LDA NIBL,X	TO DO ANY BACK TRACKING (NBUF1-1)
F2FF 99 00 02	194	STA NBUF1,Y	
F302 B9 01 02	195	LDA NBUF1+1,Y	RECOVER THAT WHICH IS NOW 'PREVIOUS'
F305 C8	196	INY	
F306 D0 F0	197	BNE PRENIB4	
F30B AA	198	TAX	USE LAST AS CHECK SUM
F309 BD 55 F3	199	LDA NIBL,X	
F30C 85 96	200	STA CKSUM	
F30E 4C 4C F3	201	JMP SETIMEG	ALL DONE.
F311	203	*****	
F311	204	*	
F311	205	POSTNIBLIZE SUBR	
F311	206	16-SECTOR FORMAT	
F311	207	*	
F311	208	*****	
F311	209	*	
F311 A0 55	210	POSTNIB16 LDY #\$55	FIRST CONVERT TO 6 BIT NIBBLES
F313 A9 00	211	LDA #0	INIT CHECK SUM
F315 BE 02 03	212	PNIBL1 LDX NBUF2,Y	GET ENCODED BYTE
F318 5D 00 F3	213	EOR DNIBL,X	
F31B 99 02 03	214	STA NBUF2,Y	REPLACE WITH 6 BIT EQUIV
F31E 88	215	DEY	
F31F 10 F4	216	BPL PNIBL1	LOOP UNTIL DONE WITH NIBBLE BUFFER 2
F321 C8	217	INY	NOW Y=0
F322 BE 00 02	218	PNIBL2 LDX NBUF1,Y	DO THE SAME WITH
F325 5D 00 F3	219	EOR DNIBL,X	
F328 99 00 02	220	STA NBUF1,Y	NIBBLE BUFFER 1
F32B C8	221	INY	DO ALL 256 BYTES
F32C D0 F4	222	BNE PNIBL2	
F32E A6 96	223	LDX CKSUM	MAKE SURE CHECK SUM MATCHES
F330 5D 00 F3	224	EOR DNIBL,X	BETTER BE ZERO
F333 38	225	SEC	ANTICIPATE ERROR
F334 D0 16	226	BNE POSTERR	BRANCH IF IT IS
F336 A2 56	227	LDX #\$56	INIT NBUF2 INDEX
F33B CA	228	POST2 DEX	NBUF IDX \$55 TO \$0
F339 30 FB	229	BMI POST1	WRAPAROUND IF NEG
F33B B9 00 02	230	LDA NBUF1,Y	
F33E 5E 02 03	231	LSR NBUF2,X	SHIFT 2 BITS FROM
F341 2A	232	ROL A	CURRENT NBUF2 NIBL
F342 5E 02 03	233	LSR NBUF2,X	INTO CURRENT NBUF1
F345 2A	234	ROL A	NIBL
F346 91 9B	235	STA (BUF),Y	BYTE OF USER DATA
F348 C8	236	INY	NEXT USER BYTE
F349 D0 ED	237	BNE POST2	
F34B 18	238	CLC	GOOD DATA
F34C	239	POSTERR EQU *	
F34C AD DF FF	240	SETIMEG LDA ENVIRON	
F34F 09 80	241	ORA #ONEMEG	SET TO ONE MEGAHERTZ CLOCK RATE
F351 8D DF FF	242	STA ENVIRON	
F354 60	243	SEV RTS	(SEV USED TO SET VFLAG)

4,383,296

43

44

F355:	245	*****	
F355:	246	*	
F355:	247	*	6-BIT TO 7-BIT
F355:	248	*	NIBL CONVERSION TABLE
F355:	249	*	
F355:	250	*****	
F355:	251	*	
F355:	252	*	CODES WITH MORE THAN
F355:	253	*	ONE PAIR OF ADJACENT
F355:	254	*	ZEROES OR WITH NO
F355:	255	*	ADJACENT ONES (EXCEPT
F355:	256	*	37) ARE EXCLUDED.
F355:	257	*	
F355:	258	*****	
F355: 96 97 9A	259	NIBL	DFB \$96, \$97, \$9A
F358: 9B 9D 9E	260		DFB \$9B, \$9D, \$9E
F35B: 9F A6 A7	261		DFB \$9F, \$A6, \$A7
F35E: AB AC AD	262		DFB \$AB, \$AC, \$AD
F361: AE AF B2	263		DFB \$AE, \$AF, \$B2
F364: B3 B4 B5	264		DFB \$B3, \$B4, \$B5
F367: B6 B7 B9	265		DFB \$B6, \$B7, \$B9
F36A: BA BB BC	266		DFB \$BA, \$BB, \$BC
F36D: BD BE BF	267		DFB \$BD, \$BE, \$BF
F370: CB CD CE	268		DFB \$CB, \$CD, \$CE
F373: CF D3 D6	269		DFB \$CF, \$D3, \$D6
F376: D7 D9 DA	270		DFB \$D7, \$D9, \$DA
F379: DB DC DD	271		DFB \$DB, \$DC, \$DD
F37C: DE DF E5	272		DFB \$DE, \$DF, \$E5
F37F: E6 E7 E9	273		DFB \$E6, \$E7, \$E9
F382: EA EB EC	274		DFB \$EA, \$EB, \$EC
F385: ED EE EF	275		DFB \$ED, \$EE, \$EF
F388: F2 F3 F4	276		DFB \$F2, \$F3, \$F4
F38B: F5 F6 F7	277		DFB \$F5, \$F6, \$F7
F38E: F9 FA FB	278		DFB \$F9, \$FA, \$FB
F391: FC FD FE	279		DFB \$FC, \$FD, \$FE
F394: FF	280		DFB \$FF
F395:	282	*****	
F395:	283	*	
F395:	284	*	7-BIT TO 6-BIT
F395:	285	*	DENIBLIZE TABL
F395:	286	*	(16-SECTOR FORMAT)
F395:	287	*	
F395:	288	*	VALID CODES
F395:	289	*	\$96 TO \$FF ONLY.
F395:	290	*	
F395:	291	*	
F395:	292	*	CODES WITH MORE THAN
F395:	293	*	ONE PAIR OF ADJACENT
F395:	294	*	ZEROES OR WITH NO
F395:	295	*	ADJACENT ONES (EXCEPT
F395:	296	*	BIT 7) ARE EXCLUDED
F395:	297	*****	
F395: 00	298	BRK	ONE BYET LEFT OVER
F300:	299	DNIBL	EQU REGRWTS+\$300
F396: 00 01 98	300		DFB \$00, \$01, \$98
F399: 99 02 03	301		DFB \$99, \$02, \$03

				45	4,383,296	46
F39C	9C	04	05	302	DFB	\$9C, \$04, \$05
F39F	06	A0	A1	303	DFB	\$06, \$A0, \$A1
F3A2	A2	A3	A4	304	DFB	\$A2, \$A3, \$A4
F3A5	A5	07	08	305	DFB	\$A5, \$07, \$08
F3A8	A8	A9	AA	306	DFB	\$A8, \$A9, \$AA
F3AB	09	0A	0B	307	DFB	\$09, \$0A, \$0B
F3AE	0C	0D	0E	308	DFB	\$0C, \$0D, \$0E
F3B1	B1	0E	0F	309	DFB	\$B1, \$0E, \$0F
F3B4	10	11	12	310	DFB	\$10, \$11, \$12
F3B7	13	B8	14	311	DFB	\$13, \$B8, \$14
F3BA	15	16	17	312	DFB	\$15, \$16, \$17
F3BD	18	19	1A	313	DFB	\$18, \$19, \$1A
F3C0	C0	C1	C2	314	DFB	\$C0, \$C1, \$C2
F3C3	C3	C4	C5	315	DFB	\$C3, \$C4, \$C5
F3C6	C6	C7	C8	316	DFB	\$C6, \$C7, \$C8
F3C9	C9	CA	1B	317	DFB	\$C9, \$CA, \$1B
F3CC	CC	1C	1D	318	DFB	\$CC, \$1C, \$1D
F3CF	1E	D0	D1	319	DFB	\$1E, \$D0, \$D1
F3D2	D2	1F	D4	320	DFB	\$D2, \$1F, \$D4
F3D5	D5	20	21	321	DFB	\$D5, \$20, \$21
F3D8	D8	22	23	322	DFB	\$D8, \$22, \$23
F3DB	24	25	26	323	DFB	\$24, \$25, \$26
F3DE	27	28	E0	324	DFB	\$27, \$28, \$E0
F3E1	E1	E2	E3	325	DFB	\$E1, \$E2, \$E3
F3E4	E4	29	2A	326	DFB	\$E4, \$29, \$2A
F3E7	2B	E8	2C	327	DFB	\$2B, \$E8, \$2C
F3EA	2D	2E	2F	328	DFB	\$2D, \$2E, \$2F
F3ED	30	31	32	329	DFB	\$30, \$31, \$32
F3F0	F0	F1	33	330	DFB	\$F0, \$F1, \$33
F3F3	34	35	36	331	DFB	\$34, \$35, \$36
F3F6	37	38	F8	332	DFB	\$37, \$38, \$F8
F3F9	39	3A	3D	333	DFB	\$39, \$3A, \$3D
F3FC	3C	3D	3E	334	DFB	\$3C, \$3D, \$3E
F3FF	3F			335	DFB	\$3F
F400:				337	*****	
F400:				338	*	*
F400:				339	* FAST SEEK SUBROUTINE	*
F400:				340	*	*
F400:				341	*****	
F400:				342	*	*
F400:				343	* ---- ON ENTRY ----	*
F400:				344	*	*
F400:				345	* X-REG HOLDS SLOTNUM	*
F400:				346	* TIMES \$10.	*
F400:				347	*	*
F400:				348	* A-REG HOLDS DESIRED	*
F400:				349	* HALFTRACK.	*
F400:				350	* (SINGLE PHASE)	*
F400:				351	*	*
F400:				352	* CURTRK HOLDS CURRENT	*
F400:				353	* HALFTRACK.	*
F400:				354	*	*
F400:				355	* ---- ON EXIT ----	*
F400:				356	*	*
F400:				357	* A-REG UNCERTAIN.	*
F400:				358	* Y-REG UNCERTAIN.	*
F400:				359	* X-REG UNDISTURBED.	*
F400:				360	*	*
F400:				361	* CURTRK AND TRKN HOLD	*
F400:				362	* FINAL HALFTRACK.	*

	47	4,383,296	48
F400:	363 *		*
F400:	364 *	PRIOR HOLDS PRIOR	*
F400:	365 *	HALFTRACK IF SEEK	*
F400:	366 *	WAS REQUIRED.	*
F400:	367 *		*
F400:	368 *	MONTIMEL AND MONTIMEH	*
F400:	369 *	ARE INCREMENTED BY	*
F400:	370 *	THE NUMBER OF	*
F400:	371 *	100 USEC QUANTUMS	*
F400:	372 *	REQUIRED BY SEEK	*
F400:	373 *	FOR MOTOR ON TIME	*
F400:	374 *	OVERLAP.	*
F400:	375 *		*
F400:	376 *	--- VARIABLES USED ---	*
F400:	377 *		*
F400:	378 *	CURTRK, TRKN, COUNT,	*
F400:	379 *	PRIOR, SLOTTEMP	*
F400:	380 *	MONTIMEL, MONTIMEH	*
F400:	381 *		*
F400:	382	*****	
F400: B5 9E	383	SEEK STA TRKN	SAVE TARGET TRACK
F402: C5 8C	384	CMP CURTRK	ON DESIRED TRACK?
F404: F0 42	385	BEG SETPHASE	YES, ENERGIZE PHASE AND RETURN
F406: A9 00	386	LDA #0	
F408: B5 95	387	STA TRKCNT	HALFTRACK COUNT.
F40A: A5 8C	388	SEEK2 LDA CURTRK	SAVE CURTRK FOR
F40C: B5 9D	389	STA PRIOR	DELAYED TURNOFF.
F40E: 38	390	SEC	
F40F: E5 9E	391	SBC TRKN	DELTA-TRACKS.
F411: F0 31	392	BEG SEEKEND	BR IF CURTRK=DESTINATION
F413: B0 06	393	BCC OUT	(MOVE OUT. NOT IN)
F415: 49 FF	394	EOR #\$FF	CALC TRKS TO GO
F417: E6 8C	395	INC CURTRK	INCR CURRENT TRACK (IN)
F419: 90 04	396	BCC MINTST	(ALWAYS TAKEN).
F41B: 69 FE	397	OUT ADC #\$FF	CALC TRKS TO GO
F41D: C6 8C	398	DEC CURTRK	DECR CURRENT TRACK (OUT)
F41F: C5 95	399	MINTST CMP TRKCNT	
F421: 90 02	400	BCC MAXTST	AND TRKS MOVED
F423: A5 95	401	LDA TRKCNT	
F425: C9 09	402	MAXTST CMP #\$9	
F427: B0 02	403	BCC STEP2	IF TRKCNT=\$8 LEAVE Y ALONE (Y=\$8)
F429: A8	404	STEP1 LAY	ELSE SET ACCELERATION INDEX IN Y
F42A: 38	405	SEC	
F42B: 20 48 F4	406	STEP2 JSR SETPHASE	
F42E: B9 67 F4	407	LDA ONTABLE, Y	FOR 'ONTIME'
F431: 20 56 F4	408	JSR MSWAIT	(100 USEC INTERVALS)
F434: A5 9D	409	LDA PRIOR	
F436: 18	410	CLC	FOR PHASEOFF
F437: 20 4A F4	411	JSR CLRPHASE	TURN OFF PRIOR PHASE
F43A: B9 70 F4	412	LDA OFFTABLE, Y	THEN WAIT 'OFFTIME'
F43D: 20 56 F4	413	JSR MSWAIT	(100 USEC INTERVALS)
F440: E6 95	414	INC TRKCNT	'TRACKS MOVED' COUNT.
F442: D0 C6	415	BNE SEEK2	(ALWAYS TAKEN)
F444: 20 56 F4	416	SEEKEND JSR MSWAIT	SETTLE 25 MSEC
F447: 18	417	CLC	SET FOR PHASE OFF
F448: A5 8C	418	SETPHASE LDA CURTRK	GET CURRENT TRACK
F44A: 29 03	419	CLRPHASE AND #3	MASK FOR 1 OF 4 PHASES
F44C: 2A	420	ROL A	DOUBLE FOR PHASEON/OFF INDEX
F44E: C1 91	421	ORA IBDSLOT	
F44F: AA	422	TAX	
F450: BD 30 C0	423	LDA PHASEOFF, X	TURN ON/OFF ONE PHASE
F453: A6 81	424	LDX IBDSLOT	RESTORE X-REG
F455: 60	425	SEEKRTS RTS	AND RETURN
F456:	427	*****	
F456:	428 *		*
F456:	429 *	MSWAIT SUBROUTINE	*
F456:	430 *		*

	49	4,383,296	50
F456	431	*****	
F456	432	*	*
F456	433	* DELAYS A SPECIFIED	*
F456	434	* NUMBER OF 100 USEC	*
F456	435	* INTERVALS FOR NOTICE	*
F456	436	* ON-TIME	*
F456	437	*	*
F456	438	* ----- ON ENTRY -----	*
F456	439	*	*
F456	440	* A-REG HOLDS NUMBER	*
F456	441	* OF 100 USEC	*
F456	442	* INTERVALS TO	*
F456	443	* DELAY	*
F456	444	*	*
F456	445	* ----- ON EXIT -----	*
F456	446	*	*
F456	447	* A-REG HOLDS 100	*
F456	448	* A-REG HOLDS 100	*
F456	449	* X-REG ON-HAND	*
F456	450	* CARRY SET	*
F456	451	*	*
F456	452	* MONTIME, MONTIME	*
F456	453	* MONTIME, MONTIME	*
F456	454	* MONTIME, MONTIME	*
F456	455	* MONTIME, MONTIME	*
F456	456	* MONTIME, MONTIME	*
F456	457	* MONTIME, MONTIME	*
F456	458	* MONTIME, MONTIME	*
F456	459	* MONTIME, MONTIME	*
F456	460	* MONTIME, MONTIME	*
F456	461	* MONTIME, MONTIME	*
F456	462	* MONTIME, MONTIME	*
F456	463	* MONTIME, MONTIME	*
F456	464	* MONTIME, MONTIME	*
F456	465	* MONTIME, MONTIME	*
F456	466	* MONTIME, MONTIME	*
F456	467	* MONTIME, MONTIME	*
F456	468	* MONTIME, MONTIME	*
F456	469	* MONTIME, MONTIME	*
F456	470	* MONTIME, MONTIME	*
F456	471	* MONTIME, MONTIME	*
F456	472	* MONTIME, MONTIME	*
F456	473	* MONTIME, MONTIME	*
F456	474	*****	
F456	475	*	*
F456	476	* PHASE ON-, OFF-TIME	*
F456	477	* TABLES IN 100-USEC	*
F456	478	* INTERVALS. (SEEK)	*
F456	479	*	*
F456	480	*****	
F456	481	ONTABLE DFB 1, \$30, \$28	
F456	482	DFB \$24, \$20, \$1E	
F456	483	DFB \$1D, \$1C, \$1C	
F456	484	OFFTABLE DFB \$70, \$2C, \$26	
F456	485	DFB \$22, \$1F, \$1E	
F456	486	DFB \$1D, \$1C, \$1C	

	51	4,383,296	52
F479: 86 83	488	BLOCKIO STX	IBTRK
F47B: A0 05	489	LDY	##5
F47D: 48	490	PHA	
F47E: 0A	491	TRKSEC ASL	A
F47F: 26 83	492	ROL	IBTRK
F481: 88	493	DEY	
F482: D0 FA	494	BNE	TRKSEC
F484: 68	495	PLA	
F485: 29 07	496	AND	##7
F487: A8	497	TAY	
F488: B9 A0 F4	498	LDA	SECTABL, Y
F48B: 85 84	499	STA	IBSECT
F48D: 20 00 F0	500	JSR	REGRWTS
F490: B0 0B	501	BCS	QUIT
F492: E6 86	502	INC	IBBUFP+1
F494: E6 84	503	INC	IBSECT
F496: E6 84	504	INC	IBSECT
F498: 20 00 F0	505	JSR	REGRWTS
F49B: C6 86	506	DEC	IBBUFP+1
F49D: A5 88	507	QUIT LDA	IBSTAT
F49F: 60	508	RTS	
F4A0:	509	*	
F4A0:	510	SECTABL EQU	*
F4A0: 00 04 0B	511	DFB	\$0, \$4, \$8
F4A3: 00 01 05	512	DFB	\$0, \$1, \$5
F4A6: 09 0D	513	DFB	\$9, \$D
F4A6:	514	*	
F4A8:	516	* * * * *	
F4A8:	517	*	
F4A8:	518	* JOYSTICK READ ROUTINE	
F4A8:	519	*	
F4A8:	520	* * * * *	
F4A8:	521	* ENTRY ACC= COUNT DOWN HIGH	
F4A8:	522	* X&Y= DON'T CARE	
F4A8:	523	*	
F4A8:	524	* EXIT ACC= TIMER HIGH BYTE	
F4A8:	525	* Y= TIMER LOW BYTE	
F4A8:	526	* CARRY CLEAR	
F4A8:	527	*	
F4A8:	528	* IF CARRY SET, ROUTINE	
F4A8:	529	* WAS INTERRUPTED &	
F4A8:	530	* ACC & Y ARE INVALID	
F4A8:	531	* * * * *	
F4A8:	532	*	
FFD9:	533	TIMLATCH EQU	\$FFD9
FFD8:	534	TIMER1L EQU	\$FFD8
FFD9:	535	TIMER1H EQU	\$FFD9
0066:	536	JOYRDY EQU	\$0066
F4A8:	537	*	
F4A8:	538	ANALOG EQU	* CARRY SHOULD BE SET!
F4A8: 8D D9 FF	539	STA TIMLATCH	START THE TIMER!
F4A8: AD EF FF	540	ANLOG1 LDA	INTERUPT
F4A8: 2D 66 0D	541	AND JOYRDY	WAIT FOR ONE OR THE OTHER TO GO LOW
F4B1: 20 F6	542	BMI ANLOG1	
F4B3: AD 66 0D	543	LDA JOYRDY	WAY IT REALLY THE JOYSTICK?
F4B4: 30 0D	544	BMI GOODTIME	NOPE, FORGET IT
F4B8: 18	545	CLC	TIME'S A SLIP SLIDIN AWAY
F4B9: AD D9 FF	546	LDA TIMER1H	NOW, WHAT TIME IS IT?
F4BC: AD D8 FF	547	LDY TIMER1L	
F4BF: 10 03	548	BPL GOODTIME	TIME WAS VALID!
F4C1: AD D9 FF	549	LDA TIMER1H	HIGH BYTE CHANGED
F4C4: 60	550	GOODTIME RTS	
*** SUCCESSFUL ASSEMBLY. NO ERRORS			

4,383,296

53

54

FOE9 ALDONE1	FOE0 ALLDONE	F119 ALLOFF	?F4A8 ANALDG
F4AB ANLDG1	?F479 BLOCKIO	9D BUF	F12D CHKDRV1
F12B CHKDRV	F13D CKDRTS	96 CKSUM	F44A CLRPHASE
FO50 CONWAIT	FOC7 CORRECTSECT	?F0BF CORRECTVOL	95 COUNT
97 CSSTV	97 CSUM1	89 CSUM	8C CURTRK
F300 DNIBL	FO31 DRIVSEL	CO8A DRVOEN	85 DRVOTRK
?CO8B DRV1EN	FOE5 DRVERR	F13E DRVINDX	FO3D DRVWAIT
E0 DVMOT	FFDF ENVIRON	9F ENVTEMP	?FOAO GOCAL1
?FOA1 GOCAL	F4C4 GOODTIME	?F116 G0SEEK	F1BA GOSERV
FOE8 HNDLERR	90 HRDERRS	85 IBBUFP	87 IBCMD
82 IBDERR	82 IBDRVN	80 IBNODRV	? 83 IBDERR
84 IBSECT	81 IBSLQT	89 IBSMOD	88 IBSTAT
83 IBTRK	81 IBWPER	8B IMASK	FFEF INTERRUPT
8A IOBPDN	CO66 JOYRDY	95 LAST	F425 MAXTST
F41F MINTST	9A MONTIMEH	99 MONTIMEL	FO4E MOTOF
CO88 MOTOROFF	CO89 MOTORON	F458 MSW1	F461 MSW2
F456 MSWAIT	F105 MYSEEK	O200 NBUF1	O302 NBUF2
F355 NIBL	?FO60 NODRIVERR	FO8D NOINTR1	FOF3 NOINTR2
F2AC NOWRITE	F11B NXOFF	F470 OFFTABLE	FO44 OK
80 ONEMEG	F467 ONTABLE	F41B OUT	CO80 PHASEOFF
?CO81 PHASEON	?CO81 PHASON	?CO80 PHSOFF	F315 PNIBL1
F322 PNIBL2	F336 POST1	F338 POST2	F34C POSTERR
F311 POSTNIB16	F2CA PRENIB1	F2C6 PRENIB16	?F2E7 PRENIB2
F2E4 PRENIB3	F2F8 PRENIB4	9D PRIOR	CO8D Q6H
CO8C Q6L	CO8F Q7H	CO8E Q7L	F49D QUIT
F14D RD1	F157 RD2	F162 RD3	F16B RD4
F17E RDSA	F17D RD5	F192 RD6	F1A5 RD7
F1AF RDB	F1CB RDA1	F1D2 RDA2	F1DD RDA3
F1EA RDA4	F1F2 RDA5	F204 RDA6	F20E RDA7
F1BD RDADR16	F1EB RDAFLD	F1CD RDASN1	F1C1 RDASYN
F19D RDCKSUM	F1BB RDERR	F217 RDEXIT	FOA7 RDRIGHT
F148 READ16	FO00 REGRWTS	93 RETRYCNT	F152 RSYNC1
F14A RSYNC	FOBB RTTRK	F4A0 SECTABL	98 SECT
?F106 SEEK1	F40A SEEK2	94 SEEKCNT	F400 SEEK
F444 SEEKEND	?F455 SEEKRTS	F2B3 SERVICE	F34C SETIMEG
F448 SETPHASE	F125 SETTRK	F354 SEV	F42B STEP2
?F429 STEP	97 TEMP	FFD9 TIMER1H	FFD8 TIMER1L
FFD9 TIMLATCH	99 TRACK	95 TRKCNT	9E TRKN
99 TRKN1	F47E TRKSEC	FO86 TRYADR2	FO7F TRYADR
FO7B TRYTRK2	FO65 TRYTRK	7F TWOMEG	9A VOLUME
F256 VRYFRST	F271 WDATA2	F281 WDATA3	F218 WEXIT
F24E WINTRPT	F26A WMIDLE	?F2BF WNIBL	F2BD WNIBL7
F2BC WNIBL9	F267 WNTTRPT1	F295 WRCKSUM	F219 WRITE16
FOF9 WRIT	?F223 WRT1	F258 WRTFRST	F230 WSYNC
7F TWOMEG	80 IBNODRV	80 HRDERRS	80 ONEMEG
81 IBSLQT	81 IBWPER	82 IBDERR	82 IBDRVN
83 IBRERR	83 IBTRK	84 IBSECT	85 DRVOTRK
85 IBBUFP	87 IBCMD	88 IBSTAT	89 CSUM
89 IBSMOD	8A IOBPDN	8D IMASK	8C CURTRK
93 RETRYCNT	94 SEEKCNT	95 LAST	95 TRKCNT
95 COUNT	96 CKSUM	97 CSSTV	97 CSUM1
97 TEMP	98 SECT	99 MONTIMEL	99 TRKN1
99 TRACK	9A MONTIMEH	9A VOLUME	9B BUF
9D PRIOR	9E TRKN	9F ENVTEMP	E0 DVMOT
O200 NBUF1	O302 NBUF2	CO66 JOYRDY	?CO80 PHSOFF
CO80 PHASEOFF	?CO81 PHASON	?CO81 PHASEON	CO88 MOTOROFF
CO89 MOTORON	CO8A DRVOEN	?CO8D DRV1EN	CO8C Q6L
CO8D Q6H	CO8E Q7L	CO8F Q7H	FO00 REGRWTS
FO31 DRIVSEL	FO3D DRVWAIT	FO44 OK	FO4E MOTOF
FO50 CONWAIT	?FO60 NODRIVERR	FO65 TRYTRK	FO7B TRYTRK2
FO7F TRYADR	FO86 TRYADR2	FO8B NOINTR1	?FOAO GOCAL1
?FOA1 GOCAL	FOA7 RDRIGHT	FOBB RTTRK	?F0BF CORRECTVOL
FOC7 CORRECTSECT	FOE0 ALLDONE	FOE5 DRVERR	FOE8 HNDLERR
FOE9 ALDONE1	FOF3 NOINTR2	FOF9 WRIT	F105 MYSEEK
?F106 SEEK1	?F116 G0SEEK	F119 ALLOFF	F11B NXOFF
F125 SETTRK	F12B CHKDRV	F12D CHKDRV1	F13D CKDRTS



0000  
0000  
0000  
0000  
0000  
0000  
0000  
0000  
0000  
0000  
0001  
0000  
0010  
0018  
0019  
001A  
0087  
0085  
0091  
F479  
0058  
00FF  
1419  
1810  
C000  
C008  
C010  
C056  
C047  
C050  
C051  
C066  
C0D0  
C0F1  
C0F2  
C0F3  
C100  
C200  
C300  
C400  
CFFF  
FFD0  
FFDF

```

2 *****
3 *
4 *SARA DIAGNOST
5 *
6 *DECEMBER 19.1
7 * BY
8 *W. BROEDNER &
9 *
10 *COPYRIGHT 197
11 *
12 *****
13 RCM EQU $
14 ZRPG EQU $
15 ZRPG1 EQU $
16 PTRLO EQU Z
17 PTRHI EQU Z
18 RNX EQU Z
19 1BCMD EQU $
20 IBUFF EQU $
21 PREVTRK EQU $
22 BLOCK10 EQU $
23 CV EQU $
24 STR EQU $
25 IDNA EQU $
26 PHP EQU $
27 KYPD EQU $
28 KEYBD EQU $
29 KBDSTRB EQU $
30 POLEN EQU $
31 ADRS EQU $
32 GRMD EQU $
33 TXMD EQU $
34 ADTO EQU $
35 DISKOFF EQU $
36 ACIAST EQU $
37 ACIAOM EQU $
38 ACIAON EQU $
39 SLT1 EQU $
40 SLT2 EQU $
41 SLT3 EQU $
42 SLT4 EQU $
43 EXPROM EQU $
44 ZPREG EQU $
45 SYSD1 EQU $

```

```

*****
BI ROUTINES
HLEH
APPLE COMPUTER
*****
FOR RAM M17
B
9
$A

PTRHI
ZRPG1

```

```

F14D RD1
F16B RD4
F19D RDCKSUM
F1BA GOSERV
F1CD RDASN1
F1EA RDA4
F217 RDEXIT
F230 WSYNC
F267 WNTRPT1
F295 WRCKSUM
F2BD WNIBL7
F2EA PRENIB3
F311 POSTNIB16
F338 POST2
F355 NIBL
F41F MINTST
F444 SEEKEND
F456 MSWAIT
F470 OFFTABLE
F4A0 SECTABL
FFD8 TIMER1L
FFEF INTERUPT

```

4,383,296

57

58

```

FFD2: 46 SYSD2 EQU $FFD2
FFD3: 47 SYSD3 EQU $FFD3
FFE0: 48 SYSE0 EQU $FFE0
FFE1: 49 BNKSW EQU $FFE1
FFE2: 50 SYSE2 EQU $FFE2
FFE3: 51 SYSE3 EQU $FFE3
FC25: 52 COUT EQU $FC25
FD07: 53 CROUT1 EQU $FD07
FD0F: 54 KEYIN EQU $FD0F
FBC7: 55 SETCVH EQU $FBC7
FD98: 56 CLDSTRT EQU $FD98
FD9D: 57 SETUP EQU $FD9D
F901: 58 MONITOR EQU $F901
0000: 59 *

```

----- NEXT OBJECT FILE NAME IS DIAG.OBJ

```

F4C5: 60 ORG $F4C5
F4C5:00 B1 B2 61 RAMTBL DFB $0, $B1, $B2, $BA, $B9, $10, $0, $13
F4C8: DA B9 10
F4CD:00 13
F4CD: 62 CHPG EQU *
F4CD:52 41 CD 63 DCI 'RAM'
F4D0:52 4F CD 64 DCI 'ROM'
F4D3:56 49 C1 65 DCI 'VIA'
F4D6:41 43 49 66 DCI 'ACIA'
F4D9:C1
F4DA:41 2F C4 67 DCI 'A/D'
F4DD:44 49 41 68 DCI 'DIAGNOSTIC'
F4E0:47 4E 4F
F4E3:53 54 49
F4E6:C3
F4E7:5A D0 69 DCI 'ZF'
F4E9:52 45 54 70 DCI 'RETRY'
F4EC:52 D9
F4EE: 71 *
F4EE: 72 * SETUP SYSTEM
F4EE: 73 *
F4EE: 74 *
F4EE:A9 53 75 LDA ##52+ROM TURN OFF SCREEN, SET 2MHZ SPEED
F4F0:8D DF FF 76 STA SYSD1 AND RUN OFF ROM
F4F3:A2 00 77 LDX ##00 SET BANK SWITCH TO ZERO
F4F5:8E E0 FF 78 STX SYSE0
F4F8:8E EF FF 79 STX BNKSW
F4FB:8E D0 FF 80 STX ZPREG AND SET ZERO PAGE SAME
F4FE:CA 81 DEX
F4FF:8E D2 FF 82 STX SYSD2 PROGRAM DDR'S
F502:8E D3 FF 83 STX SYSD3
F505:9A 84 TXS
F506:EB 85 INX
F507:A9 0F 86 LDA ##0F
F509:8D E3 FF 87 STA SYSE3
F50C:A9 3F 88 LDA ##3F
F50E:8D E2 FF 89 STA SYSE2
F511:A0 06 90 LDY ##06
F513:D9 D0 C0 91 DISK1 LDA DISKOFF,Y
F516:88 92 DEY
F517:88 93 DEY
F518:10 F9 94 BPL DISK1
F51A:AD 08 C0 95 LDA KEYBD
F51D:29 04 96 AND ##04
F51F:D0 03 97 BNE NXBYT
F521:4C 89 F6 98 JMP RECON
F524: 99 *
F524: 100 * VERIFY ZERO PAGE
F524: 101 *

```

59		4,383,296		60
F524: A9 01	102 NXBYT	LDA ##01		ROTATE A 1 THROUGH
F526: 95 00	103 NXBIT	STA ZRPG,X		EACH BIT IN THE 0 PG
F528: D5 00	104	CMP ZRPG,X		TO COMPLETELY TEST
F52A: D0 FE	105 NOGOOD	BNE NOGOOD		THE PAGE. HANG IF NOGOOD
F52C: 0A	106	ASL A		TRY NEXT BIT OF BYTE
F52D: D0 F7	107	BNE NXBIT		UNTIL BYTE IS ZERO.
F52F: E8	108	INX		CONTINUE UNTIL PAGE
F530: D0 F2	109	BNE NXBYT		IS DONE.
F532:	110 *			
F532: 8A	111 CNTWR	TXA		PUSH A DIFFERENT
F533: 48	112	PHA		BYTE ONTO THE
F534: E8	113	INX		STACK UNTIL ALL
F535: D0 FB	114	BNE CNTWR		STCK BYTES ARE FULL.
F537: CA	115	DEX		THEN PULL THEM
F538: 86 18	116	STX PTRLO		OFF AND COMPARE TO
F53A: 68	117 PULBT	PLA		THE COUNTER GOING
F53B: C5 18	118	CMP PTRLO		BACKWARDS. HANG IF
F53D: D0 E8	119	BNE NOGOOD		THEY DON'T AGREE.
F53F: C6 18	120	DEC PTRLO		GET NEXT COUNTER BYTE
F541: D0 F7	121	BNE PULBT		CONTINUE UNTIL STACK
F543: 68	122	PLA		IS DONE. TEST LAST BYTE
F544: D0 E4	123	BNE NOGOOD		AGAINST ZERO.
F546:	124 *			
F546:	125 * SIZE THE MEMORY			
F546:	126 *			
F546: A2 08	127	LDX ##08		ZERO THE BYTES USED TO DISPLAY
F548: 95 10	128 NOMEM	STA ZRPG1,X		THE BAD RAM LOCATIONS
F54A: CA	129	DEX		EACH BYTE= A CAS LINE
F54D: 10 FB	130	BPL NOMEM		ON THE SARA BOARD.
F54D:	131 *			
F54D: A2 02	132	LDX ##02		STARTING AT PAGE 2
F54F: 86 19	133 NMEM1	STX PTRHI		TEST THE LAST BYTE
F551: A9 00	134	LDA ##00		IN EACH MEM PAGE TO
F553: A0 FF	135	LDY ##FF		SEE IF THE CHIPS ARE
F555: 91 18	136	STA (PTRLO),Y		THERE. (AVOID 0 & STK PAGES)
F557: D1 18	137	CMP (PTRLO),Y		CAN THE BYTE BE 0'D?
F559: F0 07	138	BEG NMEM2		
F55B: 20 48 F7	139	JSR RAM		NO, FIND WHICH CAS IT IS.
F55E: 94 10	140	STY ZRPG1,X		SET CORRES. BYTE TO FF
F560: A6 19	141	LDX PTRHI		RESTORE X REGISTER
F562: E8	142 NMEM2	INX		AND INCREMENT TO NEXT
F563: E0 C0	143	CPX ##C0		PAGE UNTIL I/O IS REACHED.
F565: D0 E8	144	BNE NMEM1		
F567: A2 20	145	LDX ##20		THEN RESET TO PAGE 20
F569: EE EF FF	146	INC BNKSW		AND GOTO NEXT BANK TO
F56C: AD EF FF	147	LDA BNKSW		CONTINUE. (MASK INPUTS
F56F: 29 0F	148	AND ##0F		FROM BANKSWITCH TO SEE
F571: C9 03	149	CMP ##03		WHAT SWITCH IS SET TO)
F573: D0 DA	150	BNE NMEM1		CONTINUE UNTIL BANK '3'
F575:	151 *			
F575:	152 * SETUP SCREEN			
F575: 20 9D FD	153 ERRLP	JSR SETUP		CALL SCRNM SETUP ROUTINE
F578: A2 00	154	LDX ##00		SETUP I/O AGAIN
F57A: 8E E0 FF	155	STX SYSE0		FOR VIA TEST
F57D: CA	156	DEX		PROGRAM DATA DIR
F57E: 8E D2 FF	157	STX SYSD2		REGISTERS
F581: 8E D3 FF	158	STX SYSD3		
F584: A9 3F	159	LDA ##3F		
F586: 8D E2 FF	160	STA SYSE2		
F589: A9 0F	161	LDA ##0F		
F58B: 8D E3 FF	162	STA SYSE3		
F58E: A2 10	163	LDX ##10		HEADING OF 'DIAGNOSTICS' WITH
F590: 20 38 F7	164	JSR STRWT		THIS SUBROUTINE
F593: A2 00	165 ERRLP1	LDX ##00		PRINT 'RAM'
F595: 86 5D	166	STX CV		SET CURSOR TO 2ND LINE
F597: A9 04	167	LDA ##04		SPACE CURSOR OUT 3

4,383,296

	61		62
F399: 20 C7 F8	168	JSR SETCVH	(X STILL=0 ON RETURN)
F39C: 20 38 F7	169	JSR STRWT	THE SAME SUBROUTINE
F39F: A2 07	170	LDX #07	FOR BYTES 7 - 0 IN
F5A1:	171 RAMWT1	EQU *	
F5A1: 85 10	172	LDA ZRPG1,X	OUT EACH BIT AS A
F5A3: A0 08	173	LDY #08	' ' OR '1' FOR INDICATE BAD OR MISSING
F5A5: 0A	174 RAMWT2	ASL A	CHIPS SUBROUTINE 'RAM' RAM
F5A6: 48	175	PHA	SETS UP THESE BYTES
F5A7: A9 AE	176	LDA #0AE	LOAD A ' ' TO ACC
F5A9: 90 02	177	BCC RAMWT4	
F5AB: A9 31	178	LDA #031	LOAD A '1' TO ACC
F5AD: 20 25 FC	179 RAMWT4	JSR COUT	AND PRINT IT
F5D0: 68	180	PLA	RESTORE BYTE
F5B1: 88	181	DEY	AND ROTATE ALL 8
F5B2: D0 F1	182	BNE RAMWT2	TIMES
F5B4: 20 07 FD	183	JSR CROUT1	CLEAR TO END OF LINE
F5B7: CA	184	DEY	
F5B8: 10 E7	185	BPL RAMWT1	
F5BA:	186 *		
F5BA:	187 * ZPG&STK TEST		
F5BA:	188 *		
F5BA: 9A	189	TXS	
F5BB: 8C EF FF	190	STY BNKSW	
F5BE: 98	191 ZP1	TYA	
F5BF: 8D D0 FF	192	STA ZPREG	
F5C2: 85 FF	193	STA STKO	
F5C4: C8	194	INY	
F5C5: 98	195	TYA	
F5C6: 48	196	PHA	
F5C7: 68	197	PLA	
F5C8: C8	198	INY	
F5C9: C0 20	199	CPY #020	
F5CB: D0 F1	200	BNE ZP1	
F5CD: A0 C0	201	LDY #0C0	
F5CF: 8C D0 FF	202	STY ZPREG	
F5D2: 86 18	203	STX PTRLO	
F5D4: EB	204 ZP2	INX	
F5D5: 86 19	205	STX PTRHI	
F5D7: 8A	206	TXA	
F5D8: D1 18	207	CMP PTRLO,X	
F5DA: D0 06	208	BNE ZP3	
F5DC: E0 1F	209	CPX #01F	
F5DE: D0 F4	210	BNE ZP2	
F5E0: F0 05	211	BEG ROMTST	
F5E2:	212 ZP3	EQU *	CHIP IS THERE, BAD ZERO AND STACK
F5E2: A2 1A	213	LDX #01A	SO PRINT 'ZP' MESSAGE
F5E4: 20 7B FF	214	JSR MESSERR	& SET FLAG (2MHZ MODE)
F5E7:	215 *		
F5E7:	216 * ROM TEST ROUTINE		
F5E7:	217 *		
F5E7: A9 00	218 ROMTST	LDA #000	SET POINTERS TO
F5E9: A8	219	TAY	\$F000
F5EA: A2 F0	220	LDY #0FC	
F5EC: 85 18	221	STA PTRLO	
F5EE: 86 19	222	STX PTRHI	SET X TO \$FF
F5F0: A2 FF	223	LDX #0FF	FOR WINDOWING I/O
F5F2: 81 18	224 ROMTST1	EOR (PTRLO),Y	COMPUTE CHKSUM ON
F5F4: E4 19	225	CPY PTRHI	EACH ROM BYTE,
F5F6: D0 06	226	BNE ROMTST2	WINDOW OUT
F5F8: C0 BF	227	CPY #0BF	RANGES FFC0-FFEF
F5FA: D0 02	228	BNE ROMTST2	
F5FC: A0 EF	229	LDY #0EF	
F5FE: C8	230 ROMTST2	INY	
F5FF: D0 F1	231	BNE ROMTST1	
F601: E6 19	232	INC PTRHI	
F603: D0 ED	233	BNE ROMTST1	
F605: A8	234	TAY	TEST ACC. FOR 0
F606: F0 05	235	BEG VIATST	YES, NEXT TEST
F608: A2 03	236	LDX #003	PRINT 'ROM' AND
F60A: 20 7B F7	237	JSR MESSERR	SET ERROR
F60D:	238 *		

4,383,296

63

64

```

F60D: 239 * VIA TEST ROUTINE
F60D: 240 *
F60D: 18 241 VIATST CLC SET UP FOR ADDING BYTES
F60E: D8 242 CLD
F60F: AD E0 FF 243 LDA SYSEO MASK OFF INPUT BITS
F612: 29 3F 244 AND ##3F AND STORE BYTE IN
F614: 85 18 245 STA PTRLO TEMPOR. LOCATION
F616: AD EF FF 246 LDA BNKSW MASK OFF INPUT BITS
F619: 29 4F 247 AND ##4F AND ADD TO STORED
F61B: 65 18 248 ADC PTRLO BYTE IN TEMP. LOC.
F61D: 6D D0 FF 249 ADC ZPREG ADD REMAINING
F620: 85 18 250 STA PTRLO REGISTERS OF THE
F622: AD DF FF 251 LDA SYSD1 VIA'S
F625: 29 5F 252 AND ##5F (MASK THIS ONE)
F627: 65 18 253 ADC PTRLO AND TEST
F629: 6D D2 FF 254 ADC SYSD2 TO SEE
F62C: 6D D3 FF 255 ADC SYSD3 IF THEY AGREE
F62E: 6D E2 FF 256 ADC SYSE2 WITH THE RESET
F632: 6D E3 FF 257 ADC SYSE3 CONDITION.
F635: C9 E1 258 CMP ##E0+ROM =E1?
F637: F0 05 259 BEQ ACIA YES, NEXT TEST
F639: A2 06 260 LDX ##06 NO, PRINT 'VIA' MESS.
F63B: 20 7B F7 261 JSR MESSERR AND SET ERROR FLAG
F63E: 262 *
F63E: 263 * ACIA TEST ROUTINE
F63E: 264 *
F63E: 18 265 ACIA CLC SETUP FOR ADDITION
F63F: A9 9F 266 LDA ##9F MASK INPUT BITS
F641: 2D F1 C0 267 AND ACIAST FROM STATUS REG
F644: 6D F2 C0 268 ADC ACIACM AND ADD DEFAULT STATES
F647: 6D F3 C0 269 ADC ACIACN OF CONTROL AND COMMND
F64A: C9 10 270 CMP ##10 REGS. =10?
F64C: F0 05 271 BEQ ATD YES, NEXT TEST
F64E: A2 09 272 LDX ##09 NO, 'ACIA' MESSAGE AND
F650: 20 7B F7 273 JSR MESSERR THEN SET ERROR FLAG
F653: 274 *
F653: 275 * A/D TEST ROUTINE
F653: 276 *
F653: A9 C0 277 ATD LDA ##C0
F655: 8D DC FF 278 STA $FFDC
F658: AD 5A C0 279 LDA PDLEN+2
F65B: AD 5E C0 280 LDA PDLEN+6
F65E: AD 5C C0 281 LDA PDLEN+4
F661: A0 20 282 LDY ##20
F663: 88 283 ADCTST1 DEY WAIT FOR 40 USEC
F664: D0 FD 284 BNE ADCTST1
F666: AD 5D C0 285 LDA PDLEN+5 SET A/D RAMP
F669: C8 286 ADCTST3 INY COUNT FOR CONVERSION
F66A: F0 0A 287 BEQ ADCERR (255=ERROR)
F66C: AD 66 C0 288 LDA ADFO IF BIT 7 =1?
F66F: 30 FB 289 BMI ADCTST3 YES, CONTINUE
F671: 98 290 TYA NO, MOVE COUNT TO ACC
F672: 29 E0 291 AND ##E0 ACC<32?
F674: F0 05 292 BEQ KEYPLUG
F676: 293 ADCERR EQU * NO,
F676: A2 0D 294 LDX ##0D PRINT 'A/D' MESS
F678: 20 7B F7 295 JSR MESSERR AND SET ERROR FLAG
F67B: 296 *
F67B: 297 * KEYBOARD PLUGIN TEST
F67B: 298 *
F67B: AD 08 C0 299 KEYPLUG LDA KEYBD IS KYBD PLUGGED IN?
F67E: 0A 300 ASL A (IS LIGHT CURRENT

```

4,383,296

65

66

F67F: 10 41	301	BPL SEX	PRESENT?) NO, BRANCH
F681: AD DF FF	302	LDA SYSD1	IS ERROR FLAG SET?
F684: 10 03	303	BPL RECON	(2MHZ MODE) NO, BRANCH
F686: 4C 93 F3	304	JMP ERRLP1	ERROR, HANG
F689:	305 *		
F689:	306 *	RECONFIGURE SYSTEM	
F689:	307 *		
F689:	308	RECON EQU *	
F689: A9 77	309	LDA #\$77	TURN ON SCREEN
F68B: 8D DF FF	310	STA SYSD1	
F68E: 20 98 FF	311	JSR CLDSTRT	INITIALIZE MONITOR AND DEFAULT CHARACTER SET
F691: A9 10	312	LDA #\$10	TEST FOR "APPLE 1"
F693: 2D 08 C0	313	AND KEYBD	
F696: D0 09	314	BNE BOOT	NO, DO REGULAR BOOT
F698: 2C 10 C0	315	BIT KBDSTRB	CLEAR KEYBOARD
F69B: AD 50 C0	316	LDA GRMD	
F69E: 20 01 FF	317	JSR MONITOR	AND NEVER COME BACK.
F6A1: A2 01	318	LDX #1	READ BLOCK 0
F6A3: 86 B7	319	STX IBOND	
F6A5: CA	320	DEX	
F6A6: 86 B5	321	STX IBBUF	INTO RAM AT \$A000
F6A8: A9 A0	322	LDA #\$A0	
F6AA: B5 B6	323	STA IBBUF+1	
F6AC: 4A	324	LSR A	FOR TRACK 80
F6AD: B5 91	325	STA PREVTRK	MAKE IT RECALIBRATE TOO!
F6AF: 8A	326	TXA	
F6B0: 20 79 F4	327	JSR BLOCKIO	
F6B3: 90 0A	328	BCC GOBOOT	IF WE'VE SUCCEEDED, DO IT UP
F6B5: A2 1C	329	LDX #\$1C	
F6B7: 20 38 F7	330	JSR STRNT	RETRY
F6BA: 20 0F FD	331	JSR KEYIN	
F6BD: B0 E2	332	BCC BOOT	
F6BF: 4C 00 A0	333	JMP GOBOOT	GO TO IT FOOL...
F6C2:	334 *		
F6C2:	335 *	SYSTEM EXERCISER	
F6C2:	336 *		
F6C2: A0 7F	337	SEX LDY #\$7F	TRYFROM
F6C4: 98	338	SEX1 TYA	7F TO 0
F6C5: 29 FE	339	AND #\$FE	ADD =
F6C7: 49 4E	340	EOR #\$4E	4EOR4F?
F6C9: FC 03	341	BEG SEX2	YES, SKP
F6CB: B9 00 C0	342	LDA KYBD, Y	NO, CONT
F6CE: B9	343	SEX2 DEY	NXT ADD
F6CF: D0 F3	344	BNE SEX1	
F6D1: AD 51 C0	345	LDA TXTMD	SET TXT
F6D4: B9 00 C1	346	SEX3 LDA SLT1, Y	EXERCSE
F6D7: B9 00 C2	347	LDA SLT2, Y	ALL
F6DA: B9 00 C3	348	LDA SLT3, Y	SLOTS
F6DD: B9 00 C4	349	LDA SLT4, Y	
F6E0: AD FF CF	350	LDA EXPROM	DISABLE EXPANSION ROM AREA
F6E3: C8	351	INY	
F6E4: D0 EE	352	BNE SEX3	
F6E6:	353 *		
F6E6:	354 *	RAM TEST ROUTINE	
F6E6:	355 *		
F6E6: A9 73	356	USRENTY LDA #\$72+ROM	
F6E8: 8D DF FF	357	STA SYSD1	
F6EB: A9 18	358	LDA #\$18	
F6ED: 8D D0 FF	359	STA ZPREG	
F6F0: A9 00	360	LDA #\$00	
F6F2: A2 07	361	LDX #\$07	
F6F4: 95 10	362	RAMTSTO STA ZRPG1, X	
F6F6: CA	363	DEX	
F6F7: 10 FB	364	BPL RAMTSTO	
F6F9: 20 84 F7	365	JSR RAMSET	
F6FC: 08	366	PHP	
F6FD: 20 F7 F7	367	RAMTST1 JSR RAMWT	
F700: 20 F7 F7	368	JSR RAMWT	

	67	4,383,296	68
F703: 2B	369	PLP	
F704: 6A	370	ROR A	
F705: 08	371	PHP	
F706: 20 A1 F7	372	JSR PTRINC	
F709: D0 F2	373	BNE RAMTST1	
F70B: 20 84 F7	374	JSR RAMSET	
F70E: 08	375	PHP	
F70F: 20 FB F7	376	JSR RAMRD	
F712: 48	377	PHA	
F713: A9 00	378	LDA **00	
F715: 91 18	379	STA (PTRLO),Y	
F717: 68	380	PLA	
F718: 28	381	PLP	
F719: 6A	382	ROR A	
F71A: 08	383	PHP	
F71B: 20 A1 F7	384	JSR PTRINC	
F71E: D0 EF	385	BNE RAMTST4	
F720	386 *		
F720	387 *	RETURN TO START	
F720	388 *		
F720 A9 00	389	LDA **00	
F722: 8D EF FF	390	STA BNKSW	
F725: 8D D0 FF	391	STA ZPREG	
F72B: A2 07	392	LDX **07	
F72A: BD 10 18	393	RAMTST6 LDA PHP,X	
F72D: 95 10	394	STA ZRPG1,X	
F72F: CA	395	DEX	
F730: 10 FB	396	BPL RAMTST6	
F732: 20 7E F7	397	JSR ERROR	
F735: 4C 75 F5	398	JMP ERRLP	
F738	399	*****	
F738	400	* SARA TEST SUBROUTINES	
F738	401	*****	
F738	402	*	
F738	403	* SUBROUTINE STRING WRITE	
F738	404	*	
F738: BD CD F4	405	STRWT LDA CHPG,X	
F73B: 48	406	PHA	
F73C: 09 8D	407	ORA **80	NORMAL VIDEO
F73D: 20 25 F7	408	JSR COUT	& PRNT
F741: ED	409	INX	NXT
F742: 5B	410	PLA	CHR
F743: 10 FB	411	BPL STRWT	
F745: 4C 07 FD	412	JMP CROUT1	CLR TO END OF LINE
F748	413	*	
F748	414	* SUBROUTINE RAM	
F748	415	*	
F748: 48	416	RAM PHA	SV ACC
F749: 8A	417	TXA	CONVRT
F74A: 4A	418	LSR A	ADD TO
F74B: 4A	419	LSR A	USE FOR
F74C: 4A	420	LSR A	B ENTRY
F74D: 4A	421	LSR A	
F74E: 08	422	PHP	
F74F: 4A	423	LSR A	
F750: 28	424	PLP	
F751: AA	425	TAX	LOOKUP
F752: BD CD F4	426	LDA RAMTBL,X	IF VAL

			4,383,296			
			69	70		
F755	10	14	427	BPL	RAM0	CO, GET
F757	48		428	FHA		WHICH
F758	AD	EF FF	429	LDA	BANKSW	
F75B	29	0F	430	AND	##0F	
F75D	AA		431	TAX		
F75E	68		432	PLA		
F75F	E0	00	433	CPX	##00	
F761	F0	13	434	BEG	RAM1	BANKSW
F763	4A		435	LSR	A	SET
F764	4A		436	LSR	A	PROPER
F765	4A		437	LSR	A	RAM
F766	0A		438	BEQ		VALUE
F767	D0	00	439	INB	RAM1	
F769	29	05	440	ADD	##05	CONVERT
F76B	D0	19	441	RMB	RAM1	TO VAL
F76D	8A		442	TXA		
F76F	F0	02	443	BEG	RAM00	
F770	A9	03	444	CPX	#3	
F772	90	02	445	RAM00	RAM1	
F774	49	03	446	FOR	#3	
F776	29	07	447	RAM1	AND	BANKSW
F778	AA		448	TAX		
F779	68		449	PLA		
F77A	60		450	RTS		
F77B			451	*		
F77C			452	* SUBROUTINE	ERROR	
F77B			453	*		
F77B	20	08 FF	454	MESSAGE	USR	STRWT
F77E	A9	F3	455	ERROR	LDA	##F2+ROM
F780	8D	DE FF	456		STA	SYS01
F782	60		457		RTS	
F784			458	*		
F784			459	* SUBROUTINE	RAMSET	
F784			460	*		
F784	A2	01	461	RAMSET	LDX	##01
F786	86	1A	462		STX	BNK
F788	A0	00	463		LDY	##00
F78A	A9	AA	464		LDA	##AA
F78C	38		465		SEC	
F78E	48		466	RAMSET	FHA	
F78F	02		467		PHP	
F78F	A5	1A	468		LDA	BNK
F791	09	80	469		ORA	##80
F793	8D	19 14	470		STA	IDNK
F796	A9	02	471		LDA	##02
F798	85	19	472		STA	PTRHI
F79A	A2	00	473		LDX	##00
F79C	86	18	474		STX	PTRLO
F79E	28		475		PLP	
F79F	68		476		PLA	
F7A0	60		477		RTS	
F7A1			478	*		
F7A1			479	* SUBROUTINE	PTRINC	
F7A1			480	*		
F7A1	48		481	PTRINC	PHA	
F7A1	48					



		71	4,383,296	72
F7A2	E6 18	482	INC	PTRLO
F7A4	D0 1D	483	BNE	RETS
F7A6	A5 1A	484	LDA	BNK
F7A8	10 0E	485	BPL	PINCL
F7AA	A5 19	486	LDA	PTRHI
F7AC	C9 13	487	CMF	##13
F7AE	F0 06	488	BEG	PINC2
F7B0	C9 17	489	CMF	##17
F7B2	D0 04	490	BNE	PINCL
F7B4	EA 12	491	INC	PTRHI
F7B6	06 1F	492	PINCL	INC
F7B8	E6 19	493	PINCL	INC
F7BA	D0 07	494	BNE	RETS
F7BC	06 1A	495	DEC	BNK
F7BE	06 1A	496	DEC	BNK
F7C0	27 0D 07	497	JSR	RAMSET1
F7C2	68	498	PRTO	
F7C4	A5 1A	499	LDA	BNK
F7C6	E0 FD	500	LRA	##FD
F7C8	60	501	RTS	
F7CA		502	*	
F7CC		503	* SUBROUTINE	RAMERR
F7CE		504	*	
F7D0	46	505	RAMERR	PHA
F7D2	A5 19	506	LDX	PTRHI
F7D4	A4 1A	507	LDY	BNK
F7D6	60	508	PLA	
F7D8	A5 19	509	LDX	RAMERR4
F7DA	6A	510	LDY	
F7DC	30 1D	511	BPL	RAMERR5
F7DE	18	512	PLC	
F7E0	59 20	513	ADC	##20
F7E2	80 EF FF	514	RAMERR2	STX
F7E4	AA	515	TAX	
F7E6	20 4B F7	516	RAMERR3	JSR
F7E8	68	517	PLA	
F7EA	48	518	PHA	
F7EC	AD 07	519	LDY	##00
F7EE	51 18	520	LDX	(PTRLO), Y
F7F0	15 10	521	ORA	ZRPG1, X
F7F2	95 10	522	STA	ZRPG1, X
F7F4	68	523	PLA	
F7F6	60	524	RTS	
F7F8	A5 07	525	RAMERR4	LDA
F7FA	30 FF FF	526	STA	BNKSW
F7FC	F0 EA	527	BEG	RAMERR3
F7FE	38	528	RAMERR5	SEC
F7FF	E9 60	529	SBC	##60
F801	0B	530	INX	
F803	D0 80	531	BNE	RAMERR3
F805		532	*	
F807		533	* SUBROUTINE	RAMWT
F809		534	*	
F80B	40 FF	535	RAMWT	EOR
F80D	07 18	536	STA	(PTRLO), Y

4,383,296

73

74

FFFF 01 18  
FFFF 00 CA  
FFFF 60

537 RAMRD      MP      PTRLO, Y  
538              BNE      RAMERR  
539              RTS

\*\*\* SUCCESSFUL ASSEMBLY NO ERRORS

COF3 ACIACN	F63E ACIA	COF2 ACIACM	COF1 ACIAST
F676 ADCERR	F663 ADCTST1	F667 ADCTST3	?C047 ADRS
C066 ADTO	F653 ATD	F479 BLOCKIO	1A BNK
FFEF BNKSW	F6A1 BDOT	F4CD CHPG	FD98 CLDSTR1
F532 CNTWR	FC25 COUT	FD07 CROUT1	5D CV
F513 DISK1	C0D0 DISKOFF	F575 ERRLP	F593 ERRLP1
F77E ERROR	CFFF EXPROM	F6BF GOBOOT	C050 GRMD
85 IBUFFP	87 IBCMD	1419 IBNK	C010 KBDSTRB
C008 KEYBD	FD0F KEYIN	F67D KEYPLUG	C000 KYBD
F77B MESSERR	F901 MONITOR	F54F NMEM1	F562 NMEM2
F52A NOGOOD	F548 NOMEM	F526 NXBIT	F524 NXBYT
C058 PDLEN	1810 PHP	F7B8 PINC1	F7B6 PINC2
91 PREVTRK	19 PTRHI	F7A1 PTRINC	18 PTRLO
F53A PULBT	F772 RAM00	F776 RAM1	F748 RAM
F76B RAMO	F7DB RAMERR3	F7EA RAMERR4	F7C9 RAMERR
F7D7 RAMERR2	F7F1 RAMERR5	F7FB RAMRD	F784 RAMSET
F78D RAMSET1	F4C5 RAMTBL	F6F4 RAMTST0	F6FD RAMTST1
F70F RAMTST4	F72A RAMTST6	F5A1 RAMWT1	F5AD RAMWT4
F7F7 RAMWT	F5A5 RAMWT2	F689 RECON	F7C3 RETS
F5F2 ROMTST1	F5FE ROMTST2	F5E7 ROMTST	01 ROM
FBC7 SETCVH	FD9D SETUP	F6C4 SEX1	F6C2 SEX
F6CE SEX2	F6D4 SEX3	C100 SLT1	C200 SLT2
C300 SLT3	C400 SLT4	FF STKO	F738 STRWT
FFDF SYSD1	FFD2 SYSD2	FFD3 SYSD3	FFE0 SYSE0
FFE2 SYSE2	FFE3 SYSE3	C051 TXTMD	?F6E6 USRENTY
F60D VIATST	F5BE ZP1	F5D4 ZP2	F5E2 ZP3
FFD0 ZPREG	10 ZRPG1	00 ZRPG	18 PTRLO
00 ZRPG	01 ROM	10 ZRPG1	85 IBUFFP
19 PTRHI	1A BNK	5D CV	1419 IBNK
87 IBCMD	91 PREVTRK	FF STKO	C010 KBDSTRB
1810 PHP	C000 KYBD	C008 KEYBD	C058 PDLEN
?C047 ADRS	C050 GRMD	C051 TXTMD	COF2 ACIACM
C066 ADTO	C0D0 DISKOFF	COF1 ACIAST	C300 SLT3
COF3 ACIACN	C100 SLT1	C200 SLT2	F4C5 RAMTBL
C400 SLT4	CFFF EXPROM	F479 BLOCKIO	F526 NXBIT
F4CD CHPG	F513 DISK1	F524 NXBYT	F548 NOMEM
F52A NOGOOD	F532 CNTWR	F53A PULBT	F593 ERRLP1
F54F NMEM1	F562 NMEM2	F575 ERRLP	F58E ZP1
F5A1 RAMWT1	F5A5 RAMWT2	F5AD RAMWT4	F5F2 ROMTST1
F5D4 ZP2	F5E2 ZP3	F5E7 ROMTST	F653 ATD
F5FE ROMTST2	F60D VIATST	F63E ACIA	F678 KEYPLUG
F663 ADCTST1	F669 ADCTST3	F676 ADCERR	F6C2 SEX
F689 RECON	F6A1 BDOT	F6BF GOBOOT	?F6E6 USRENTY
F6C4 SEX1	F6CE SEX2	F6D4 SEX3	F72A RAMTST6
F6F4 RAMTST0	F6FD RAMTST1	F70F RAMTST4	F772 RAM00
F738 STRWT	F748 RAM	F76D RAMO	F784 RAMSET
F776 RAM1	F77B MESSERR	F77E ERROR	F7B8 PINC1
F78D RAMSET1	F7A1 PTRINC	F7D6 PINC2	F7DB RAMERR3
F7C3 RETS	F7C9 RAMERR	F7D7 RAMERR2	F7FB RAMRD
F7EA RAMERR4	F7F1 RAMERR5	F7F7 RAMWT	FD07 CROUT1
F901 MONITOR	FBC7 SETCVH	FC25 COUT	FFD0 ZPREG
FD0F KEYIN	FD98 CLDSTR1	FD9D SETUP	FFE0 SYSE0
FFD2 SYSD2	FFD3 SYSD3	FFDF SYSD1	
FFE2 SYSE2	FFE3 SYSE3	FFEF BNKSW	

----- NEXT OBJECT FILE NAME IS MON.OBJ

F7FF:                      2                      ORG      \*F7FF

F7FF                      3 \*

	75	4,383,296	76
E7FF	4 *		
E7FF 60	5 RETI	RTS	
F800 E9 01	6	SBC #1	
F802 F0 FB	7	BEG RETI	
F804 E9 01	8	SBC #1	
F806 F0 F7	9	BEG RETI	
F808 E9 01	10	SBC #1	
F80A F0 FB	11	BEG RETI	
F80C E9 01	12	SBC #1	
F80E F0 FB	13	BEG RETI	
F810 E9 01	14	SBC #1	
F812 F0 FB	15	BEG RETI	
F814 E9 01	16	SBC #1	
F816 F0 F7	17	BEG RETI	
F818 E9 01	18	SBC #1	
F81A F0 FB	19	BEG RETI	
F81C E9 01	20	SBC #1	
F81E F0 FB	21	BEG RETI	
F820 E9 01	22	SBC #1	
F822 F0 FB	23	BEG RETI	
F824 E9 01	24	SBC #1	
F826 F0 FB	25	BEG RETI	
F828 E9 01	26	SBC #1	
F82A F0 FB	27	BEG RETI	
F82C E9 01	28	SBC #1	
F82E F0 FB	29	BEG RETI	
F830 E9 01	30	SBC #1	
F832 F0 FB	31	BEG RETI	
F834 E9 01	32	SBC #1	
F836 F0 FB	33	BEG RETI	
F838 E9 01	34	SBC #1	
F83A F0 FB	35	BEG RETI	
F83C E9 01	36	SBC #1	
F83E F0 FB	37	BEG RETI	
F840 E9 01	38	SBC #1	
F842 F0 FB	39	BEG RETI	
F844 E9 01	40	SBC #1	
F846 F0 FB	41	BEG RETI	
F848 E9 01	42	SBC #1	
F84A F0 FB	43	BEG RETI	
F84C E9 01	44	SBC #1	
F84E F0 FB	45	BEG RETI	
F850 E9 01	46	SBC #1	
F852 F0 FB	47	BEG RETI	
F854 E9 01	48	SBC #1	
F856 F0 FB	49	BEG RETI	
F858 E9 01	50	SBC #1	
F85A F0 FB	51	BEG RETI	
F85C E9 01	52	SBC #1	
F85E F0 FB	53	BEG RETI	
F860 E9 01	54	SBC #1	
F862 F0 FB	55	BEG RETI	
F864 E9 01	56	SBC #1	
F866 F0 FB	57	BEG RETI	
F868 E9 01	58	SBC #1	

		4,383,296	
		77	78
F86A: F0 93	59	BEG RET1	
F86C: E9 01	60	SBC #1	
F86E: F0 8F	61	BEG RET1	
F870: E9 01	62	SBC #1	
F872: F0 8B	63	BEG RET1	
F874: E9 01	64	SBC #1	
F876: F0 87	65	BEG RET1	
F878: E9 01	66	SBC #1	
F87A: F0 83	67	BEG RET1	
F87C: E9 01	68	SBC #1	
F87E: F0 02	69	BEG RET3	
F880: E9 01	70	SBC #1	
F882: F0 7C	71 RET3	BEG RET2	
F884: E9 01	72	SBC #1	
F886: F0 78	73	BEG RET2	
F888: E9 01	74	SBC #1	
F88A: F0 74	75	BEG RET2	
F88C: E9 01	76	SBC #1	
F88E: F0 70	77	BEG RET2	
F890: E9 01	78	SBC #1	
F892: F0 6C	79	BEG RET2	
F894: E9 01	80	SBC #1	
F896: F0 68	81	BEG RET2	
F898: E9 01	82	SBC #1	
F89A: F0 64	83	BEG RET2	
F89C: E9 01	84	SBC #1	
F89E: F0 60	85	BEG RET2	
F8A0: E9 01	86	SBC #1	
F8A2: F0 5C	87	BEG RET2	
F8A4: E9 01	88	SBC #1	
F8A6: F0 58	89	BEG RET2	
F8A8: E9 01	90	SBC #1	
F8AA: F0 54	91	BEG RET2	
F8AC: E9 01	92	SBC #1	
F8AE: F0 50	93	BEG RET2	
F8B0: E9 01	94	SBC #1	
F8B2: F0 4C	95	BEG RET2	
F8B4: E9 01	96	SBC #1	
F8B6: F0 48	97	BEG RET2	
F8B8: E9 01	98	SBC #1	
F8BA: F0 44	99	BEG RET2	
F8BC: E9 01	100	SBC #1	
F8BE: F0 40	101	BEG RET2	
F8C0: E9 01	102	SBC #1	
F8C2: F0 3C	103	BEG RET2	
F8C4: E9 01	104	SBC #1	
F8C6: F0 38	105	BEG RET2	
F8C8: E9 01	106	SBC #1	
F8CA: F0 34	107	BEG RET2	
F8CC: E9 01	108	SBC #1	
F8CE: F0 30	109	BEG RET2	
F8D0: E9 01	110	SBC #1	
F8D2: F0 2C	111	BEG RET2	
F8D4: E9 01	112	SBC #1	

4,383,296

80

79

F8DA	FO	28	113	BEG	RET2
F8DB	E9	01	114	SBC	#1
F8EA	FO	24	115	BEG	RET2
F8EC	E9	01	116	SBC	#1
F8DE	FO	20	117	BEG	RET2
F8EO	E9	01	118	SBC	#1
F8E2	FO	10	119	BEG	RET2
F8E4	E9	01	120	SBC	#1
F8E6	FO	14	121	BEG	RET2
F8E8	E9	01	122	SBC	#1
F8EA	FO	18	123	BEG	RET2
F8EC	E9	01	124	SBC	#1
F8EE	FO	22	125	BEG	RET2
F8F0	E9	01	126	SBC	#1
F8F2	FO	26	127	BEG	RET2
F8F4	E9	01	128	SBC	#1
F8F6	FO	30	129	BEG	RET2
F8F8	E9	01	130	SBC	#1
F8FA	FO	34	131	BEG	RET2
F8FC	E9	01	132	SBC	#1
F8FE	FO	38	133	BEG	RET2
F900	E9	01	134	SBC	#1
F901			135	END	MUN9A
F901			2 *		
F901			3 *		
0058			4	SCRNLOC EQU	\$58
F901			5 *		
0058			6	LMARGIN EQU	SCRNLOC
0059			7	RMARGIN EQU	SCRNLOC+1
005A			8	WINTOP EQU	SCRNLOC+2
005B			9	WINETM EQU	SCRNLOC+3
005C			10	CH EQU	SCRNLOC+4
005D			11	CV EQU	SCRNLOC+5
005E			12	BAS4L EQU	SCRNLOC+6
005F			13	BAS4H EQU	SCRNLOC+7
0060			14	BAS8L EQU	SCRNLOC+8
0061			15	BAS8H EQU	SCRNLOC+9
0062			16	TBAS4L EQU	SCRNLOC+\$A
0063			17	TBAS4H EQU	SCRNLOC+\$B
0064			18	TBAS8L EQU	SCRNLOC+\$C
0065			19	TBAS8H EQU	SCRNLOC+\$D
0066			20	FORGND EQU	SCRNLOC+\$E
0067			21	BKGND EQU	SCRNLOC+\$F
0068			22	MODES EQU	SCRNLOC+\$10
0069			23	CURSOR EQU	SCRNLOC+\$11
006A			24	STACK EQU	SCRNLOC+\$12
006B			25	PROMPT EQU	SCRNLOC+\$13
006C			26	TEMPX EQU	SCRNLOC+\$14
006D			27	TEMPY EQU	SCRNLOC+\$15
006E			28	CSWL EQU	SCRNLOC+\$16
006F			29	CSWH EQU	SCRNLOC+\$17
0070			30	KSWL EQU	SCRNLOC+\$18
0071			31	KSWH EQU	SCRNLOC+\$19
0072			32	PCL EQU	SCRNLOC+\$1A
0073			33	PCH EQU	SCRNLOC+\$1B
0074			34	A1L EQU	SCRNLOC+\$1C
0075			35	A1H EQU	A1L+1
0076			36	A2L EQU	A1L+2
0077			37	A2H EQU	A1L+3
0078			38	A3L EQU	A1L+4
0079			39	A3H EQU	A1L+5

4,383,296

81

82

```

007A: 40 A4L EQU A1L+6
007B: 41 A4H EQU A1L+7
007C: 42 STATE EQU A1L+8
007D: 43 YSAV EQU A1L+9
007E: 44 INBUF EQU A1L+$A ; AND $B
0080: 45 TEMP EQU A1L+$C
0069: 46 MASK EQU CURSOR
F901: 47 *
C000: 48 KBD EQU $C000
C010: 49 KBDSTRB EQU $C010
F901: 50 *
03F8: 51 USERADR EQU $3F8
F479: 52 BLOCKIO EQU $F479
F689: 53 RECON EQU $F689 AS OF 12/20/79
F4EE: 54 DIAGN EQU $F4EE
0050: 55 INBUFLN EQU $50 ; ONLY 80 BYTES ($3A0-3EF)
0081: 56 IB SLOT EQU $81
0082: 57 IBDRVN EQU IB SLOT+1
0083: 58 IBDFP EQU IB SLOT+4
0087: 59 IBCMD EQU IB SLOT+6
F901 60 *
F901: 61 ENTRY EQU *
F901:DA 62 TSX
F902 86 6A 63 STX STACK
F904 64 *
F904: D8 65 MON CLD ; MUST BE HEX MODE
F905: 20 3A FC 66 JSR DELL
F908 A6 6A 67 MONZ LDX STACK ; RESTORE STACK TO ORIGINAL LOCATION
F90A 9A 68 TSX
F90B: A9 DF 69 LDA #$DF ; PROMPT (APPLE) FOR SARA MONITOR
F90D 85 6B 70 STA PROMPT
F90F: 20 D5 FC 71 JSR GETLNZ ; GET A LINE OF INPUT
F912: 20 67 F9 72 SCAN JSR ZSTATE ; SET REGULAR SCAN
F915: 20 2C F9 73 NXTINP JSR GETNUM ; ATTEMPT TO READ HEX BYTE
F918 84 7D 74 STY YSAV ; STORE CURRENT INPUT POINTER
F91A A0 11 75 LDY #$11 ; 17 COMMANDS
F91C 8B 76 CMDSRCH DEY
F91D: 30 E5 77 BMI MON ; GIVE UP IF UNRECOGNIZABLE
F91F: D9 6C F9 78 CMP CMDTAB,Y ; FOUND?
F922: D0 F8 79 BNE CMDSRCH ; NO KEEP LOOKING
F924: 20 5E F9 80 JSR TOSUB ; PERFORM FUNCTION
F927 A4 7D 81 LDY YSAV ; GET NEXT POINTER
F929 4C 15 F9 82 JMP NXTINP ; DO NEXT COMMAND
F92C: 83 *
F92C: A2 00 84 GETNUM LDX #0 ; CLEAR A2
F92E 86 76 85 STX A2L
F930 86 77 86 STX A2H
F932 B1 7E 87 NXTCHR LDA (INBUF),Y
F934 C8 88 INY ; BUMP INDEX FOR NEXT TIME
F935 49 D0 89 EOR #$B0
F937 C9 0A 90 CMP #$A ; TEST FOR DIGIT
F939 90 06 91 BCC DIGIT ; SAVE IT IF 1-9
F93B 69 8B 92 ADC #$B8 ; TEST FOR HEX A-F
F93D C9 FA 93 CMP #$FA
F93F 90 2A 94 BCC DIGRET
F941: A2 03 95 DIGIT LDX #3
F943: 0A 96 ASL A
F944: 0A 97 ASL A
F945: 0A 98 ASL A
F946: 0A 99 ASL A
F947: 0A 100 NXTBIT ASL A ; SHIFT HEX DIGITS INTO A2
F948: 26 76 101 ROL A2L
F94A: 26 77 102 ROL A2H
F94C: CA 103 DEX ; SHIFTED ALL YET?
F94D: 10 F8 104 BPL NXTBIT
F94F A5 7C 105 NXTBAS LDA STATE
F951: D0 06 106 BNE NXTBS2 ; IF ZERO THEN COPY TO A1,3

```

4,383,296

83

84

F953: B5 77	107	LDA	A2H, X	
F955: 95 75	108	STA	A1H, X	
F957: 95 79	109	STA	A3H, X	
F959: EB	110	NXTBS2	INX	
F95A: F0 F3	111	BEQ	NXTBAS	
F95C: D0 D4	112	BNE	NXTCHR	
F95E:	113	*		
F95E: A9 FA	114	TOSUB	LDA #ASCII	PUSH ADDRESS OR FUNCTION
F960: 48	115	PHA		AND RETURN TO IT.
F961: B9 7C F9	116	LDA	CMDVEC, Y	
F964: 48	117	PHA		
F965: A5 7C	118	LDA	STATE	PASS MODE VIA ACC.
F967: A0 00	119	ZSTATE	LDY #0	
F969: 84 7C	120	STY	STATE	RESET STATE OF SCAN
F96B: 60	121	DIGRET	RTS	
F96C:	122	*		
F96C:	123	CMDTAB	EQU *	
F96C: 00	124	DFB	\$0	G =GO (CALL) SUBROUTINE
F96D: 03	125	DFB	\$3	J =JUMP (CONT) PROGRAM
F96E: 06	126	DFB	\$6	M =MOVE MEMORY
F96F: EB	127	DFB	\$EB	R =READ DISK BLOCK
F970: EE	128	DFB	\$EE	U =USER FUNCTION
F971: EF	129	DFB	\$EF	V =VERIFY MEMORY BLOCKS
F972: F0	130	DFB	\$F0	W =WRITE DISK BLOCK
F973: F1	131	DFB	\$F1	X =REPEAT LINE OF COMMANDS
F974: 99	132	DFB	\$99	SP =SPACE (DYTE SEPARATOR)
F975: 9B	133	DFB	\$9B	" =ASCII (HI BIT ON)
F976: A0	134	DFB	\$A0	' =ASCII (HI BIT OFF)
F977: 93	135	DFB	\$93	:
F978: A7	136	DFB	\$A7	=SET STORE MODE
F979: A8	137	DFB	\$A8	=RANGE SEPARATOR
F97A: 95	138	DFB	\$95	/ =COMMAND SEPARATOR
F97B: C6	139	DFB	\$C6	< =DEST/SOURCE SEPARATOR
F97C:	140	*		CR =CARRAGE RETURN
F97C:	141	CMDVEC	EQU *	
F97C: 7C	142	DFB	GO-1	
F97D: 7A	143	DFB	JUMP-1	
F97E: 2B	144	DFB	MOVE-1	
F97F: DF	145	DFB	READ-1	
F980: 77	146	DFB	USER-1	
F981: 3A	147	DFB	VRFY-1	
F982: C2	148	DFB	WRITE-1	
F983: 18	149	DFB	REPEAT-1	
F984: A3	150	DFB	SPCE-1	
F985: C6	151	DFB	ASCII-1	
F986: C6	152	DFB	ASCII0-1	
F987: 27	153	DFB	SETMODE-1	
F988: B7	154	DFB	SETMODE-1	
F989: 99	155	DFB	SEP-1	
F98A: 90	156	DFB	DEST-1	
F98B: 25	157	DFB	CRMON-1	
F98C:	158	*		
F98C:	159	*		
F98C: E6 7A	160	NXTA4	INC A4L	BUMP 16 BIT POINTERS
F98E: D0 02	161	BNE	NXTA1	
F990: E6 7B	162	INC	A4H	
F992: E6 74	163	NXTA1	INC A1L	BUMP A1
F994: D0 05	164	BNE	TSTA1	
F996: E6 75	165	INC	A1H	
F998: 3E	166	SEC		IN CASE OF ROLL OVER.
F999: F0 10	167	BEQ	RETA1	
F99B: A5 74	168	TSTA1	LDA A1L	TEST A1/A2
F99D: 3E	169	SEC		
F99E: E5 76	170	SBC	A2L	
F9A0: 85 80	171	STA	TEMP	

4,383,296

85		86	
F9A2: A5 75	172	LDA A1H	
F9A4: E5 77	173	SBC A2H	
F9A6: 05 30	174	ORA TEMP	
F9A8: D0 01	175	BNE RETA1	IF A1 LESS THAN OR EQUAL TO A2
<b>F9AA: 18</b>	<b>176</b>	<b>CLC</b>	<b>THEN CARRY CLEAR ON RETURN</b>
<b>F9AB: 40</b>	<b>177 RETA1</b>	<b>RTS</b>	
F9AC:	178 *		
F9AC:	179 *		
F9AC: 48	180 PRBYTE	PHA	SAVE LOW NIBBLE
F9AD: 4A	181	LSR A	
F9AE: 4A	182	LSR A	SHIFT HI NIBBLE TO PRINT.
F9AF: 4A	183	LSR A	
F9B0: 4A	184	LSR A	
F9B1: 20 B7 F9	185	JSR PRHEXZ	
F9B4: 6B	186	PLA	
F9B5: 29 0F	187 PRHEX	AND ##0F	STRIP HI NIBBLE
F9B7: 09 B0	188 PRHEXZ	ORA ##B0	MAKE IT NUMERIC
F9B9: C9 BA	189	CMP ##BA	IS IT >'9'
F9BB: 90 02	190	BCC PRHEX2	
F9BD: 69 06	191	ADC ##6	MAKE IT 'A'-'F'
F9BF: 4C 25 FC	192 PRHEX2	JMP COUT	
F9C2:	193 *		
F9C2: 20 AC F9	194 PRBYCOL	JSR PRBYTE	
F9C5:	195 *		
F9C5: A9 BA	196 PRCOLON	LDA ##BA	PRINT A COLON
F9C7: D0 F6	197	BNE PRHEX2	BRANCH ALWAYS
F9C9:	198 *		
F9C9: A9 07	199 TSTBOWID	LDA #7	ANTICIPATE
F9CB: 24 68	200	BIT MODES	TEST FOR 80
F9CD: 50 02	201	BVC SVMASK	
F9CF: A9 0F	202	LDA ##F	
F9D1: 85 69	203 SVMASK	STA MASK	
F9D3: 60	204	RTS	
F9D4:	205 *		
F9D4: 8A	206 A1PC	TXA	TEST FOR NEW PC
F9D5: F0 07	207	BEG OLDP	
F9D7: B5 74	208 A1PC1	LDA A1L, X	
F9D9: 95 72	209	STA PCL, X	
F9DB: CA	210	DEX	
F9DC: 10 F9	211	BPL A1PC1	
F9DE: 60	212 OLDP	RTS	
F9DF:	213 *		
F9DF: 85 69	214 ASCII1	STA MASK	SAVE HI BIT STATUS
F9E1: A4 7D	215 ASCII2	LDY YSAV	MOVE ASCII TO MEMORY
F9E3: B1 7E	216	LDA (INBUF), Y	
F9E5: E6 7D	217	INC YSAV	BUMP FOR NEXT THING.
F9E7: A0 00	218	LDY #0	
F9E9: C9 A2	219	CMP ##A2	ASCII " ?
F9EB: D0 05	220	BNE ASCII3	NOPE, CONTINUE.
F9ED: A5 69	221	LDA MASK	
F9EF: 10 20	222	BPL BITON	HE'S CHANGED MODES.
F9F1: 60	223	RTS	NO, HE'S DONE.
F9F2: C9 A7	224 ASCII3	CMP ##A7	ASCII ' ?
F9F4: D0 05	225	BNE CRCHK	NO, TEST FOR EOL.
F9F6: A5 69	226	LDA MASK	
F9F8: 30 1B	227	BMI BITOFF	CHANGE MODES.
F9FA: 60	228	RTS	
F9FB: C9 8D	229 CRCHK	CMP ##8D	END OF LINE?
F9FD: F0 07	230	BEG ASCDONE	YES, FINISHED
F9FF: 25 69	231	AND MASK	
FA01: 20 AF FA	232	JSR STOR1	GO STORE IT!
FA04: D0 DB	233	BNE ASCII2	DO NEXT.



4,383,296

87

88

FA06: 60	234	ABCDONE	RTS		
FA07:	235	*			
FA07: 38	236	ASCII	SEC		INDICATE HI ON.
FA08: 90	237	DFB	\$90		(BCC - NEVER TAKEN)
FA09: 18	238	ASCII0	CLC		INDICATE HI OFF
FA0A: AA	239	CKMDE	TAX		SAVE STATE
FA0B: 86 7C	240	STX	STATE		RETAIN STATE
FA0D: 49 BA	241	EOR	#\$BA		ARE WE IN STORE MODE?
FA0F: D0 7D	242	DNE	ERROR		
FA11: A7 FF	243	DITON	LDA	#\$FF	SET HI BIT UNMASKED
FA13: D0 CA	244	BCS	ASCII1		
FA15: A9 7F	245	DITOFF	LDA	#\$7F	MASK HI BIT
FA17: 10 C6	246	BPL	ASCII1		ALWAYS
FA19: 20 00 C0	247	REPEAT	DIT	KBD	REPEAT UNTIL KEYPRESS
FA1C: 10 03	248	BPL	REPEAT1		
FA1E: 40 0F FD	249	JMP	KEYIN		
FA21: 68	250	REPEAT1	PLA		CLEAN UP STACK
FA22: 68	251	PLA			
FA23: 40 12 F9	252	JMP	SCAN		
FA26:	253	*			
FA26:	254	*			
FA26: 20 A0 FA	255	CRMON	JSR	BL1	
FA29: 40 08 F9	256	JMP	MONZ		
FA2C:	257	*			
FA2C: 20 9B F9	258	MOVE	JSR	TSTA1	DON'T MOVE ANYTHING IF ILLEGAL INPUT
FA2F: B0 5D	259	BCS	ERROR		
FA31: B1 74	260	MOVNXT	LDA	(A1L),Y	MOVE A BYTE
FA33: B1 7A	261	STA	(A4L),Y		
FA35: 20 9C F9	262	JSR	NXTA4		DUMP BOTH A1 AND A4
FA38: 90 F7	263	BCC	MOVNXT		
FA3A: 60	264	RTS			ALL DONE WITH MOVE
FA3B:	265	*			
FA3B:	266	*			
FA3B: 20 9B F9	267	VREFY	JSR	TSTA1	TEST VALID RANGE
FA3E: B0 4E	268	BCS	ERROR		
FA40: B1 74	269	VREFY1	LDA	(A1L),Y	COMPARE BYTE FOR BYTE
FA42: D1 7A	270	CMP	(A4L),Y		MATCH?
FA44: F0 06	271	BEG	VREFY2		YES, DO NEXT.
FA46: 20 52 FA	272	JSR	MISMATCH		PRINT BOTH BYTES
FA49: 20 EF FC	273	JSR	CROUT		GOTO NEWLINE
FA4C: 20 8C F9	274	VREFY2	JSR	NXTA4	DUMP BOTH A1 AND A4
FA4F: 90 EF	275	BCC	VREFY1		
FA51: 60	276	RTS			VERIFY DONE.
FA52:	277	*			
FA52: A5 7B	278	MISMATCH	LDA	A4H	PRINT ADDRESS OF A4
FA54: 20 AC F9	279	JSR	PRBYTE		
FA57: A5 7A	280	LDA	A4L		
FA59: 20 C2 F9	281	JSR	PRBYCOL		OUTPUT A COLON FOR SEPARATOR
FA5C: D1 7A	282	LDA	(A4L),Y		AND THE DATA...
FA5E: 20 70 FA	283	JSR	PRBYTSP		PRINT THE BYTE AND A SPACE
FA61: 20 73 FA	284	PRINTA1	JSR	PRSPC	LEAD WITH A SPACE
FA64: A5 75	285	LDA	A1H		OUTPUT ADDRESS A1
FA66: 20 AC F9	286	JSR	PRBYTE		
FA69: A5 74	287	LDA	A1L		
FA6B: 20 C2 F9	288	JSR	PRBYCOL		SEPARATE WITH A COLEN
FA6E: B1 74	289	PRA1BYTE	LDA	(A1L),Y	PRINT BYTE POINTED TO BY A1
FA70: 20 AC F9	290	PRBYTSP	JSR	PRBYTE	
FA73: A5 70	291	PRSPC	LDA	#\$A0	PRINT A SPACE
FA75: 40 25 FC	292	JMP	COUT		END VIA OUTPUT ROUTINE.
FA78:	293	*			
FA78: 40 F8 05	294	USER	JMP	USERADR	
FA7B:	295	*			
FA7B: 68	296	JUMP	PLA		
FA7C: 68	297	PLA			LEAVE STACK WITH NOTHING ON IT
FA7D: 20 D4 F9	298	CD	JSR	A1PC	STUFF PROGRAM COUNTER
FA80: 60 72 00	299	JMP	(PCL)		JUMP TO USER PROG
FA83:	300	*			
FA83:	301	RWERROR	EQU	*	PRINT ERROR NUMBER

4,383,296

89

90

FAB2 20 AC F9	302	JSR	PRDYTE	PRINT THE OFFENDER
FAB6 A9 A1	303	LDA	##A1	FOLLOWED BY A "!"
FAB8 20 25 FC	304	JSR	COOT	
FAB8 20 07 FD	305 ERROR2	JSR	NOSTOP	OUTPUT A CARRAGE RETURN AND STOPLIST
FABE 4C 04 F9	306 ERROR	JMP	MON	
FAB1	307 *			
FAB1 A5 26	308 DEST	LDA	A2L	COPY A2 TO A4 FOR DESTINATION OF
FA93 B5 7A	309	STA	A4L	
FA95 A5 77	310	LDA	A2H	
FA97 B5 7B	311	STA	A4H	
FA99 80	312	RTS		
FA9A	313 *			
FA9A 20 44 FA	314 SEP	JSR	SPACE	SEPARATOR TEST STORE MODE OR DUMP
FA9D 98	315	TYA		ZERO MODE
FA9E FD 1D	316	BEQ	SETMDZ	BRANCH ALWAYS
FAA0	317 *			
FAA2 15 1D	318 D11	DEC	YSAV	TEST FOR NO LINE
FAA2 FD 15	319	BEQ	DUMPH	IF NO LINE, GIVEN A ROW OF BYTES
FAA4 CA	320 S11	DEY		TEST IF AFTER ANOTHER SPACE
FAA5 D0 16	321	BNE	SETMDZ	
FAA7 C9 DA	322	CMF	##CA	STORE MODE?
FAA9 D0 4B	323	BNE	TRUMF	
FAAB B5 7C	324 STOP	STA	STATE	KEEP IT IN STORE STATE
FAAD A5 7A	325	LDA	A2L	GET BYTE TO BE STORED
FAAF 91 7B	326 STORE	SEA	CABU	PUT IT IN MEMORY
FAB1 E6 7B	327	INC	A3L	BUMP POINTER
FAB1 D0 02	328	BNE	DUMMY	
FAB5 E5 79	329	INC	A0H	
FAB7 80	330 DUMMY	RTS		ALSO USED FOR CLEAR MODE
FA	331			
FA 15 1D	332 S11 MODE	LDY	YSAV	LINE INPUT CHARACTER
FADA 1F	333	DEY		
FAB8 B1 7E	334	LDA	(INBUFF)	TO SET MODE
FABE 20 7C	335 SETMDZ	STA	STATE	
FABF 80	336	RTS		
FA	337 *			
FAC1 A1 01	338 READ	LDA	#1	SET DISK COMMAND TO READ
FAC2 20	339	DFB	##2C	DUMMY BIT TO SKIP 2 BYTES
FAC3 A9 02	340 WRTE	LDA	##2	SET DISK COMMAND TO WRITE
FAC5 B5 87	341 SAVCMD	STA	IBCMD	
FAC7 A5 74	342 RWLOOP	LDA	A1L	COMMAND FORMAT IS
FAC9 B5 85	343	STA	IBBUFP	BLOCKNUMBER (ADDRESS) ENDADDRESS
FAD1 A5 7B	344	LDA	A1H	
FAD1 D0 80	345	STA	IBBUFP+1	
FAD3 A5 7D	346	LDX	A4H	SEND BLOCK NUMBER VIA X & A
FAD1 A5 7A	347	LDA	A4L	
FAD3 7B	348	SEI		NO INTERRUPTS WHILE IN MONITOR
FADA 20 74 FA	349	JSR	BLOCKIO	DO DISK FEVER
FAD7 B0 AA	350	BCS	RWERROR	GIVE UP IF ERROR ENCOUNTERED
FAD9 E6 7A	351	INC	A4L	BUMP BLOCK NUMBER
FADB D0 02	352	BNE	NOVER	
FAD9 E6 7B	353	INC	A4H	
FAD9 E6 75	354 NOVER	INC	A1H	BUMP RAM ADDRESS BY 412 BYTES
FAD1 E6 75	355	INC	A1H	
FAE3 20 9B F9	356	JSR	TSTA1	TEST FOR FINISHED
FAE6 90 DF	357	BCC	RWLOOP	NOT DONE. DO NEXT BLOCK
FAE8 80	358	RTS		
FAE7	359 *			
FAE9	360	CHN	MONPB	
FAE9	1 DUMPB	EGU	*	OUTPUT 1 ROW OF BYTES
FAE9 A5 75	2	LDA	A1H	
FAEB B5 77	3	STA	A2H	
FAED 20 09 F9	4	JSR	TSTROWID	GET WIDTH MASK INTO ACC
FAE0 05 74	5	ORA	A1L	
FAE2 B5 76	6	STA	A2L	
FAE4 D0 06	7	BNE	DUMPB	BRANCH ALWAYS
FAE6	8 *			
FAE6 4A	9 TSTDUMP	LSR	A	DUMP?
FAE7 B0 95	10 ERROR1	BCS	ERROR	

4,383,296

91

92

FAFE 20 09 FF	11 DUMP	JSR TSTBOWID	SET FOR EITHER 80 OR 40 COLUMNS
FAFD A5 74	12 DUMP	LDA A1L	USE A4 FOR ASCII DUMP
FAFE 85 7A	13	STA A4L	
FB00 A5 75	14	LDA A1H	
FB02 95 7B	15	STA A4H	
FB04 20 9B FF	16	JSR TSTA1	TEST FOR VALID RANGE
FB07 80 EE	17	BCS ERROR1	
FB07 20 81 FF	18 DUMP	JSR PRINTA1	PRINT ADDRESS AND FIRST BYTE
FB0C 20 92 FF	19 DUMP	JSR NXTA1	
FB0F 80 10	20	BCS DUMPASC	END WITH ASCII
FB11 A5 74	21	LDA A1L	TEST END OF LINE
FB11 25 59	22	AND MASK	FOR 40/80 COLUMN
FB15 20 05	23	BNE DUMP3	
FB17 20 21 FF	24	JSR DUMPASC	
FB1A 20 ED	25	BNE DUMP1	BRANCH ALWAYS
FB1C 20 8E FA	26 DUMP	JSR PRABYTE	GO PRINT NEXT BYTE AND A SPACE
FB1F 20 EB	27	BNE DUMP2	ALWAYS (ACC JUST PULLED AS \$A0)
FB21	28 *		
FB21 A5 7A	29 DUMFASC	LDA A4L	RESET TO BEGINING OF LINE
FB23 25 74	30	STA A1L	
FB25 A5 7B	31	LDA A4H	
FB27 85 75	32	STA A1H	
FB29 20 73 FF	33	JSR PRSPC	PRINT AN EXTRA SPACE
FB2C 20 00	34 ASCII	LDY #0	TO INDEX MEMORY INDIRECT
FB2E 20 71	35	LDA \$A1L	
FB30 20 80	36	ORA #\$80	SET NORMAL VIDEO
FB32 20 A0	37	CMP #\$A0	TEST FOR CONTROL CHARACTERS
FB34 20 02	38	BCS ASC2	OK TO PRINT NON CONTROLS
FB35 A5 AC	39	LDA #\$AE	OTHERWISE PRINT A SPACE
FB38 20 05 FF	40 ASCII	JSR CRUIT	PUT IT OUT
FB3C 20 80 FF	41	JSR DUMP4	BRANCH BOTH A1 AND A4
FB3E 20 06	42	BCS ASC3	FINISHED
FB41 A5 74	43	LDA A1L	TEST END OF LINE
FB43 20 09	44	AND MASK	
FB45 20 05	45	BNE ASC1	NOT DONE, PRINT NEXT
FB48 20 00 FF	46 ASCII	JMP CROUT	
FB49	47 *		
FB49	48 *		
FB49	49 *		
FB49 38	50 COL80	SEC	INDICATE 80 COLUMNS DESIRED
FB4A AD 53 C0	51	LDA \$C053	GOTO 80 COLUMN MODE
FB4B 20 04	52	BCS SET80	BRANCH ALWAYS
FB4F	53 *		
FB4F 18	54 COL40	CLC	INDICATE 40 COLUMNS DESIRED
FB50 AD 52 C0	55	LDA \$C052	GOTO 40 COLUMN MODE
FB53 A5 68	56 SET80	LDA MODES	
FB55 20 40	57	ORA #\$40	ASSUME 80
FB57 20 02	58	BCS SET80A	AND BRANCH IF IT IS
FB59 29 BF	59	AND #\$BF	BUT FIX FOR 40 IF NOT
FB5B 85 68	60 SET80A	STA MODES	
FB5D 09 7F	61	ORA #\$7F	ISOLATE BIT 7
FB5F 29 A0	62	AND #\$A0	(BIT 7 SETS NORMAL/INVERSE)
FB61 85 66	63	STA FORGND	
FB63 20 02	64	BCS SET80B	AGAIN ASSUMES 80 COLUMNS
FB65 A9 F0	65	LDA #\$F0	IF NOT, SET FOR/BACKGROUND COLOR
FB67 85 67	66 SET80B	STA BKGND	
FB69	67 *		
FB69 A5 58	68 CLSCRN	LDA LMARGIN	SET CURSOR TO TOP LEFT OF WINDOW
FB6B 20 5C	69	STA CH	
FB6D A5 5A	70	LDA WINTOP	
FB6F 85 5D	71	STA CV	NOW DROP INTO CLEAR END OF PAGE
FB71	72 *		
FB71 A5 5C	73 CLEOP	LDA CH	SAVE CURRENT CURSOR POSITION
FB73 48	74	PHA	
FB74 A5 5D	75	LDA CV	
FB75 48	76	PHA	
FB77 20 D1 FB	77	JSR SETCV	
FB7A 20 8E FB	78 CLEOP1	JSR CLEOL	CLEAR TO END OF FIRST LINE
FB7D A5 58	79	LDA LMARGIN	
FB7F 85 5C	80	STA CH	
FB81 20 09 FB	81	JSR CURDOWN	GOTO NEXT LINE

4,383,296

93

94

FBB4 90 F4	82	BCC	CLEOP1	
FBB6 68	83	PLA		
FBB7 A8	84	TAY		
FBB8 68	85	PLA		RESTORE CURSOR POSITION
FBB9 85 5C	86	STA	CH	
FBBB 98	87	TYA		GET OLD CV IN ACC AGAIN
FBBC B0 23	88	BCS	SETCV	BRANCH ALWAYS
FBBE *	89	*		
FBBE A5 5C	90	CLEOL	LDA	CH
FBD0 4C B9 FC	91	JMP	CLEOL1	CLEAR TO END OF LINE FIRST
FBD3 *	92	*		
FBD3 C9 80	93	CONTROL	CMP	#\$80
FBD5 90 65	94	BCC	DISPLAYX	IF INVERSE
FBD7 C9 8D	95	TSTCR	CMP	#\$8D
FBD9 D0 3A	96	BNE	TSTBACK	IF CARRAGE RETURN THEN NEW LINE
FBD9 20 8E FB	97	CARRAGE	JSR	CLEOL
FBD9 20 C3 FB	98	JSR	SETCHZ	FIRST CLEAR TO THE END OF THIS LINE
FBA1 4C D2 FC	99	JMP	NXTLIN	RESET CURSOR AND GOTO NEXT LINE (CARRY IS SET)
FBA4 *	100	*		
FBA4 *	101	*		
FBA4 A5 5D	102	CURUP	LDA	CV
FBA6 C6 5D	103	DEC	CV	TEST FOR TOP OF SCREEM
FBA8 C5 5A	104	CMP	WINTOP	ANTICIPATE 'NOT' TOP
FBA8 D0 02	105	BNE	CURUP1	IT'S NOT TOP, CONTINUE
FBA8 A5 5B	106	LDA	WINBTM	WRAP AROUND TO BOTTOM
FBAE 38	107	CURUP1	SEC	DECREMENT BY ONE
FBAF E9 01	108	STY	#\$	
FBB1 85 5D	109	SETCV	STA	CH
FBB3 *	110	BASCALC	EQV	SAVE NEW VERTICAL LINE
FBB3 *	111	CURDN1	EQV	
FBB3 A5 5D	112	LDA	CV	GET VALUES FOR FIRST PAGE (\$400)
FBB5 10 4E	113	BPL	BASCALC	ALWAYS
FBB7 *	114	*		
FBB7 24 68	115	CURLEFT	BIT	MODES
FBB9 70 02	116	BPL	CURLEFT	TEST FOR 80 OR 40
FBBB E6 5C	117	BPL	CH	
FBBD E6 5C	118	RIGHT	BIT	CH
FBBF A5 5C	119	SET	CH	DUMP CURSOR HORIZONTAL
FBC1 C5 5A	120	JSR	RMARGIN	TEST FOR NEW LINE
FBC3 A5 5B	121	SETCHZ	LDA	LMARGIN
FBC5 90 5D	122	BPL	CURLEFT	JUST IN CASE WE HAVE
FBC7 85 5C	123	SETCVH	STA	CH
FBC9 *	124	*DEP INTO		CURSOR AT START OF NEXT LINE
FBC9 *	125	*		PREPARE FOR WRAP AROUND
FBC9 E6 5D	126	CURDN1	IN	MOVE CURSOR DOWN ONE LINE
FBCB A5 5D	127	LDA	CV	ANTICIPATE NOT BOTTOM
FBCD C5 5B	128	CMP	WINBTM	TEST FOR BOTTOM
FBCF 90 E2	129	BCC	CURDN1	
FBD1 A5 5A	130	LDA	WINTOP	
FBD3 20 0C	131	BCS	SETCV	BRANCH ALWAYS
FBD5 *	132	*		
FBD5 C9 68	133	BACKSP	CMP	#\$88
FBD7 D0 5D	134	BNE	TSTBEL	BACKSPACE?
FBD9 24 58	135	CURLEFT	BIT	MODES
FBD9 70 02	136	BVS	LEFT80	TEST FOR FORTY OR EIGHTY MODE
FBD9 C6 5C	137	DEC	CH	
FBD9 C6 5C	138	LEFT80	DEC	CH
FBD9 30 06	139	BMI	LEFTUP	
FBD9 A5 5	140	LDA	CH	TEST FOR WRAP AROUND
FBD9 C5 5B	141	CMP	LMARGIN	
FBD9 10 3B	142	BPL	CTRLRET	
FBD9 20 A4 FB	143	LEFTUP	JSR	CURUP
FBD9 A5 59	144	LDA	RMARGIN	
FBD9 25 5C	145	STA	CH	SAVE NEW CURSOR POSITION
FBD9 25 5C	146	BNE	CURLEFT	BRANCH ALWAYS
FBD9 *	147	*		
FBD9 C9 A0	148	COUT2	CMP	#\$A0
FBD9 90 9D	149	BCC	CONTROL	IS IT CONTROL CHARACTER
FBD9 24 68	150	BIT	MODES	TEST FOR INVERSE
FBD9 20 02	151	BMI	DISPLAYX	AND PUT IT OUT

95	4,383,296	96
FCFA 20 1F	152 AND #17F	STRIP HI BIT
FBFC 20 1D FC	153 DISPLAYX USR DISPLAY	
ADFF	154 *	
FCFF 20 B7 FB	155 INCHORX USR CURIGHT	MOVE CURSOR RIGHT
FCFF 20 B7	156 NYFLIN BNO SCROLL	THIS BOTTOM. RESET CHRD AND SCROLL
FCFF	157 RTS	RESET CH UNL.
FCFF	158	
FCFF	159 BASCALC1 PHF	CALC BASE ADR IN BAS4LH
FCFF	160 PHA	
FCFF	161 LSR A	FOR GIVEN LINE NO.
FCFF	162 AND #803	LOC=LINE NO. C=817
FCFF	163 ORA #804	ARG=000ABCDE, GENERATE
FC0C 85 5F	164 STA BAS4H	BAS4H=000001CD
FC0E 49 0C	165 EOR #8C	
FC11 85 81	166 STA BAS8H	
FC12 68	167 PLA	AND
FC13 29 18	168 AND #18	BAS4L=EABAB000
FC15 90 02	169 BCC BSCLC2	
FC17 49 7F	170 ADC #7F	
FC19 85 5E	171 BSCLC2 STA BAS4L	
FC1B 0A	172 ASL A	
FC1C 0A	173 ASL A	
FC1D 02 0E	174 ORA BAS4L	
FC1F 85 5E	175 STA BAS4L	
FC21 85 60	176 STA BAS8L	SAME FOR PAGE 2
FC23 29	177 PLP	
FC24 10	178 CTRLRET RTS	
FC25	179 *	
FC25 40	180 CONT PHA	SAVE CHARACTER
FC26 94 6D	181 STY TEMPY	
FC28 66 6C	182 STX TEMPX	
FC2A 20 33 FC	183 USR COUNT1	
FC2F A4 6D	184 LDY TEMPY	
FC31 A4 6C	185 LDX TEMPX	
FC32 6C	186 PLA	
FC32 6C	187 RTS	
FC33	188 *	
FC33 6C 6E 00	189 COUNT1 JMP (CSWL)	NORMALLY COUNT1
FC34	190 *	
FC34 C9 87	191 T5FRELL CMP #87	BELL?
FC38 90 12	192 BNE LNFD	NO TEST FOR FORM FEED
FC3A	193 *	
FC3A A2 10	194 BELL LDX #10	
FC3C 8A	195 TXA	
FC3D A8	196 BELL1 TAY	
FC3E 02 18 00	197 BELL2 BIT #FFD8	
FC41 90 1B	198 BEQ BELL2	
FC43 90 08 FF	199 BELL3 BIT #FFD8	
FC46 00 FB	200 BNE BELL3	
FC48 68	201 DEY	
FC4F 10 F3	202 BNE BL12	
FC4F 20 20 00	203 BIT #0000	
FC4F 6C	204 PLP	
FC4F D0 EC	205 BNE BELL1	
FC51 50	206 RTS	
FC52	207 *	
FC52 02 1A	208 CTRLRET LNF #5A	LINE FEED?
FC52	209	
FC52 02 1A	210 USR CURDOWN	MOVE CURSOR DOWN A LINE
FC52 90 1A	211 BCC CTRLRET	BRANCH IF NO SCROLL NECESSARY
FC5B	212 *	
FC5B A2 5A	213 SCROLL LDA WINTOP	START WITH TOP LINE
FC5F A2	214 PHA	SAVE IT FOR NOW
FC61 02 1A	215 USR SETCV	GET BASCALC FOR THIS LINE
FC61	216	
FC61 02 1A	217 SCRL2 LDX #3	MOVE CURRENT BASCALC AS DESTINATION
FC61 02 1A	218 LDA BAS4LX	
FC61 02 1A	219 STA TBAS4LX	(TEMPORARY BASE ADDR.)
FC61 02 1A	220 DEX	
FC61 02 1A	221 BPL SCRL2	
FC61 02 1A	222 PLA	GET DESTINATION LINE

4,383,296

97

98

FC6B: 18	222	CLC		
FC6C: 69 01	223	ADC #1		CALCULATE SOURCE LINE
FC6E C9 58	224	CMF WINDTH		IS IT THE LAST LINE?
FC70 D0 15	225	BOS LASTLN		YES, CLEAR IT
FC72 48	226	PHA		SAVE AS NEXT DESTINATION LINE
FC73: 20 B1 FB	227	JSR SETCV		GET BASE ADDR FOR SOURCE LINE
FC76 A5 59	228	LDA RMARGIN		MOVE SOURCE TO DESTINATION
FC78 4A	229	LSR A		DIVIDE BY 2
FC79 A8	230	TAY		
FC7A 88	231	SCRL3 DEY		DONE YET?
FC7B 30 E4	232	BMI SCRL1		YES, DO NEXT LINE
FC7D B1 5E	233	LDA (BAS4L),Y		
FC7F 91 62	234	STA (TBAS4L),Y		
FC81 B1 60	235	LDA (BAS8L),Y		MOVE BOTH PAGES
FC83 91 64	236	STA (TBAS8L),Y		
FC85 90 F3	237	BCC SCRL3		BRANCH ALWAYS
FC87 A5 58	238	LDA LMARGIN		BLANK FILL THE LAST LINE
FC89 4A	239	LSR A		DIVIDE BY 2
FC8A A8	240	TAY		
FC8B B0 04	241	BOS CLEOL2		
FC8D A5 66	242	LDA FORGND		(NORMALLY A SPACE)
FC8F 91 5E	243	STA (BAS4L),Y		
FC91 A5 67	244	LDA BKGND		(IF 80 COLUMNS, ALSO A SPACE)
FC93 91 60	245	STA (BAS8L),Y		
FC95 C8	246	INY		
FC96 98	247	TYA		TEST FOR END OF LINE
FC97 0A	248	ASL A		MULT BY 2 AGAIN
FC9B C5 59	249	CMF RMARGIN		
FC9A 90 ED	250	BCC CLEOL1		CONTINUE IF MORE TO DO
FC9C 60	251	RTS		ALL DONE
FC9D	252 *			
FC9D 24 68	253	DISPLAY BIT	MODES	TEST FOR 40 OR 80
FC9F 70 0C	254	BOS DSPL80		STORE THE SINGLE CHARACTER AND RETURN
FCA1 46 5C	255	LSR CH		INSURE PROPER 40 COLUMN DEL. LAY
FCA3 06 5C	256	ASL CH		BY DROPPING BIT 0
FCA5 20 AD FC	257	JSR DSPL80		DISPLAY IN \$400 PAGE
FCA8 A5 67	258	LDA BKGND		ALSO SET BACKGROUND COLOR
FCAA 91 60	259	DSPBKGNB STA	(BAS8L),Y	
FCAE 60	260	RTS		
FCAD	261 *			
FCAD 48	262	DSPL80	PHA	PRESERVE CHARACTER
FCAE A5 5C	263	LDA CH		DETERMINE WHICH PAGE
FCB0 4A	264	LSR A		
FCB1 A8	265	TAY		
FCB2 68	266	PLA		
FCB3 B0 F5	267	BOS DSPBKGNB		BRANCH IF \$800 PAGE
FCB5 91 5E	268	STA (BAS4L),Y		
FCB7 60	269	RTS		
FCB8	270 *			
FCB8 B1 7E	271	NOTCH LDA	(INBUF),Y	ECHO CHARACTER
FCBA 20 25 FC	272	JSR COUT		
FCBD C9 88	273	CMF #88		BACKSPACE?
FCBF F0 1D	274	BEG BKSPCE		
FCC1 C9 98	275	CMF #98		CANCEL?
FCC3 F0 08	276	BEG CANCEL		
FCC5 E6 80	277	INC TEMP		
FCC7 A5 80	278	LDA TEMP		
FCC9 C9 90	279	CMF #INBUFLN		
FCCB D0 17	280	BNE NXTCHAR		NO WRAP AROUND ALLOWED
FCCD A9 DC	281	CANCEL LDA	#8DC	OUTPUT BACKSLASH
FCCF 20 25 FC	282	JSR COUT		
FCD2 20 EF FC	283	JSR CROUT		
FCD5	284	GETLNZ	EGU *	
FCD5 A5 68	285	GETLN LDA	PROMPT	
FCD7 20 25 FC	286	JSR COUT		
FCD A0 01	287	LDY #1		
FCD C8 80	288	STY TEMP		START AT BEGINNING OF INBUF
FCD E A4 80	289	BKSPCE LDY	TEMP	
FCE0 F0 F3	290	BEG GETLN		
FCE2 C6 80	291	DEC TEMP		BACK UP INPUT BUFFER
FCE4 20 60 FD	292	NXTCHAR JSR	RDCHAR	GET INPUT
FCE7 A4 80	293	LDY TEMP		

4,383,296

99

100

```

FCE9: 91 7E      294      STA (INBUF),Y
FCEB: C9 8D      295      CMP  #$8D
FCED: D0 C9      296      BNE  NOTCR
FCEF:           297      CRDUT EQU  *
FCE9: 2C 00 C0    298      BIT  KBD          ;TEST FOR START/STOP
FCF2: 10 13      299      BPL  NOSTOP
FCF4: 20 2E FD    300      JSR  KEYIN3        ;READ KBD
FCF7: C9 A0      301      CMP  #$A0        ;IS IT A SPACE?
FCF9: F0 07      302      BEQ  STOPLST    ;YES, PAUSE TIL NEXT KEYPRESS
FCFB: C9 8D      303      CMP  #$8D        ;QUIT THIS OPERATION?
FCFD: D0 08      304      BNE  NOSTOP    ;NO, IGNORE THIS KEY
FCFF: 4C 8B FA    305      JMP  ENDOR2    ;YES, RESTART
FD02: AD 00 C0    306      STOPLST LDA  KBD
FD05: 10 FB      307      BPL  STOPLST
FD07: A9 8D      308      NOSTOP LDA  #$FD
FD09: 4C 25 FC    309      JMP  COUT
FD0C:           310      *
FD0C: 6C 70 00    311      RDKEY  JMP  (KSWL)
FD0F:           312      *
FD0F: A9 7F      313      KEYIN  LDA  #$7F        ;MAKE SURE FIRST IS CURSOR
FD11: 85 63      314      STA  TBAS4H
FD13: 20 88 FD    315      JSR  PICK          ;GO READ SCREEN
FD16: 48          316      KEYIN1 PHA          ;SAVE CMD AT CURSOR POSITION
FD17: 20 35 FD    317      JSR  KEYWAIT    ;TEST FOR KEYPRESS
FD1A: B0 08      318      BCS  KEYIN2    ;GO GET IT
FD1C: A5 69      319      LDA  CURSOR    ;GIVE THEM AN UNDERSCORE FOR A TIME
FD1E: 20 9D FC    320      JSR  DISPLAY
FD21: 20 35 FD    321      JSR  KEYWAIT    ;GO SEE IF KEYPRESSED
FD24: 68          322      KEYIN2 PLA
FD25: 08          323      PHP          ;SAVE KEYPRESS STATUS
FD26: 48          324      PHA
FD27: 20 9D FC    325      JSR  DISPLAY
FD2A: 68          326      PLA
FD2B: 28          327      PLP
FD2C: 90 E8      328      RCC  KEYIN1
FD2E: AD 00 C0    329      KEYIN3 LDA  KBD          ;READ KEYBOARD
FD31: 20 10 C0    330      KEYIN4 BIT  KBDSTRB    ;CLEAR KEYBOARD STROBE
FD34: 60          331      RTS
FD35: E6 62      332      KEYWAIT INC  TBAS4L    ;JUST KEEP COUNTING
FD37: D0 09      333      BNE  KWAIT2
FD39: E6 63      334      INC  TBAS4H
FD3B: A9 7F      335      LDA  #$7F        ;TEST FOR DONE
FD3D: 18          336      CLC
FD3E: 25 63      337      AND  TBAS4H
FD40: F0 05      338      BEQ  KEYRET    ;RETURN IF TIMED OUT
FD42: 0E 00 C0    339      KWAIT2 ASL  KBD
FD45: 90 EE      340      BCC  KEYWAIT
FD47: 60          341      KEYRET RTS
FD48:           342      *
FD48:           343      *
FD48:           344      ESC3 EQU  *
FD48: 20 77 FD    345      JSR  GOESC
FD4B: A5 68      346      ESCAPE LDA  MODES    ;GET TO + SIGN FOR CURSOR MOVES
FD4D: 29 B0      347      AND  #$B0
FD4F: 49 AB      348      EOR  #$AD
FD51: 85 69      349      STA  CURSOR
FD53: 20 0C FD    350      ESC1  JSR  RDKEY    ;READ NEXT CHARACTER
FD56: A0 08      351      LDY  #8          ;TEST FOR ESCAPE COMMAND
FD58: D9 F0 FF    352      ESC2  CMP  ESCTABL,Y
FD5B: F0 EB      353      BEQ  ESC3
FD5D: 88          354      DEY
FD5E: 10 FB      355      BPL  ESC2    ;LOOP TIL FOUND OR DONE
FD60:           356      *
FD60: A9 80      357      RDCHAR  LDA  #$80    ;GO READ A CHARACTER
FD62: 25 68      358      AND  MODES
FD64: 85 69      359      STA  CURSOR    ;SAVE STANDARD CURSOR
FD66: 20 0C FD    360      JSR  RDKEY
FD69: C9 9B      361      CMP  #$9B    ;ESCAPE CHARACTER?
FD6B: F0 DE      362      BEQ  ESCAPE
FD6D: C9 95      363      CMP  #$95    ;FORWARD COPY?
FD6F: D0 D6      364      BNE  KEYRET
FD71: 20 88 FD    365      JSR  PICK    ;GET CHARACTER FROM SCREEN

```

4,383,296

97

98

FC68: 10	222	CLC		
FC6C: 49 01	223	ADC	#1	CALCULATE SOURCE LINE
FC6E: C9 58	224	CMF	WINDTH	IS IT THE LAST LINE?
FC70: D0 15	225	BCC	LASTLN	YES, CLEAR IT
FC72: 48	226	PHA		SAVE AS NEXT DESTINATION LINE
FC73: 20 B1 FB	227	JSR	SETCV	GET BASE ADDR FOR SOURCE LINE
FC76: A5 59	228	LDA	RHARGIN	MOVE SOURCE TO DESTINATION
FC78: 4A	229	LSR	A	DIVIDE BY 2
FC79: A8	230	TAY		
FC7A: 88	231	DEY		DONE YET?
FC7B: 30 E4	232	BMI	SCRL1	YES, DO NEXT LINE
FC7D: B1 5E	233	LDA	(BAS4L),Y	
FC7F: 91 62	234	STA	(TBAS4L),Y	
FC81: B1 60	235	LDA	(BAS8L),Y	MOVE BOTH PAGES
FC83: 91 64	236	STA	(TBAS8L),Y	
FC85: 90 F3	237	BCC	SCRL3	BRANCH ALWAYS
FC87: A5 58	238	LDA	RHARGIN	BLANK FILL THE LAST LINE
FC89: 4A	239	CLEOL1	LSR	A
FC8A: A8	240	TAY		DIVIDE BY 2
FC8B: B0 04	241	BCC	CLEOL2	
FC8D: A5 66	242	LDA	FORGND	(NORMALLY A SPACE)
FC8F: 91 5E	243	STA	(BAS4L),Y	
FC91: A5 67	244	CLEOL2	LDA	BKGD
FC93: 91 60	245	STA	(BAS8L),Y	(IF 80 COLUMNS, ALSO A SPACE)
FC95: C8	246	INY		
FC96: 98	247	TYA		TEST FOR END OF LINE
FC97: 0A	248	ASL	A	MULT BY 2 AGAIN
FC98: C9 59	249	CMF	RHARGIN	
FC9A: 90 ED	250	BCC	CLEOL1	CONTINUE IF MORE TO DO
FC9C: 60	251	RTS		ALL DONE
FC9D:	252 *			
FC9D: 24 68	253	DISPLAY	BIT	MODE5
FC9F: 7C 0C	254	BVS	DISPL60	STORE THE SINGLE CHARACTER AND RETURN
FCA1: 46 5C	255	JSR	CH	INSURE PROPER 40 COLUMN DEL. LAY
FCA3: 06 5C	256	ASL	CH	BY DROPPING BIT 0
FCA5: 20 AD FC	257	JSR	DISPL60	DISPLAY IN \$400 PAGE
FCA8: A5 67	258	LDA	BKGD	ALSO SET BACKGROUND COLOR
FCAA: 91 60	259	DSPBKGD	STA	(BAS8L),Y
FCAE: 60	260	RTS		
FCAE:	261 *			
FCAE: 48	262	DISPL60	PHA	PRESERVE CHARACTER
FCAE: A5 5C	263	LDA	CH	DETERMINE WHICH PAGE
FCB0: 4A	264	LSR	A	
FCB1: A8	265	TAY		
FCB2: 68	266	PLA		
FCB3: B0 F5	267	BCC	DSPBKGD	BRANCH IF \$800 PAGE
FCB5: 91 5E	268	STA	(BAS4L),Y	
FCB7: 60	269	RTS		
FCB8:	270 *			
FCB8: B1 7E	271	NOTCH	LDA	(INBUF),Y
FCBA: 20 25 FC	272	JSR	COUT	ECHO CHARACTER
FCBD: C9 88	273	CMF	##88	BACKSPACE?
FCBF: F0 1D	274	BEG	BKSPCE	
FCC1: C9 98	275	CMF	##98	CANCEL?
FCC3: F0 08	276	BEG	CANCEL	
FCC5: E6 80	277	INC	TEMP	
FCC7: A5 80	278	LDA	TEMP	
FCC9: C9 50	279	CMF	#INBUFLN	
FCCB: D0 17	280	BNE	NXTCHAR	NO WRAP AROUND ALLOWED
FCCD: A9 DC	281	CANCEL	LDA	##DC
FCCF: 20 25 FC	282	JSR	COUT	OUTPUT BACKSLASH
FCD2: 20 EF FC	283	JSR	CROUT	
FCD5:	284	GETLNZ	EGU	*
FCD5: A5 6B	285	GETLN	LDA	PROMPT
FCD7: 20 25 FC	286	JSR	COUT	
FCD9: A0 01	287	LDY	#1	
FCD9: 84 80	288	STY	TEMP	START AT BEGINNING OF INBUF
FCD9: A4 80	289	BKSPCE	LDY	TEMP
FCE0: F0 F3	290	BEG	GETLN	
FCE2: C6 80	291	DEC	TEMP	BACK UP INPUT BUFFER
FCE4: 20 60 FD	292	NXTCHAR	JSR	RDCHAR
FCE7: A4 80	293	LDY	TEMP	GET INPUT



4,383,296

99

100

FCE9: 91 7E	294	STA	(INBUF),Y	
FCEB: C9 8D	295	CMP	##8D	
FCED: D0 C9	296	BNE	NOTCR	
FCEF:	297	CROUT	EQU	*
FCE9: 2C 00 C0	298	BIT	KBD	; TEST FOR START/STOP
FCF2: 10 13	299	BPL	NOSTOP	
FCF4: 20 2E FD	300	JSR	KEYIN3	; READ KBD
FCF7: C9 A0	301	CMP	##A0	; IS IT A SPACE?
FCF9: F0 07	302	BEG	STOPLST	; YES, PAUSE TIL NEXT KEYPRESS.
FCFB: C9 8D	303	CMP	##8D	; QUIT THIS OPERATION?
FCFD: D0 08	304	BNE	NOSTOP	; NO, IGNORE THIS KEY.
FCFF: 4C 8B FA	305	JMP	ERRR2	; YES, RESTART
FD02: AD 00 C0	306	STOPLST	LDA	KBD
FD05: 10 FB	307	BPL	STOPLST	
FD07: A9 8D	308	NOSTOP	LDA	##8D
FD09: 4C 25 FC	309	JMP	COUT	
FD0C:	310	*		
FD0C: 6C 70 00	311	RDKEY	JMP	(KSWL)
FD0F:	312	*		
FD0F: A9 7F	313	KEYIN	LDA	##7F ; MAKE SURE FIRST IS CURSOR
FD11: 85 63	314	STA	TBAS4H	
FD13: 20 88 FD	315	JSR	PICK	; GO READ SCREEN
FD16: 48	316	KEYIN1	PHA	; SAVE CHR AT CURSOR POSITION
FD17: 20 35 FD	317	JSR	KEYWAIT	; TEST FOR KEYPRESS
FD1A: B0 08	318	BCS	KEYIN2	; GO GET IT
FD1C: A5 69	319	LDA	CURSOR	; GIVE THEM AN UNDERSCORE FOR A TIME
FD1E: 20 9D FC	320	JSR	DISPLAY	
FD21: 20 35 FD	321	JSR	KEYWAIT	; GO SEE IF KEYPRESSED
FD24: 68	322	KEYIN2	PLA	
FD25: 08	323	PHP		; SAVE KEYPRESS STATUS
FD26: 48	324	PHA		
FD27: 20 9D FC	325	JSR	DISPLAY	
FD2A: 68	326	PLA		
FD2B: 28	327	PLP		
FD2C: 90 E8	328	RCC	KEYIN1	
FD2E: AD 00 C0	329	KEYIN3	LDA	KBD ; READ KEYBOARD
FD31: 2C 10 C0	330	KEYIN4	RIT	KBDSTR8 ; CLEAR KEYBOARD STROBE
FD34: 60	331	RTS		
FD35: E6 62	332	KEYWAIT	INC	TBAS4L ; JUST KEEP COUNTING
FD37: D0 09	333	BNF	KWAIT2	
FD39: E6 63	334	INT	TBAS4H	
FD3B: A9 7F	335	LDA	##7F	; TEST FOR DONE
FD3D: 18	336	CLC		
FD3E: 25 63	337	AND	TBAS4H	
FD40: F0 05	338	BEG	KEYRET	; RETURN IF TIMED OUT
FD42: 0E 00 C0	339	KWAIT2	ASL	KBD
FD45: 90 EE	340	BCC	KEYWAIT	
FD47: 60	341	KEYRET	RTS	
FD48:	342	*		
FD48:	343	*		
FD48:	344	ESC3	EQU	*
FD48: 20 77 FD	345	JSR	GOESC	
FD4B: A5 68	346	ESCAPE	LDA	MODES ; SET TO + SIGN FOR CURSOR MOVES
FD4D: 29 80	347	AND	##80	
FD4F: 49 AB	348	EOR	##AD	
FD51: 85 69	349	STA	CURSOR	
FD53: 20 0C FD	350	ESC1	JSR	RDKEY ; READ NEXT CHARACTER
FD56: A0 08	351	LDY	##8	; TEST FOR ESCAPE COMMAND
FD58: D9 F0 FF	352	ESC2	CMP	ESCTABL,Y
FD5B: F0 EB	353	BEG	ESC3	
FD5D: 88	354	DEY		
FD5E: 10 FB	355	BPL	ESC2	; LOOP TIL FOUND OR DONE
FD60:	356	*		
FD60: A9 80	357	RDCHAR	LDA	##80 ; GO READ A CHARACTER
FD62: 25 68	358	AND	MODES	
FD64: 85 69	359	STA	CURSOR	; SAVE STANDARD CURSOR
FD66: 20 0C FD	360	JSR	RDKEY	
FD69: C9 9B	361	CMP	##9B	; ESCAPE CHARACTER?
FD6B: F0 DE	362	BEG	ESCAPE	
FD6D: C9 95	363	CMP	##95	; FORWARD COPY?
FD6F: D0 D6	364	BNE	KEYRET	
FD71: 20 88 FD	365	JSR	PICK	; GET CHARACTER FROM SCREEN

4,383,296

101

102

```

FD74 09 80      366      ORA  ##80      ;SET TO NORMAL ASCII
FD76 60          367      RTS
FD77:          368 *
FD77: A9 FB      369 GOESC LDA  ##CLSCRN
FD79 48          370      PHA
FD7A B9 7F FD    371      LDA  ESCVECT,Y
FD7D 48          372      PHA
FD7E 60          373      RTS
FD7F:          374 *
FD7F 8D          375 ESCVECT DFB CLEOL-1
FD80 70          376      DFB CLEOP-1
FD81 60          377      DFB CLSCRN-1
FD82 40          378      DFB COL40-1
FD83 48          379      DFB COL80-1
FD84 08          380      DFB CURLEFT-1
FD85 06          381      DFB CURRIGHT-1
FD86 08          382      DFB CURDOWN-1
FD87 A3          383      DFB CURUP-1
FD88:          384 *
FD88 A5 5C       385 PICK LDA  CH      ;GET A CHARACTER AT CURRENT CURSOR POSITION
FD8A 4A          386      LSR  A      ;DETERMINE WHICH PAGE
FD8B A8          387      TAY
FD8C 24 68       388      BIT  MODES   ;AND IF 80 COLUMN MODE
FD8E 50 05       389      BVC  PICK40  ;FORGET CARRY IF 40 COLUMNS
FD90 90 02       390      BCC  PICK40  ;GET CHARACTER FROM #400 PAGE
FD92 B1 60       391      LDA  (BASBL),Y
FD94 60          392      RTS
FD95 B1 5E       393 PICK40 LDA  (BAS4L),Y
FD97 60          394      RTS
FD98:          395 *
FD98:          2 CLDSTRT EQU  *
FD98: A9 03      3      LDA  ##3
FD9A 8D D0 FF    4      STA  $FFD0      ;ZERO PAGE IS ON 3!
FD9D:          5 SETCF  EQU  *
FD9D: D8         6      CLD          ;OF COURSE!
FD9E A2 03      7      LDX  #3
FDA0 86 7F      8      STX  INBUF+1
FDA2 BD BC FF   9 SETUP1 LDA  NMIRG,X
FDA5 9D CA FF   10     STA  $FFCA,X
FDAQ BD B4 FF   11     LDA  HOOKS,X
FDAQ 95 6E      12     STA  CSWL,X
FDAQ BD B8 FF   13     LDA  VBOUNDS,X
FDB0 95 58      14     STA  LMARGIN,X
FDB2 CA         15     DEX
FDB3 10 ED      16     BPL  SETUP1
FDB5 85 B2      17     STA  IBDRVN
FDB7 A9 A0      18     LDA  #4A0      ;INPUT BUFFER AT $3A0
FDB9 85 7E      19     STA  INBUF
FDBB A9 60      20     LDA  ##60
FDBD 85 B1      21     STA  IB SLOT
FDBF A9 FF      22     LDA  ##FF
FDC1 85 68      23     STA  MODES
FDC3 20 4F FB    24     JSR  COL40      ;SET 40 COLUMNS, CLEAR SCREEN
FDC6:          25 *
00A0:          27 ADR      EQU  $A0
00A0:          28 CPORTL  EQU  ADR
00A1:          29 CPORTH  EQU  ADR+1
00A2:          30 CTEMP   EQU  ADR+2
00A3:          31 CTEMP1  EQU  ADR+3
00A4:          32 YTEMP    EQU  ADR+4
00B4:          33 ROWTEMP  EQU  ADR+20
C0DB:          34 CWRTON   EQU  1C0DB
C0DA:          35 CWRTOFF  EQU  1C0DA
FFEC:          36 CTEMP2  EQU  $FFEC
FFED:          37 CTEMP3  EQU  $FFED
FDC6:          38 *
FDC6:          39 *
FDC6 A9 78      40 GENENTR LDA  ##78      ;INIT SCREEN INDX LOCATIONS
FDC8 B5 A0      41      STA  CPORTL
FDCA A9 08      42      LDA  #8
FDCC B5 A1      43      STA  CPORTH

```

**4,383,296**

103		104	
FDDC A9 F0	44	LDA #240	SET UP INDEX TO CHRSET
FDD0 B5 A4	45	STA YTEMP	
FDD2 A9 00	46	LDA #0	
FDD4 A4	47	LDA	
FDD5 95 B4	48	LDI TEMP	ROWTEMP, X
FDD7 E8	49	INR	
FDD8 E0 20	50	CPA	##20
FDDA D0 5F	51	BNE ZIPTEMPS	
FDDC A9 05	52	LDA #1	FAKE THE FIRST BIT PATTERN
FDD5 10	53	AND	PHANTOM 9TH BIT SHIFTER AS BIT 0
FDD7 08	54	PHP	
FDD8 4E	55	PHA	
FDE1 86 A2	56	GENASC	CTEMP
FDE3 A7 07	57	CASCI1	CODES FOR THE FIRST PASS
FDE5 A6 A2	58	CASCI2	
FDE7 0A	59	CASCI3	
FDE9 47 A7	60	STA	PORTL
FDEA 70	61	INR	
FDEB 08	62	DEY	
FDEE 00 00	63	EMI	CASCI4
FDEE 00 00	64	EX	##3
FDEE 00 00	65	MI	CASCI12
FDEE 00 00	66	MI	CASCI12
FDEE 00 00	67	MI	CASCI12
FDEE 00 00	68	MI	CASCI12
FDEE 00 00	69	MI	CASCI12
FDEE 00 00	70	MI	CASCI12
FDEE 00 00	71	MI	CASCI12
FDEE 00 00	72	MI	CASCI12
FDEE 00 00	73	MI	CASCI12
FDEE 00 00	74	MI	CASCI12
FDEE 00 00	75	MI	CASCI12
FDEE 00 00	76	MI	CASCI12
FDEE 00 00	77	MI	CASCI12
FDEE 00 00	78	MI	CASCI12
FDEE 00 00	79	MI	CASCI12
FDEE 00 00	80	MI	CASCI12
FDEE 00 00	81	MI	CASCI12
FDEE 00 00	82	MI	CASCI12
FDEE 00 00	83	MI	CASCI12
FDEE 00 00	84	MI	CASCI12
FDEE 00 00	85	MI	CASCI12
FDEE 00 00	86	MI	CASCI12
FDEE 00 00	87	MI	CASCI12
FDEE 00 00	88	MI	CASCI12
FDEE 00 00	89	MI	CASCI12
FDEE 00 00	90	MI	CASCI12
FDEE 00 00	91	MI	CASCI12
FDEE 00 00	92	MI	CASCI12
FDEE 00 00	93	MI	CASCI12
FDEE 00 00	94	MI	CASCI12
FDEE 00 00	95	MI	CASCI12
FDEE 00 00	96	MI	CASCI12
FDEE 00 00	97	MI	CASCI12
FDEE 00 00	98	MI	CASCI12
FDEE 00 00	99	MI	CASCI12
FDEE 00 00	100	MI	CASCI12
FDEE 00 00	101	MI	CASCI12
FDEE 00 00	102	MI	CASCI12
FDEE 00 00	103	MI	CASCI12
FDEE 00 00	104	MI	CASCI12
FDEE 00 00	105	MI	CASCI12
FDEE 00 00	106	MI	CASCI12
FDEE 00 00	107	MI	CASCI12
FDEE 00 00	108	MI	CASCI12
FDEE 00 00	109	MI	CASCI12
FDEE 00 00	110	MI	CASCI12
FDEE 00 00	111	MI	CASCI12
FDEE 00 00	112	MI	CASCI12
FDEE 00 00	113	MI	CASCI12
FDEE 00 00	114	MI	CASCI12
FDEE 00 00	115	MI	CASCI12
FDEE 00 00	116	MI	CASCI12
FDEE 00 00	117	MI	CASCI12
FDEE 00 00	118	MI	CASCI12
FDEE 00 00	119	MI	CASCI12
FDEE 00 00	120	MI	CASCI12
FDEE 00 00	121	MI	CASCI12
FDEE 00 00	122	MI	CASCI12
FDEE 00 00	123	MI	CASCI12
FDEE 00 00	124	MI	CASCI12
FDEE 00 00	125	MI	CASCI12
FDEE 00 00	126	MI	CASCI12
FDEE 00 00	127	MI	CASCI12
FDEE 00 00	128	MI	CASCI12
FDEE 00 00	129	MI	CASCI12
FDEE 00 00	130	MI	CASCI12
FDEE 00 00	131	MI	CASCI12
FDEE 00 00	132		

4,383,296

105

106

```

FE44:A9 01      114 GENDONE LDA #1          ;SET NORMAL MODE
FE46:B5 A2      115      STA CTEMP
FE48:A9 60      116 GEN1  LDA ##60        ;PREPARE TO SEND BYTES TO CHARACTER
FE4A:20 DB C0   117      BIT CURTCN      ;GENERATOR RAM
FE4D:20 AE FF   118      JRP VRETRCE     ;WAIT FOR NEXT VERTICAL RETRACE
FE50:A9 20      119      LDA ##20        ;WAIT AGAIN
FE52:20 AE FE   120      JSR VRETRCE
FE55:20 DA C0   121      BIT CWRTOFF     ;CHARACTERS ARE NOW LOADED
FE58:20 8B FE   122      JSR ALTCHR      ;REPEAT THIS SET FOR OTHER 64 CHARACTERS
FE5B:05 A2      123      BCC CTEMP      ;HAVE WE DONE ALTERNATES YET?
FE5D:10 16      124      BPL GEN2        ;NO, DO IT!
FE5F:A9 08      125      LDA #10         ;DUMP ASCII VALUES FOR NEXT SET
FE61:B5 A1      126      STA CPORTH
FE63:A0 07      127 NXTASCI LDY #7         ;THE USUAL COUNTDOWN
FE65:B1 A0      128 NXTASCI LDA (CPORTL),Y
FE67:18         129      DEC Y
FE68:69 0A      130      ADC ##8
FE6A:01 00      131      STA (CPORTL),Y
FE6C:88         132      DEY
FE6D:10 16      133      BPL NXTASCI
FE6F:20 99 FF   134      JSR NXTPORT
FE72:90 0F      135      BCC NXTASCI
FE74:00         136      RTS
FE76:A0 00      137 GEN2  LDY #33        ;SETUP ALTERNATE WITH UNDERLINES
FE77:A9 7F      138      LDA ##7F
FE79:99 FC 05   139 UNDER STA $5FC,Y
FE7C:99 FC 07   140      STA $7FC,Y
FE7F:88         141      DEY
FE80:10 16      142      BPL UNDER
FE82:A0 00      143      LDA ##8
FE84:01 A1      144      STA CPORTH
FE86:00 00      145      BNE GEN1
FE88         146 *
FE8A:A0 01      147 ALTCHR LDY #7          ;ADJUST ASCII FOR ALTERNATE SET
FE8C:B1 A0      148 ALTCH1 LDA (CPORTL),Y
FE8E:49 20      149      EOR ##20        ;$20-->0 $40-->$60
FE90:91 A0      150      STA (CPORTL),Y
FE92:80         151      DEY
FE94:10 F7      152      BPL ALTCH1      ;ADJUST THEM ALL
FE96:20 99 FE   153      JSR NXTPORT
FE98:90 F0      154      BCC ALTCHR
FE9A:60         155      RTS
FE9C         156 *
FE9E:A5 A0      157 NXTPORT LDA CPORTL     ;CONVERT $78->$FB OR $FB-$78
FEA0:49 B0      158      EOR ##B0
FEA2:85 A0      159      STA CPORTL
FEA4:30 02      160      BMI NOHIGH
FEA6:E6 A1      161      INC CPORTH      ;IF =C THEN =4
FEA8:A5 A1      162 NOHIGH LDA CPORTH
FEAA:C9 0C      163      CMP ##C
FEAC:D0 04      164      BNE PORTDN
FEAE:A9 04      165      LDA #14
FEAB:B5 A1      166      STA CPORTH
FEAD:60         167 PORTDN RTS
FEAE         168 *
FEAF         169 *
FEAE:B5 A3      170 VRETRCE STA CTEMP1     ;SAVE BITS TO BE STORED
FEB0:AD EC FF   171      LDA CB2CTRL      ;CONTROL PORT FOR 'CB2'
FEB3:29 3F      172      AND ##3F        ;RESET HI BITS TO 0
FEB5:05 A3      173      ORA CTEMP1
FEB7:8D EC FF   174      STA CB2CTRL
FEB9:A9 08      175      LDA ##8          ;TEST VERTICAL RETRACE
FEBB:8D ED FF   176      STA CB2INT
FEBF:2C ED FF   177 VWAIT BIT CB2INT      ;WAIT FOR RETRACE
FEC2:F0 FB      178      BEQ VWAIT
FEC4:60         179      RTS
FEC5         180 *
FEC5         181 CHRSET EQU *

```

107	4,383,296	108
FEC5 F0 01 82 182	DFB \$F0, \$01, \$82, \$18	
FEC8 18		
FEC9 40 84 81 183	DFB \$40, \$84, \$81, \$2F	
FEC0 2F		
FEC0 58 44 81 184	DFB \$58, \$44, \$81, \$29	
FED0 29		
FED1 02 1E 01 185	DFB \$02, \$1E, \$01, \$91	
FED4 91		
FED5 7C 1F 49 186	DFB \$7C, \$1F, \$49, \$30	
FED8 30		
FED9 8A 08 43 187	DFB \$8A, \$08, \$43, \$14	
FEDC 14		
FEDD 31 2A 22 188	DFB \$31, \$2A, \$22, \$13	
FEE0 13		
FEE1 E3 F7 C4 189	DFB \$E3, \$F7, \$C4, \$91	
FEE4 91		
FEE5 48 A2 DA 190	DFB \$48, \$A2, \$DA, \$24	
FEE8 24		
FEE9 C6 4A 62 191	DFB \$C6, \$4A, \$62, \$8C	
FEEC 8C		
FEED 24 C6 F8 192	DFB \$24, \$C6, \$F8, \$63	
FEF0 63		
FEF1 8C 01 46 193	DFB \$8C, \$C1, \$46, \$17	
FEF4 17		
FEF5 52 8A AF 194	DFB \$52, \$8A, \$AF, \$16	
FEF8 16		
FEF9 14 E3 33 195	DFB \$14, \$E3, \$33, \$31	
FEFC 31		
FEFD C6 F8 DC 196	DFB \$C6, \$F8, \$DC, \$73	
FEFD 73		
FF01 3F 46 17 197	DFB \$3F, \$46, \$17, \$62	
FF04 62		
FF05 8C 21 E6 198	DFB \$8C, \$21, \$E6, \$18	
FF08 18		
FF09 6A 8D 61 199	DFB \$6A, \$8D, \$61, \$CF	
FF0C 0F		
FF0D 18 62 74 200	DFB \$18, \$62, \$74, \$D1	
FF10 D1		
FF11 B9 18 49 201	DFB \$B9, \$18, \$49, \$4C	
FF14 4C		
FF15 91 C0 F3 202	DFB \$91, \$C0, \$F3, \$09	
FF18 09		
FF19 2C 91 C0 203	DFB \$2C, \$91, \$C0, \$14	
FF1C 14		
FF1D 1D 8C EF 204	DFB \$1D, \$8C, \$EF, \$07	
FF20 27		
FF21 17 48 82 205	DFB \$17, \$48, \$88, \$31	
FF24 31		
FF25 84 1E DF 206	DFB \$84, \$1E, \$DF, \$0B	
FF28 0B		
FF29 31 84 F8 207	DFB \$31, \$84, \$F8, \$FE	
FF2C FE		
FF2D 72 3E 9E 208	DFB \$72, \$3E, \$9E, \$17	
FF30 17		
FF31 62 8C FD 209	DFB \$62, \$8C, \$FD, \$C7	
FF34 C7		
FF35 50 E3 0B 210	DFB \$50, \$E3, \$0B, \$51	

				109	4,383,296	110
FF39	51					
FF39	05	E8	00	211	DFB	\$05, \$E8, \$08, \$73
FF3C	7D					
FF3D	18	0C	42	212	DFB	\$18, \$0C, \$42, \$3E
FF40	3E					
FF41	01	02	20	213	DFB	\$01, \$02, \$20, \$42
FF44	42					
FF45	3F	41	18	214	DFB	\$3F, \$41, \$18, \$8C
FF48	0C					
FF49	08	00	70	215	DFB	\$08, \$00, \$70, \$EE
FF4C	EE					
FF4D	00	11	1	216	DFB	\$00, \$11, \$11, \$21
FF50	21					
FF51	11	02	8	217	DFB	\$11, \$02, \$02, \$3C
FF54	00					
FF55	21	31	0	218	DFB	\$21, \$31, \$02, \$E0
FF58	E0					
FF59	1C	00	0	219	DFB	\$1C, \$00, \$08, \$B9
FF5C	89					
FF5D	80	62	14	220	DFB	\$80, \$62, \$14, \$1F
FF60	1F					
FF61	46	A2	DE	221	DFB	\$46, \$A2, \$DE, \$43
FF64	43					
FF65	2C	04	88	222	DFB	\$2C, \$04, \$88, \$BE
FF68	BF					
FF69	FF	CE	7D	223	DFB	\$FF, \$CE, \$7D, \$37
FF6C	37					
FF6D	49	88	95	224	DFB	\$49, \$88, \$95, \$18
FF70	12					
FF71	98	09	62	225	DFB	\$98, \$09, \$62, \$D1
FF74	01					
FF75	44	E8	88	226	DFB	\$44, \$E8, \$88, \$FB
FF78	FB					
FF79	02	90	40	227	DFB	\$02, \$90, \$40, \$00
FF7C	00					
FF7D	10	E0	03	228	DFB	\$10, \$E0, \$03, \$02
FF80	02					
FF81	00	40	00	229	DFB	\$00, \$40, \$00, \$00
FF84	00					
FF85	08	00	00	230	DFB	\$08, \$00, \$00, \$2B
FF88	24					
FF89	10	42	44	231	DFB	\$10, \$42, \$44, \$25
FF9C	25					
FF8D	82	88	2F	232	DFB	\$82, \$88, \$2F, \$48
FF90	48					
FF91	25	44	10	233	DFB	\$25, \$44, \$10, \$82
FF94	82					
FF95	02	00	2F	234	DFB	\$02, \$00, \$2F, \$5A
FF98	5A					
FF99	40	45	02	235	DFB	\$40, \$45, \$02, \$8E
FF9C	8E					
FF9D	64	50	90	236	DFB	\$64, \$50, \$90, \$01
FFA0	01					
FFA1	3E	26	42	237	DFB	\$3E, \$26, \$42, \$80
FFA4	80					
FFA5	21	80	00	238	DFB	\$21, \$80, \$00, \$05
FFA8	05					
FFA9	00	FB	80	239	DFB	\$00, \$FB, \$80, \$00
FFAC	00					
FFAD	05	08	FB	240	DFB	\$05, \$08, \$FB, \$80
FFB0	80					
FFB1	28	05	88	241	DFB	\$28, \$05, \$88
FFB4				242 *		
FFB4				243 HOOKS	EGU	*
FFB4	F2	FB		244	DW	COUT2
FFB6	0F	FD		245	DW	KEYIN
FFB8				246 *		

4,383,296

111

112

FFB8 247 VBOUNDS EGU \*  
 FFB8 00 50 00 248 DFB \$0, \$50, 0, \$18  
 FFB8 1B  
 FFB8 249 \*  
 FFB8 4C 89 F5 250 NMING JMP RECON IN DIAGNOSTICS  
 FFB8 40 251 RTI  
 FFB8 C3 CF D0 252 ASC COPYRIGHT JANUARY, 1980 APPLE COMPUTER INC. JRH  
 FFB8 D9 D2 C9  
 FFB8 C7 C8 D4  
 FFB8 A0 CA C  
 FFB8 CF D5 C  
 FFB8 D2 D9 AC  
 FFB8 A0 B1 B9  
 FFB8 B8 B0 A0  
 FFB8 A0 C1 D0  
 FFB8 D0 C0 C5  
 FFB8 A0 C3 CF  
 FFB8 CD D0 D5  
 FFB8 D4 C5 D2  
 FFB8 A0 C9 CE  
 FFB8 C3 AE AE  
 FFB8 CA D2 C8  
 FFB8  
 FFB8 253 CHN MONVECT  
 FFB8 1 \*  
 FFB8 CC 2 ESCTABL DFB \$CC  
 FFB8 D0 3 DFB \$D0  
 FFB8 D3 4 DFB \$D3  
 FFB8 B4 5 DFB \$B4  
 FFB8 B8 6 DFB \$B8  
 FFB8 B8 7 DFB \$B8  
 FFB8 95 8 DFB \$95  
 FFB8 8A 9 DFB \$8A  
 FFB8 8B 10 DFB \$8B  
 FFB8 00 11 DFB \$00  
 FFB8 12 \*  
 FFB8 CA FF 13 NMI DW \$FFCA  
 FFB8 EE F4 14 RESET DW DIAGN FIRST DIAGNOSTICS  
 FFB8 CD FF 15 IRQ DW \$FFCD  
 FFB8 0000 16 \*

J. Richard (Dick)  
 Huston

(also worked on  
 Apple III SOS)

\*\*\* SUCCESSFUL ASSEMBLY: NO ERRORS

75 A1H	74 A1L	F9D4 A1PC	F9D7 A1PC1
77 A2H	76 A2L	79 A3H	78 A3L
7B A4H	7A A4L	A0 ADR	FEB8 ALTC1
FEB8 ALTCHR	FB2C ASC1	FB38 ASC2	FB46 ASC3
FA06 ASCDONE	FA09 ASCI10	F9DF ASCI11	F9E1 ASCI12
FA07 ASCI1	F9F2 ASCI13	5F BAS4H	5E BAS4L
61 BAS8H	60 BAS8L	FC05 BASCALC1	?FBB3 BASCALC
FC3D BELL1	FC3E BELL2	FC43 BELL3	FC3A BELL
FA15 BITOFF	FA11 BITON	67 BKOND	FCDE BKSPCE
FAA0 BL1	F479 BLOCKIO	FC19 BSCLC2	FCCD CANCEL
?FB9B CARRAGE	FFEC CB2CTRL	FFED CB2INT	FE01 CBYTES
FE05 CCOLMS	5C CH	FEC5 CHRSET	?FA0A CKMDE
?FD9B CLDSTRT	FC89 CLEOL1	FB8E CLEOL	FC91 CLEOL2
FB71 CLEOP	FB7A CLEOP1	FB69 CLSCRN	F91C CMDSRCH
F96C CMDTAB	F97C CMDVEC	FB4F COL40	FB49 COL80
FB93 CONTROL	FC33 COUT1	FBF2 COUT2	FC25 COUT
A1 CPORTH	A0 CPORTL	F9FB CRCHK	FA26 CRMON
FCEF CROUT	FE07 CSHFT	? 6F CSWH	6E CSWL
A3 CTEMP1	A2 CTEMP	FC24 CTRLRET	FBB3 CURDN1
FBC9 CURDOWN	FB87 CURIGHT	FBD9 CURLEFT	69 CURSOR

113

4,383,296

114

FBAE CURUP1	FBA4 CURUP	5D CV	CODA CWRTOFF
CODB CWRTON	FA91 DEST	F4EE DIAGN	F941 DIGIT
F96D DIGRET	FBFC DISPLAYX	FC9D DISPLAY	FE28 DONE
FCAA DSPBKOND	FCAD DSPL80	FAB7 DUMMY	FAFC DUMPO
FB09 DUMP1	FB0C DUMP2	FB1C DUMP3	FAE9 DUMPS
FB21 DUMPASC	?FAF9 DUMP	?F901 ENTRY	FA8B ERROR2
FABE ERROR	FAF7 ERROR1	?FD53 ESC1	FD58 ESC2
FD48 ESC3	FD4B ESCAPE	FFF0 ESCTABL	FD7F ESCVECT
66 FORGND	FDE3 GASC11	FDE5 GASC12	FDE7 GASC13
FDF4 GASC14	FE48 GEN1	FE75 GEN2	FDE1 GENASC
FE44 GENDONE	?FDC6 GENENTR	FCD5 GETLN	FCD5 GETLNZ
F92C GETNUM	FD77 GOESC	FA7D GO	FFB4 HOOKS
85 IDBUFF	87 IBCMD	82 IBDVRN	81 IBLSLOT
50 INBUFLEN	7E INBUF	?FBFF INCHORZ	?FFFE IRQ
FA7D JUMP	CO10 KBDSTRB	CO00 KBD	FD16 KEYIN1
FD24 KEYIN2	FD2E KEYIN3	?FD31 KEYIN4	FDOF KEYIN
FD47 KEYRET	FD35 KEYWAIT	? 71 KSWH	70 KSWL
FD42 KWAIT2	FC87 LASTLN	FBDF LEFT80	FBE9 LEFTUP
58 LMARGIN	FC52 LNFD	69 MASK	FA52 MISMATCH
68 MODES	F904 MON	F908 MONZ	FA2C MOVE
FA31 MOVNXT	FFBC NMIRG	?FFFA NMI	FEA3 NOHIGH
FD07 NOSTOP	FCB8 NOTCR	FADE NOVER	F992 NXTA1
F98C NXTA4	FE65 NXTASC2	FE63 NXTASCI	F94F NXTBAS
F947 NXTBIT	F959 NXTBS2	FCE4 NXTCHAR	F932 NXTCHR
F915 NXTINP	FC02 NXTLIN	FE99 NXTPORT	F9DE OLDPC
? 73 PCH	72 PCL	FD95 PICK40	FD88 PICK
FEAD PORTDN	FA6E PRA1BYTE	F9C2 PRBYCOL	F9AC PRBYTE
FA70 PRBYTSP	?F9C5 PRCOLON	F9BF PRHEX2	F9B7 PRHEXZ
?F9B5 PRHEX	FA61 PRINTA1	6D PROMPT	FA73 PRSPC
FD60 RDCHAR	FDOC RDKEY	FAC0 READ	F689 RECON
FA19 REPEAT	FA21 REPEAT1	?FFFC RESET	F7FF RET1
F900 RET2	F882 RET3	F9AD RETA1	FDBD RIGHT1
59 RMARGIN	84 ROWTEMP	FAB3 RWERROR	FAC7 RWLOOP
?FAC5 SAVCMD	F912 SCAN	FC61 SCRL1	FC63 SCRL2
FC7A SCRL3	58 SCRNL0C	FC5B SCROLL	FA9A SEP
FB5B SET80A	FB53 SET80	FB67 SET80B	FBC3 SETCHZ
FBB1 SETCV	?FBC7 SETCVH	FADD SETMDZ	FAB8 SETMODE
?FD9D SETUP	FDA2 SETUP1	FE1A SHFTCNT	FAA4 SPCE
6A STACK	7C STATE	FD02 STOPLST	FAAF STOR1
FE28 STORCHRS	FE2C STOROW	FE2A STORSET	?FAAD STOR
F9D1 SVMASK	63 TBAS4H	62 TBAS4L	? 65 TBAS8H
64 TBAS8L	6C TEMPX	80 TEMP	6D TEMPY
F95E TOSUB	F9C9 TST8OWID	F99B TSTA1	FBD3 TSTBACK
FC36 TSTBELL	?FB97 TSTCR	FAF6 TSTDUMP	FE79 UNDER
Q3F8 USERADR	FA78 USER	FFB8 VBOUNDS	FEAE VRETRCE
FA4C VRFY2	FA3D VRFY	FA40 VRFY1	FE8F VWAIT
5B WINBTM	5A WINTOP	FAC3 WRTE	7D YSAV
A4 YTEMP	FDD5 ZIPTEMPS	F967 ZETATE	59 RMARGIN
50 INBUFLEN	5B SCRNL0C	5B LMARGIN	5D CV
5A WINTOP	5B WINBTM	5C CH	61 BAS8H
5E BAS4L	5F BAS4H	60 BAS8L	? 65 TBAS8H
62 TBAS4L	63 TBAS4H	64 TBAS8L	69 MASK
66 FORGND	67 BKOND	68 MODES	6C TEMPX
69 CURSOR	6A STACK	6B PROMPT	70 KSWL
6D TEMPY	6E CSWL	? 6F CSWH	74 A1L
? 71 KSWH	72 PCL	? 73 PCH	78 A3L
75 A1H	76 A2L	77 A2H	7C STATE
79 A3H	7A A4L	7B A4H	81 IBLSLOT
7D YSAV	7E INBUF	80 TEMP	A0 CPORTL
82 IDDRVN	85 IDBUFF	87 IBCMD	A3 CTEMP1
A0 ADR	A1 CPORTH	A2 CTEMP	CO00 KBD
A4 YTEMP	B4 ROWTEMP	Q3F8 USERADR	F479 BLOCKID
CO10 KBDSTRB	CODA CWRTOFF	CODB CWRTON	F882 RET3
F4EE DIAGN	F689 RECON	F7FF RET1	F908 MONZ
F900 RET2	?F901 ENTRY	F904 MON	



## 4,383,296

115

F912 SCAN  
 F932 NXTCHR  
 F959 NXTBS2  
 F96C CMDTAB  
 F99B TSTA1  
 F9B7 PRHEX2  
 F9C9 TSTBOWID  
 F9DE OLDPC  
 F9FB CRCHK  
 FFA0A CKMDE  
 FA21 REPEAT1  
 FA3B VRFY  
 FA61 PRINTA1  
 FA78 USER  
 FAB8 ERROR2  
 FAA0 BL1  
 FAB7 DUMMY  
 FAC3 WRTE  
 FAE9 DUMPS  
 FAFC DUMPO  
 FB21 DUMPASO  
 FB49 COL80  
 FB67 SETBOB  
 FB8E CLEOL  
 FBA4 CURUP  
 FBB3 BASCALC  
 FBCC7 SETCVH  
 FBDF LEFT80  
 FBF7F INCHORZ  
 FC24 CTRLRET  
 FC3A BELL  
 FC52 LNFD  
 FC7A SCRL3  
 FC9D DISPLAY  
 FCD0 CANCEL  
 FCE4 NXTCHAR  
 FDOC RDKEY  
 FE0E KEYIN3  
 FD47 KEYRET  
 FE58 ESC2  
 FE89 PICK  
 FDA2 SETUP1  
 FDE3 GASCI1  
 FE01 CBYTES  
 FE28 DONE  
 FE44 GENDONE  
 FE75 GEN2  
 FE99 NXTPORT  
 FEBF VWAIT  
 FFB0 NMIRG  
 FFFFA NMI

F915 NXTINP  
 F941 DIGIT  
 F95E TOSUB  
 F97C CMDVEC  
 F9AD RETA1  
 F9BF PRHEX2  
 F9D1 SVMASK  
 F9DF ASCII1  
 FA06 ASCDONE  
 FA11 BITON  
 FA26 CRMON  
 FA40 VRFY1  
 FA6E PRA1BYTE  
 FA7B JUMP  
 FASE ERROR  
 FAA4 SPCE  
 FAB8 SETMODE  
 FFA05 SAVCMD  
 FAF6 TSTDUMP  
 FB09 DUMP1  
 FB2C ASC1  
 FB4F COL40  
 FB69 CLSCRN  
 FB93 CONTROL  
 FBAE CURUP1  
 FBB7 CURIGHT  
 FBC9 CURDOWN  
 FBE9 LEFTUP  
 FC02 NXTLIN  
 FC25 COUT  
 FC3D BELL1  
 FC5B SCROLL  
 FC87 LASTLN  
 FCAA DSPBKOND  
 FCD5 GETLN  
 FCE7 CROUT  
 FDOF KEYIN  
 FFD31 KEYIN4  
 FD48 ESC3  
 FE60 RDCHAR  
 FE95 PICK40  
 FFD06 GENENTR  
 FDE5 GASCI2  
 FE05 CCOLMS  
 FE28 STORCHRS  
 FE48 GEN1  
 FE79 UNDER  
 FEAB NOHIGH  
 FEC5 CHRSET  
 FFEC CB2CTRL  
 FFFFC RESET

F91C CMDSRCH  
 F947 NXTBIT  
 F967 ZSTATE  
 F98C NXTA4  
 F9AC PRBYTE  
 F9C2 PRBYCOL  
 F9D4 A1PC  
 F9E1 ASCII2  
 FA07 ASCII  
 FA15 BITOFF  
 FA2C MOVE  
 FA4C VRFY2  
 FA70 PRBYTSP  
 FA7D GO  
 FA91 DEST  
 FFAAB STOR  
 FABD SETMDZ  
 FAC7 RWLOOP  
 FAF7 ERROR1  
 FB0C DUMP2  
 FB38 ASC2  
 FB53 SET80  
 FB71 CLEDP  
 FFB97 TSTCR  
 FBB1 SETCV  
 FBD0 RIGHT1  
 FBD5 TSTBACK  
 FBF2 COUT2  
 FC05 BASCALC1  
 FC33 COUT1  
 FC3E BELL2  
 FC61 SCRL1  
 FC89 CLEOL1  
 FCAD DSPL80  
 FCD5 GETLNZ  
 FDD2 STOPLST  
 FD16 KEYIN1  
 FD35 KEYWAIT  
 FD4B ESCAPE  
 FD77 QDESC  
 FFD98 CLDSTRT  
 FDD5 ZIPTMP5  
 FDE7 GASCI3  
 FE07 CSHFT  
 FE2A STORSET  
 FE63 NXTASCI  
 FEB8 ALTCHR  
 FEAD PORTDN  
 FFB4 HOOKS  
 FFED CB2INT  
 FFFFE IRQ

116

F92C GETNUM  
 F94F NXTBAS  
 F96B DIGRET  
 F992 NXTA1  
 F9B5 PRHEX  
 F9C5 PROCOLON  
 F9D7 A1PC1  
 F9F2 ASCII3  
 FA09 ASCII0  
 FA19 REPEAT  
 FA31 MOVNXT  
 FA52 MISMATCH  
 FA73 PRSPC  
 FAS3 RWERROR  
 FA9A SEP  
 FAAF STOR1  
 FAC0 READ  
 FADF NOVER  
 FFAF9 DUMP  
 FB1C DUMPS  
 FB46 ASC2  
 FB58 SET80A  
 FB7A CLEDP1  
 FFB9B CARRAGE  
 FBB3 CURDN1  
 FBC3 SETCH2  
 FBD9 CURLEFT  
 FBFC DISPLAYX  
 FC19 BSCL22  
 FC36 TSTBELL  
 FC43 BELL3  
 FC63 SCRL2  
 FC91 CLEOL2  
 FCBB NOTOP  
 FCD5 BKSPCE  
 FDD7 NOBTOP  
 FD24 KEYIN2  
 FD42 KWAIT2  
 FFD53 ESC1  
 FD7F ESCVCT  
 FFD90 SETUP  
 FDE1 GENASC  
 FDF4 GASCI4  
 FE1A SHFTCNT  
 FE2C STOROW  
 FE65 NXTASC2  
 FEB8 ALTCL  
 FEAE VRETRCE  
 FFB8 VBOUNDS  
 FFF0 ESCTABL

117

4,383,296

118

I claim:

1. In a digital computer which includes a central processing unit (CPU), a random-access memory (RAM), an address bus interconnecting said CPU and RAM such that said CPU addresses locations in said RAM and a data bus interconnecting said CPU and RAM, said CPU for certain functions addressing predetermined locations in said RAM with a predetermined range of address signals, an improvement comprising:

detection means for detecting said predetermined range of address signals, coupled to said address bus;

register means for storing digital signals, coupled to said data bus, and;

switching means for coupling said digital signals stored in said register means to said address bus when said detection means detects said predetermined range of said address signals;

whereby data for said certain functions normally stored by said CPU in said predetermined locations may be stored elsewhere in said RAM, thereby enhancing the performance of said computer.

2. The improvement defined by claim 2 wherein said detection means detects all binary zeros.

3. The improvement defined by claim 1 wherein said switching means comprises a multiplexer controlled by said detection means for selecting said register means.

4. The improvement defined by claim 1 including a read-only memory coupled to said address bus and said data bus.

5. The improvement defined by claim 4 wherein said stored signals in said register means provide a pointer for locations in said RAM during a direct memory access transfer.

6. The improvement defined by claim 5 wherein said read-only memory in response to signals on said address bus provides instructions to said CPU causing it to increment address signals during said direct memory access transfer.

7. In a digital computer which includes a central processing unit (CPU), a random-access memory (RAM), an address bus having a first plurality and a second plurality of lines for coupling said CPU with said RAM, and a data bus interconnecting said CPU and RAM, said CPU for certain operations addressing predetermined locations in said RAM with address signals on said first plurality of lines by coupling a predetermined address on said second plurality of lines, an improvement comprising:

register means for storing signals, coupled to said data bus;

multiplexing means coupled to said second plurality of lines and said register means for selecting signals from one of said second plurality of lines and said register means;

logic means coupled to said second plurality of lines and said multiplexing means for causing said multiplexing means to select signals from said register means when said CPU couples said predetermined address on said second plurality of lines;

whereby said signals from said register means provide alternate locations in RAM for storage associated with said certain operations.

8. The improvement defined by claim 7 wherein said predetermined address is all binary zeros.

9. The improvement defined by claim 7 including a read-only memory coupled to said address bus and said data bus.

10. The improvement defined by claim 8 wherein said stored signal in said register means provides a pointer for locations in said RAM during a direct memory access transfer.

11. The improvement defined by claim 9 wherein said read-only memory in response to signals on said address bus provides instructions to said CPU causing it to increment address signals during said direct memory access transfer.

12. In a digital processor used in conjunction with a display, said processor including a data bus and an address bus, a memory comprising:

a first plurality of memory devices for storing data, coupled to receive data from said data bus;

a first memory output bus coupled to receive data from said first plurality of memory device;

a second plurality of memory devices for storing data coupled to receive data from said data bus;

a second memory output bus coupled to receive data from said second plurality of memory devices;

addressing means coupled to said address bus for providing address signal for addressing said first and second plurality of memory devices;

first switching means for selecting data from one of said first and second memory buses for coupling to said data bus, said first switching means coupled to said first and second memory bus and said data bus;

second switching means for selecting data from said first and second memory buses for coupling to said display, said second switching means coupled to said first and second memory buses and said display; and,

circuit means for coupling one of a selected said first and second memory buses to said addressing means such that data from said selected one of said buses provides addressing information for selecting subsequent locations in said memory devices when said data bus is receiving data from the other of said memory buses,

whereby said memory provides data for a high resolution display and whereby some data stored in said memory is used for remapping locations in said memory.

13. The memory defined by claim 12 wherein said circuit means comprises a multiplexer, said multiplexer selecting between said data from said selected one of said buses and bank switching signals coupled to said multiplexer.

14. The memory defined by claim 13 wherein said multiplexer is controlled by a logic circuit which is coupled to said address bus and said selected one of said buses.

15. The memory defined by claim 14 wherein said logic circuit causes said multiplexer to select said bank switching signals each time said processor switches an OP code.

16. In a digital computer with a memory, which is used in conjunction with a raster scanned display, said display including a digital counter which provides a vertical count representative of the horizontal line scanned by the beam for said display, said memory providing data for displaying rows of characters, an

4,383,296

119

addressing means coupled to said memory for scrolling displayed characters, comprising:

an adder having a first and a second input terminal, the output of said adder providing a portion of an address signal for said memory, said first terminal of said adder being coupled to receive the lesser significant bits of said vertical count;

said computer providing a periodically repeated sequence of digital numbers coupled to said second terminal of said adder, said sequence of digital numbers provided by said computer having a maximum value equal to the number of scanned lines in each of said rows,

whereby the characters on said display are scrolled with a minimum of movement of data within said memory.

17. The addressing means defined by claim 16 wherein said sequence of digital numbers is incremented for each displayed frame.

18. In a digital computer which includes a single chip central processing unit (CPU), a random-access memory (RAM), an address bus interconnecting said CPU and RAM such that said CPU addresses locations in said RAM, and a data bus coupled to said CPU and RAM, said CPU for certain functions addressing the zero page in said RAM by providing binary zeroes on certain lines of said address bus; an improvement comprising:

a detection circuit for detecting said binary zeroes on said certain lines of said address bus;

a register for storing digital signals, said register coupled to said data bus for receiving digital signals from said data bus; and,

a multiplexer for selecting between said digital signals stored in said register and said certain lines of said address bus, said multiplexer being controlled by said detection circuit so as to select said register when said binary zeroes are detected on said certain lines of said address bus;

whereby data for said certain functions normally stored on page one of said RAM, may be stored elsewhere in said RAM, and still easily addressed by said CPU.

19. The improvement defined by claim 18 wherein one of said stored signals from said register is coupled to said multiplexer through an exclusive OR gate, said gate being coupled to one of said certain lines of said address bus.

20. The improvement defined by claim 18 or 19 wherein said computer provides an alternate stack sig-

120

nal and wherein said detection circuit also detects addresses for page one on said address bus, and said multiplexer selects said register if said page one addresses are detected and said alternate stack signal is in a predetermined state.

21. In a digital computer which includes a central processing unit (CPU), a random-access memory (RAM), an address bus interconnecting said CPU and RAM such that said CPU addresses locations in said RAM and a data bus interconnecting said CPU and RAM, said CPU for certain functions addressing predetermined locations in said RAM with a predetermined range of address signals, an improvement comprising:

detection means for detecting said predetermined range of address signals, coupled to said address bus;

register means for storing digital signals, coupled to said data bus; and;

switching means for coupling said digital signals stored in said register means to said address bus when said detection means detects said predetermined range of said address signals, said switching means also for coupling said digital signals stored in said register means to said address bus when a certain direct memory access (DMA) signal is in a predetermined state;

a read-only memory (ROM) coupled between said address bus and said data bus, said ROM in response to signals on said address bus providing instructions to said CPU on said data bus to cause said CPU to increment address signals when said DMA signal is in said predetermined state;

said register providing a pointer for locations in said RAM when said DMA signal is in said predetermined state, and said register providing RAM address signals when said certain functions are selected by said CPU,

whereby data for said certain functions normally stored by said CPU in said predetermined locations may be stored elsewhere in said RAM, thereby enhancing the performance of said computer.

22. The improvement defined by claim 21 wherein said switching means comprise a multiplexer which selects said register when said detection means detects all binary zeroes or when said DMA signal is in said predetermined state.

\* \* \* \* \*

FINIS

Apple III computer system diagram

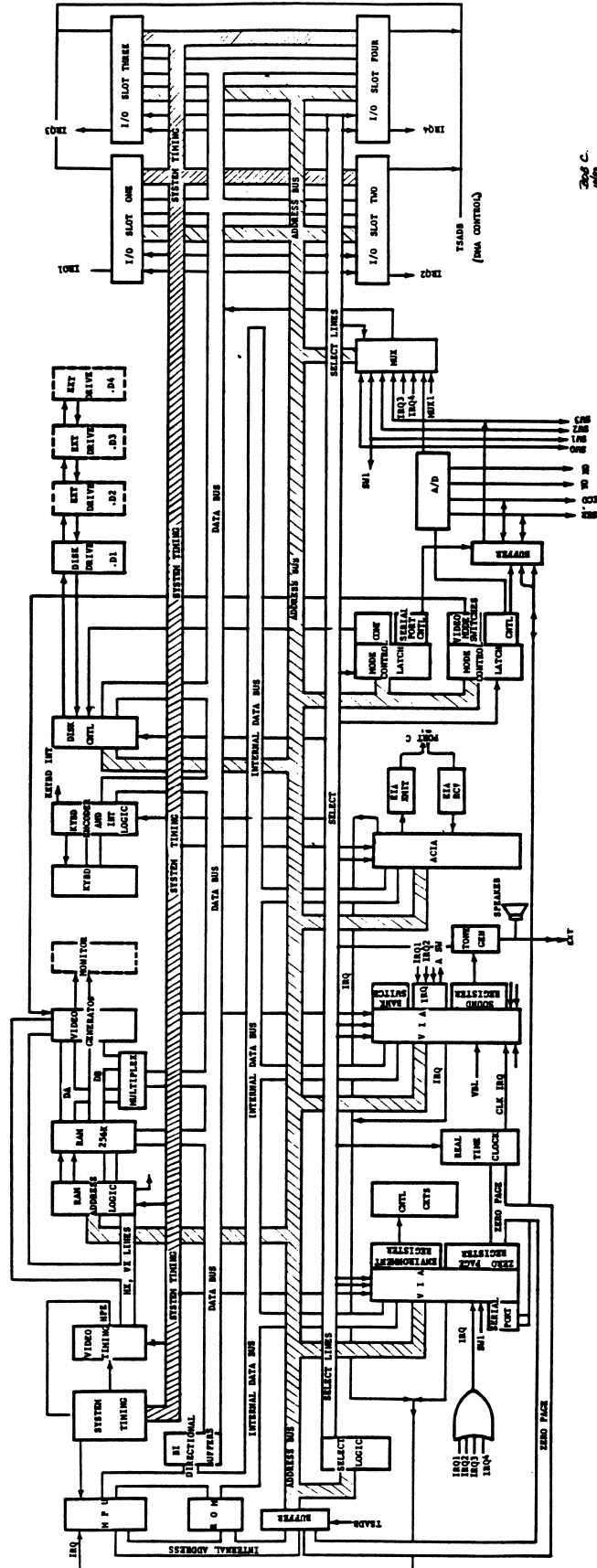
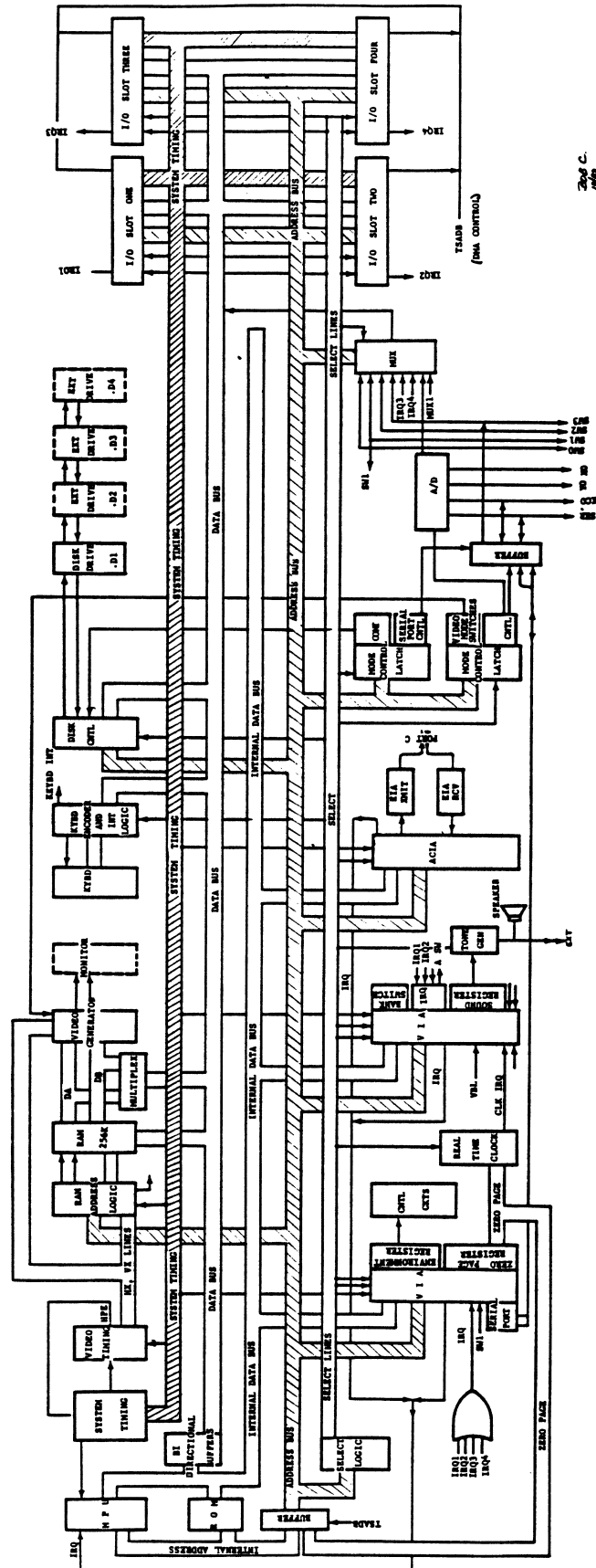


Fig. 1

Fig. 2

# Apple III computer system diagram



*Apple /// ROM Information*

---

# APPLE /// ROM INFORMATION

---

*ROM  
REVISION  
#0*

by  
**David Craig**  
*736 Edgewater, Wichita, Kansas 67230*  
**1986**

This document describes the Apple /// microcomputer ROM organization. The ROM listing used was from Apple Computer's patent (# 4,383,296) of May 10, 1983 as assigned to Wendell B. Sander. The ROM listing appears to be from December 20, 1979.

The ROM occupies 4K bytes of memory in the address range \$F000—\$FFFF. This ROM is used by the Apple /// at system power-up to test various hardware components, initialize the character generator bitmap, and boot SOS (Sophisticated Operating System) from the Apple ///'s internal floppy diskette drive.

The ROM is organized as follows (routine names in lowercase were created by me since the source code did not contain a name at the particular location):

Addresses	Name	Description
F000-F124	REGWTS	Read/Write a disk track and sector
F125-F12A	SETTRK	Set slot dependent track location
F12B-F13D	CHKDRV	Check if disk motor is stopped
F13E-F147	DRVINDX	Get index to drive number
F148-F1B9	READ16	Read disk sector
F1BA-F1BC	GOSERV	Interrupt service vector
F1BD-F218	ROADR16	Read disk sector address field
F219-F2B2	WRITE16	Write disk sector
F2B3-F2BB	SERVICE	Interrupt servicer
F2BC-F2C5	WNIBL9	Write 7-bit nibbles to disk
F2C6-F310	PRENIB16	Pre-nibblize disk sector data
F311-F354	POSTNIB16	Post-nibblize disk sector data
F355-F395	NIBL	6-bit to 7-bit nibble conversion table
F396-F3FF	DNIBL	7-bit to 6-bit denibbleize conversion table
F400-F455	SEEK	Disk track seeker
F456-F466	MSWAIT	100 microsecond delayer
F467-F46F	ONTABLE	Disk phase ON time table (in 100 microsecs)
F470-F478	OFFTABLE	Disk phase OFF time table (in 100 microsecs)
F479-F49F	BLOCKIO	Read/write a disk block (2 sectors)

*Apple /// ROM Information*

F4A0-F4A7	SECTABL	Block to sector conversion table
F4A8-F4C4	ANALOG	Joystick read routine
F4C5-F4CC	RAMTBL	RAM test bytes
F4CD-F4ED	CHPG	Hardware component phrases (eg "RAM", "ROM",...)
F4EE-F523	DIAGN	ROM system power-up entry (calls RECON [F689])
F524-F531	NXBYT	Test RAM page 0 (Zero Page)
F532-F545	CNTWR	Test RAM page 1 (Stack Page)
F546-F574	memsize	Size the RAM
F575-F589	ERRLP	Display screen error line ("DIAGNOSTICS")
F58A-F5E6	zpgstkst	Test RAM zero page & stack page
F5E7-F60C	ROMTST	Test ROM hardware
F60D-F63D	VIATST	Test VIA hardware
F63E-F652	ACIA	Test ACIA hardware
F653-F67A	ATD	Test A/D hardware
F67B-F688	KEYPLUG	Test keyboard plugin
F689-F6C1	RECON	Reconfigure system (tests for Apple-1 key)
F6C2-F6E5	SEX	System exerciser
F5E6-F737	USRENTY	Main RAM tester
F738-F747	STRWT	Error message string writer
F748-F77A	RAM	Determine size of RAM
F77B-F783	MESSERR	Display error message
F784-F7A0	RAMSET	Setup RAM
F7A1-F7C8	PTRINC	Increment extended addressing pointer
F7C9-F7F6	RAMERR	RAM error handler
F7F7-F7FF	RAMWT	RAM write
F800-F900	RET1	Nested RTS 'table' routine
F901-F92B	ENTRY	SARA Monitor entry point
F92C-F95D	GETNUM	Get number from user
F92E-F96B	TOSUB	Execute Monitor command
F96C-F97B	CMDTAB	Monitor command code table
F97C-F98B	CMDVEC	Monitor command vector table (byte-long entries)
F98C-F9AB	NXTA4	Increment 2 byte pointer
F9AC-F9C1	PRBYTE	Output a byte to screen
F9C2-F9C8	PRBYCOL	Output a byte followed by a colon
F9C9-F9D3	TST80WID	Test for 80-column screen width
F9D4-F9DE	A1PC	Test for new P.C.
F9DF-FA06	ASCII1	Store user ASCII string into memory
FA07-FA25	ASCII	Fetch ASCII character from keyboard
FA26-FA2B	CRMON	Dump line of hexadecimal bytes due to user CR
FA2C-FA3A	MOVE	Move bytes around in memory
FA3B-FA51	VRFY	Verify memory byte range
FA52-FA77	MISMATCH	Output verify mismatch data line
FA78-FA7A	USER	User control vector
FA7B-FA82	JUMP	Transfer control to user routine
FA83-FA90	RWERROR	Output error number
FA91-FA99	DEST	Copy source pointer to destination pointer
FA9A-FAB7	SEP	Test for separator character in input line
FAB8-FABF	SETMODE	Setup user mode
FAC0-FAE8	READ	Handle Monitor READ disk block command
FAE9-FB20	DUMPB	Output line of memory bytes
FB21-FB48	DUMPASC	Output line of memory bytes as ASCII
FB49-FB4E	COL80	Setup 80-column display mode
FB4F-FB92	COL40	Setup 40-column display mode

ENTRY  
POINT

### *Apple /// ROM Information*

FB93-FBA3	CONTROL	Handle user control character input
FBA4-FBB6	CURUP	Handle cursor up motion
FBB7-FBC8	CURIGHT	Handle cursor right motion
FBC9-FBD4	DURDOWN	Handle cursor down motion
FBD5-FBD8	LSTBACK	Handle backspace motion
FBD9-FBF1	CURLEFT	Handle cursor left motion
FBF2-FC04	COUT2	Output character to screen
FC05-FC24	BASCALC1	Compute character base address for screen output
FC25-FC32	COUT	Output character to current output device
FC33-FC35	COUT1	Character output vector
FC36-FC51	TSTBELL	Handle BELL character output (beep speaker)
FC52-FC5A	LNFD	Handle LINE FEED character output
FC5B-FC9C	SCROLL	Scroll screen lines
FC9D-FCAC	DISPLAY	Display character on 40-column screen
FCAD-FCBA	DSPL80	Display character on 80-column screen
FCBB-FCD4	NOTCR	Handle non-control character output
FCD5-FD0B	GETLNZ	Read user ASCII line from keyboard
FDOC-FD0E	RDKEY	Read keyboard key input vector
FD0F-FD47	KEYIN	Read raw keyboard key
FD48-FD5F	ESC3	Handle ESC character cursor motion
FD60-FD76	RDCHAR	Read keyboard character
FD77-FD7E	GOESC	ESC key cursor motion handler
FD7F-FD87	ESCVECT	ESC key editing command key code table
FD88-FD97	PICK	Read character from current cursor location
FD98-FDC5	CLDSTART	Cold boot system (initialize ROM globals)
FDC6-FEAD	GENENTR	Load character generator RAM with bitmap
FEAE-FEC4	VRETRCE	Wait/poll for CRT vertical retrace
FEC5-FFB3	CHRSET	Character generator character bitmap table
FFB4-FFB7	HOOKS	Output/Input vectors
FFB8-FFBB	VBOUNDS	Screen dimension bounds (0,80,0,24)
FFBC-FFBF	NMIIRQ	NMI request vector (JMP RECON [F689] RTI)
FFC0-FFEF	applecwrite	Apple Computer, Inc. 1980 copyright phrase
FFF0-FFF9	ESCTABL	ESC character table
FFFA-FFFB	NMI	NMI vector [FFCA]
FFFC-FFFF	RESET	RESET vector [F4EE] (Power-up Diagnostics)
FFFE-FFFF	IRG	IRG vector [FFCD]

---- *The End* ----



Assembler: Apple III Pascal TLA 6502 Assembler  
(converted to TLA format most likely by Scott Stinson)

41 pages

Source Code Listing  
for

Apple III

Sara ROM

\$F000 - \$FFFF 4KB

REVISION 1

(see AIII patent for Rev 0 ROM)

David T. Craig  
736 Edgewater  
Wichita, Kansas 67230

Copyrighted Jan. 1980

Source Code Listing  
for

**Apple ///**

**ROM - Disk I/O**

David T. Craig  
736 Edgewater  
Wichita, Kansas 67230

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 1

```

0000| :*****
0000| : APPLE /// ROM - DISK I/O ROUTINES
0000| : COPYRIGHT 1979 BY APPLE COMPUTER, INC.
0000| :*****
0000|
0000| .ABSOLUTE
0000| .PROC DISKIO
0000| .ORG 0F000
0000|
F000| :*****
F000| ; CRITICAL TIMING *
F000| ; REQUIRES PAGE BOUND *
F000| ; CONSIDERATIONS FOR *
F000| ; CODE AND DATA *
F000| ; -----CODE----- *
F000| ; VIRTUALLY THE ENTIRE *
F000| ; 'WRITE' ROUTINE *
F000| ; MUST NOT CROSS *
F000| ; PAGE BOUNDARIES *
F000| ; CRITICAL BRANCHES IN *
F000| ; THE 'WRITE', 'READ', *
F000| ; AND 'READ ADR' SUBRS *
F000| ; WHICH MUST NOT CROSS *
F000| ; PAGE BOUNDARIES ARE *
F000| ; NOTED IN COMMENTS *
F000| ; *****
F000| ;
F000| ; EQUATES *
F000| ;
F000| 0200 NBUF1 .EQU 0200
F000| 0302 NBUF2 .EQU 0302 ; (ZERO PAGE AT $300)
F000|
F000| 0080 HRDERRS .EQU 80
F000| 00E0 DVMOT .EQU 0E0
F000|
F000| 0081 IBSLOT .EQU 81
F000| 0082 IBDRVN .EQU IBSLOT+1
F000| 0083 IBTRK .EQU IBSLOT+2
F000| 0084 IBSECT .EQU IBSLOT+3
F000| 0085 IBBUFP .EQU IBSLOT+4 ; & 5
F000| 0087 IBCMD .EQU IBSLOT+6
F000| 0088 IBSTAT .EQU IBSLOT+7
F000| 0089 IBSMOD .EQU IBSLOT+8
F000| 0089 CSUM .EQU IBSMOD ; USED ALSO FOR ADDRESS HEADER CKSUM
F000| 008A IOBPDN .EQU IBSLOT+9
F000| 008B IMASK .EQU IBSLOT+0A
F000| 008C CURTRK .EQU IBSLOT+0B
F000| 0085 DRVOTRK .EQU CURTRK-7
F000| ; SLOT 4, DRIVE 1
F000| ; SLOT 4, DRIVE 2
F000| ; SLOT 5, DRIVE 1
F000| ; SLOT 5, DRIVE 2
F000| ; SLOT 6, DRIVE 1
F000| ; SLOT 6, DRIVE 2
F000| 0093 RETRYCNT .EQU IBSLOT+12
F000| 0094 SEEKCNT .EQU IBSLOT+13
F000| 009B BUF .EQU IBSLOT+1A
F000| 009F ENVTEMP .EQU IBSLOT+1E
F000| ; IBSLOT+$1F NOT USED
F000| ;
F000| ; *****
F000| ;
F000| ; ----READADR---- *
F000| ;
F000| ; *****
F000| ;
F000| 0095 COUNT .EQU IBSLOT+14 ; 'MUST FIND' COUNT.
F000| 0095 LAST .EQU IBSLOT+14 ; 'ODD BIT' NIBLS.
F000| 0096 CKSUM .EQU IBSLOT+15 ; CHECKSUM BYTE.
F000| 0097 CSSTV .EQU IBSLOT+16 ; FOUR BYTES
F000| ; CHECKSUM, SECTOR, TRACK, AND VOLUME.
F000| ;
F000| ; *****
F000| ;
F000| ; ----WRITE---- *
F000| ;
F000| ; USES ALL NBUFS *
F000| ; AND 32-BYTE *
F000| ; DATA TABLE 'NIBL' *
F000| ;
F000| ; *****
F000| ;
F000| ; *****
F000| ;
F000| ; ----READ---- *
F000| ;
F000| ; USES ALL NBUFS *
F000| ; USES LAST 54 BYTES *
F000| ; OF A CODE PAGE FOR *

```

## HD:Apple ///:ROM - Disk I/O

Page 2

```
F000|          ; SIGNIFICANT BYTES      *
F000|          ; OF DNIBL TABLE.      *
F000|          ;                          *
F000|          ; *****                *
F000|          ; *****                *
F000|          ;                          *
F000|          ; -----SEEK-----    *
F000|          ; *****                *
F000|          ; *****                *
F000| 0095     TRKCNT   .EQU    COUNT        ; HALFTRACKS MOVED COUNT.
F000| 009D     PRIOR    .EQU    IBSLOT+1C
F000| 009E     TRKN     .EQU    IBSLOT+1D
F000|          ; *****                *
F000|          ; -----MSWAIT-----    *
F000|          ; *****                *
F000|          ; *****                *
F000| 0099     MONTIMEL .EQU    CSSTV+2       ; MOTOR-ON TIME
F000| 009A     MONTIMEH .EQU    MONTIMEL+1    ; COUNTERS.
F000|          ; *****                *
F000|          ; DEVICE ADDRESS           *
F000|          ; ASSIGNMENTS              *
F000|          ; *****                *
F000|          ; *****                *
F000| C080     PHASEOFF  .EQU    0C080        ; STEPPER PHASE OFF.
F000| C081     PHASEON   .EQU    0C081        ; STEPPER PHASE ON.
F000| C08C     Q6L       .EQU    0C08C        ; Q7L,Q6L=READ
F000| C08D     Q6H       .EQU    0C08D        ; Q7L,Q6H=SENSE WPROT
F000| C08E     Q7L       .EQU    0C08E        ; Q7H,Q6L=WRITE
F000| C08F     Q7H       .EQU    0C08F        ; Q7H,Q6H=WRITE STORE
F000| FFEE     INTERRUPT .EQU    0FFEF
F000| FFDF     ENVIRON   .EQU    0FFDF
F000| 0080     ONEMEG    .EQU    80
F000| 007F     TWOMEG    .EQU    7F
F000|          ; *****                *
F000|          ; *****                *
F000|          ; EQUATES FOR RWTS AND BLOCK
F000|          ; *****                *
F000|          ; *****                *
F000| C088     MOTOROFF  .EQU    0C088
F000| C089     MOTORON   .EQU    0C089
F000| C08A     DRVOEN    .EQU    0C08A
F000| C08B     DRVLEN    .EQU    0C08B
F000| C081     PHASON    .EQU    0C081
F000| C080     PHSOFF    .EQU    0C080
F000| 0097     TEMP      .EQU    CSSTV        ; PUT ADDRESS INFO HERE
F000| 0097     CSUM1     .EQU    TEMP
F000| 0098     SECT      .EQU    CSUM1+1
F000| 0099     TRACK     .EQU    SECT+1
F000| 0099     TRKN1     .EQU    TRACK
F000| 009A     VOLUME    .EQU    TRACK+1
F000| 0083     IBRERR     .EQU    HRDERRS+3
F000| 0082     IBDERR     .EQU    HRDERRS+2
F000| 0081     IBWPER     .EQU    HRDERRS+1
F000| 0080     IBNODRV    .EQU    HRDERRS
F000|          ; *****                *
F000|          ; *****                *
F000|          ; READ WRITE A            *
F000|          ; TRACK AND SECTOR         *
F000|          ; *****                *
F000|          ; *****                *
F000| A0 01     REGRWTS   LDY      #01        ; RETRY COUNT
F002| A6 81     LDX      IBSLOT        ; GET SLOT # FOR THIS OPERATION
F004| 84 94     STY      SEEKCNT      ; ONLY ONE RECALIBRATE PER CALL
F006| A9 05     LDA      #005
F008| 85 8F     STA      08F
F00A| 08        PHP
F00B| 68        PLA
F00C| 6A        ROR      A
F00D| 6A        ROR      A
F00E| 6A        ROR      A
F00F| 6A        ROR      A
F010| 85 8B     STA      IMASK
F012| AD DFFF    LDA      ENVIRON      ; PRESERVE ENVIRONMENT
F015| 85 9F     STA      ENVTEMP
F017| 20 2BF1    JSR      CHKDRV        ; SET ZERO FLAG IF MOTOR STOPPED
F01A| 08        PHP
F01B| A5 85     LDA      IBUFFP        ; MOVE OUT POINTER TO BUFFER INTO ZPAGE
F01D| 85 9B     STA      BUF
```

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 3

```

F01F| A5 86          LDA    IBBUFP+1
F021| 85 9C          STA    BUF+1
F023| A9 E0          LDA    #DVMOT
F025| 85 9A          STA    MONTIMEH
F027| A5 82          LDA    IBDRVN      ; DETERMINE DRIVE ONE OR TWO
F029| C5 8A          CMP    IOBPDN      ; SAME DRIVE USED BEFORE
F02B| 85 8A          STA    IOBPDN      ; SAVE IT FOR NEXT TIME
F02D| 08            PHP            ; KEEP RESULTS OF COMPARE
F02E| 6A            ROR    A            ; GET DRIVE NUMBER INTO CARRY
F02F| BD 89C0        LDA    MOTORON,X  ; TURN ON THE DRIVE
F032| 9001          BCC    DRVSEL      ; BRANCH IF DRIVE 1 SELECTED
F034| E8            INX            ; SELECT DRIVE 2
F035| BD 8AC0        LDA    DRVSEL    LDA    DRVOEN,X
F038| 20 4CF3        JSR    SETIMEG   ; INSURE ONE MEGAHERTZ OPERATION
F03B| 28            PLP            ; WAS IT SAME DRIVE?
F03C| F00A          BEQ    OK
F03E| 28            PLP            ; MUST INDICATE DRIVE OFF BY SETTING ZERO FLAG
F03F| A0 07          LDY    #07        ; DELAY 150 MS BEFORE STEPPING
F041| 20 56F4        JSR    MSWAIT    ; (ON RETURN A=0)
F044| 88            DEY
F045| D0FA          BNE    DRVWAIT
F047| 08            PHP            ; NOW ZERO FLAG SET
F048| A5 83          LDA    IBTRK     ; GET DESTINATION TRACK
F04A| A6 81          LDX    IBSLOT     ; RESTORE PROPER X (SLOT*16)
F04C| 20 04F1        JSR    MYSEEK    ; AND GO TO IT
F04F| 28            PLP            ; NOW AT THE DESIRED TRACK WAS THE MOTOR ON TO START WITH?
F050| D017          BNE    TRYTRK     ; WAS MOTOR ON?
F052|               ; IF SO, DON'T DELAY, GET IT TODAY!
F052|               ;
F052|               ; MOTOR WAS OFF, WAIT FOR IT TO SPEED UP
F052|               ;
F052| A0 12          LDY    #12        ; WAIT EXACTLY 100 US FOR EACH COUNT
F054| 88            CONWAIT          ; IN MONTIME
F055| D0FD          BNE    CONWAIT
F057| E6 99          INC    MONTIMEH  ; COUNT UP TO 0000
F059| D0F7          BNE    MOTOF
F05B| E6 9A          INC    MONTIMEH
F05D| 30F3          BMI    MOTOF
F05F|               ;
F05F|               ; *****
F05F|               ; MOTOR SHOULD BE UP TO SPEED
F05F|               ; IF IT STILL LOOKS STOPPED THEN
F05F|               ; THE DRIVE IS NOT PRESENT.
F05F|               ; *****
F05F|               ;
F05F| 20 2BF1        JSR    CHKDRV      ; IS DRIVE PRESENT?
F062| D005          BNE    TRYTRK     ; YES, CONTINUE
F064| A9 80          NODRIVERR      LDA    #IBNODRV ; NO, GET TELL EM NO DRIVE
F066| 4C EAF0        JMP    HNDLERR
F069|               ;
F069|               ; NOW CHECK IF IT IS NOT THE FORMAT DISK COMMAND,
F069|               ; LOCATE THE CORRECT SECTOR FOR THIS OPERATION
F069|               ;
F069| A5 87          TRYTRK    LDA    IBCMD      ; GET COMMAND CODE #
F06B| F076          BEQ    ALLDONE    ; IF NULL COMMAND, GO HOME TO BED
F06D| C9 03          CMP    #03      ; COMMAND IN RANGE?
F06F| B072          BCS    ALLDONE    ; NO, DO NOTHING!
F071| 6A            ROR    A            ; SET CARRY=1 FOR READ, 0 FOR WRITE
F072| B00B          BCS    TRYTRK2    ; MUST PRENIBBLIZE FOR WRITE
F074| AD DFFF        LDA    ENVIRON
F077| 29 7F          AND    #TWOMEG   ; SHIFT TO HIGH SPEED!
F079| 8D DFFF        STA    ENVIRON
F07C| 20 C4F2        JSR    PRENIB16
F07F| A0 7F          TRYTRK2    LDY    #7F      ; ONLY 127 RETRIES OF ANY KIND
F081| 84 93          STY    RETRYCNT
F083| A6 81          TRYADR    LDX    IBSLOT    ; GET SLOT NUM INTO X-REG
F085| 20 B9F1        JSR    RDADR16    ; READ NEXT ADDRESS FIELD
F088| 9022          BCC    RDRIGHT    ; IF READ IS RIGHT, HURRAH!
F08A| 20 AAF1        TRYADR2    JSR    CHKINT  ; BRANCH TO CHECK FOR INTERRUPTS
F08D| C6 93          DEC    RETRYCNT  ; ANOTHER MISTAKE!!
F08F| 10F2          BPL    TRYADR     ; WELL, LET IT GO THIS TIME
F091| C6 94          DEC    SEEKCNT    ; ONLY RECALIBRATE ONCE!
F093| D053          BNE    DRVERR     ; TRIED TO RECALIBRATE A SECOND TIME, ERROR!
F095| A5 8F          LDA    08F        ; ANOTHER MISTAKE!!
F097| 30EA          BMI    TRYADR     ; WELL, LET IT GO THIS TIME
F099| A5 8C          LDA    CURTRK
F09B| 48            PHA            ; SAVE TRACK WE REALLY WANT
F09C| A9 60          LDA    #60        ; RECALIBRATE ALL OVER AGAIN! ERROR!
F09E| 20 25F1        JSR    SETTRK    ; PRETEND TO BE ON TRACK 80
F0A1| A9 00          LDA    #00
F0A3| 20 04F1        JSR    MYSEEK    ; MOVE TO TRACK 00
F0A6| 68            GOCAL1    PLA
F0A7| 20 04F1        GOCAL    JSR    MYSEEK    ; GO TO CORRECT TRACK THIS TIME!
F0AA| 90D7          BCC    TRYADR     ; LOOP BACK, TRY AGAIN ON THIS TRACK
F0AC|               ;
F0AC|               ; HAVE NOW READ AN ADDRESS FIELD CORRECTLY.
F0AC|               ; MAKE SURE THIS IS THE TRACK, SECTOR, AND VOLUME DESIRED.
F0AC|               ;
F0AC|               ;
F0AC| A4 99          RDRIGHT    LDY    TRACK    ; ON THE RIGHT TRACK?

```

CMD

1 → Read  
2 → Write

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 4

```

F0AE| C4 8C          CPY      CURTRK
F0B0| F00E          BEQ      RTRK      ; IF SO, GOOD
F0B2|              ;
F0B2|              ; RECALIBRATING FROM THIS TRACK
F0B2|              ;
F0B2| A5 8C          LDA      CURTRK      ; PRESERVE DESTINATION TRACK
F0B4| 48            PHA
F0B5| 98            TYA
F0B6| 0A          ASL      A
F0B7| 20 25F1      JSR      SETTRK
F0BA| 68          PLA
F0BB| 20 04F1      JSR      MYSEEK
F0BE| 90CA        BCC      TRYADR2
F0C0| A5 9A        LDA      VOLUME      ; GET ACTUAL VOLUME HERE
F0C2| 85 89        STA      IBMOD      ; TELL OPSYS WHAT VOLUME WAS THERE
F0C4| A5 98        LDA      SECT      ; CHECK IF THIS IS THE RIGHT SECTOR
F0C6| C5 84        CMP      IBSECT
F0C8| D0C0        BNE      TRYADR2      ; NO, TRY ANOTHER SECTOR
F0CA| A5 87        LDA      IBCMD      ; READ OR WRITE?
F0CC| 4A          LSR      A           ; THE CARRY WILL TELL
F0CD| 902A        BCC      WRIT      ; CARRY WAS SET FOR READ OPERATION,
F0CF| 20 48F1      JSR      READ16     ; CLEARED FOR WRITE
F0D2| B0B6        BCS      TRYADR2     ; CARRY SET UPON RETURN IF BAD READ
F0D4| AD DFFF      LDA      ENVIRON
F0D7| 29 7F        AND      #TWOMEG
F0D9| 8D DFFF      STA      ENVIRON      ; SET TWO MEGAHERTZ
F0DC| 20 0FF3      JSR      POSTNIB16 ; DO PARTIAL POSTNIBBLE CONVERSION
F0DF| A6 81        LD      IBSLOT      ; RESTORE SLOTNUM INTO X
F0E1| B0A7        BCS      TRYADR2     ; CHECKSUM ERROR
F0E3| 18          CLC
F0E4| A9 00        LDA      #00        ; NO ERROR
F0E6| 9003        BCC      ALDONE1     ; SKIP OVER NEXT BYTE WITH BIT OPCODE
F0E8| A9 82        LDA      #IBDERR    ; BAD DRIVE
F0EA| 38          SEC      ; INDICATE AN ERROR
F0EB| 85 88        STA      IBSTAT     ; GIVE HIM ERROR
F0ED| BD 88C0      LDA      MOTOROFF,X ; TURN IT OFF
F0F0| 20 AAF1      JSR      CHKINT     ; BRANCH TO CHECK FOR INTERRUPTS
F0F3| A5 9F        LDA      ENVTEMP    ; RESTORE ORIGINAL ENVIRONMENT
F0F5| 8D DFFF      STA      ENVIRON
F0F8| 60          RTS
F0F9|
F0F9| 20 16F2      WRIT      JSR      WRITE16 ; WRITE NYBBLES NOW
F0FC| 90E5        BCC      ALDONE5     ; IF NO ERRORS
F0FE| A9 81        LDA      #IBWPER    ; DISK IS WRITE PROTECTED!!
F100| 50E8        BVC      HNDLERR     ; TAKEN IF TRUELY WRITE PROTECT ERROR
F102| D086        BNE      TRYADR2     ; OTHERWISE ASSUME AN INTERRUPT MESSED THINGS UP
F104|
F104|              ; THIS IS THE 'SEEK' ROUTINE
F104|              ; SEEKS TRACK 'N' IN SLOT #X/$10
F104|              ; IF DRIVEN0 IS NEGATIVE, ON DRIVE 0
F104|              ; IF DRIVEN0 IS POSITIVE, ON DRIVE 1
F104|              ;
F104| 0A          MYSEEK  ASL      A           ; ASSUME TWO PHASE STEPPER.
F105| 85 99        SEEK1  STA      TRKN1      ; SAVE DESTINATION TRACK(*2)
F107| 20 18F1      JSR      ALLOFF      ; TURN ALL PHASES OFF TO BE SURE.
F10A| 20 3EF1      JSR      DRVINDX     ; GET INDEX TO PREVIOUS TRACK FOR CURRENT DRIVE
F10D| B5 85        LDA      DRVOTRK,X
F10F| 85 8C        STA      CURTRK      ; THIS IS WHERE I AM
F111| A5 99        LDA      TRKN1      ; AND WHERE I'M GOING TO
F113| 95 85        STA      DRVOTRK,X
F115| 20 00F4      GOSEEK JSR      SEEK      ; GO THERE!
F118| A0 03        ALLOFF LDY      #03     ; TURN OFF ALL PHASES BEFORE RETURNING
F11A| 98          NXOFF  TYA           ; (SEND PHASE IN ACC.)
F11B| 20 4AF4      JSR      CLRPHASE    ; CARRY IS CLEAR, PHASES SHOULD BE TURNED OFF
F11E| 88          DEY
F11F| 10F9        BPL      NXOFF
F121| 46 8C        LSR      CURTRK      ; DIVIDE BACK NOW
F123| 18          CLC
F124| 60          RTS
F125|
F125|              ; THIS SUBROUTINE SETS THE SLOT DEPENDENT TRACK
F125|              ; LOCATION
F125|              ;
F125| 20 3EF1      SETTRK  JSR      DRVINDX     ; GET INDEX TO DRIVE NUMBER
F128| 95 85        STA      DRVOTRK,X
F12A| 60          RTS
F12B|
F12B| *****
F12B|
F12B|              ; SUBR TO TELL IF MOTOR IS STOPPED
F12B|
F12B|              ; IF MOTOR IS STOPPED, CONTROLLER'S
F12B|              ; SHIFT REG WILL NOT BE CHANGING.
F12B|
F12B|              ; RETURN Y=0 AND ZERO FLAG SET IF IT IS STOPPED.
F12B|
F12B| *****
F12B|
F12B|
F12B| A0 00        CHKDRV  LDY      #00     ; INIT LOOP COUNTER
F12D| BD 8CC0      CHKDRV1 LDA      Q6L,X   ; READ THE SHIFT REG

```

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 5

```

F130| 20 3DF1      JSR    CKDRTS      ; DELAY
F133| 48          PHA
F134| 68          PLA
F135| DD 8CC0      CMP    Q6L,X      ; HAS SHIFT REG CHANGED?
F138| D003         BNE    CKDRTS      ; YES, MOTOR IS MOVING
F13A| 88          DEY
F13B| D0F0         BNE    CHKDRV1     ; NO, DEC RETRY COUNTER
F13D| 60          RTS                ; AND TRY 256 TIMES
F13E|             ;                  ; THEN RETURN
F13E| 48          DRVINDX PHA          ; PRESERVE ACC.
F13F| 8A          TXA                ; GET SLOT(*$10)/8
F140| 4A          LSR    A
F141| 4A          LSR    A
F142| 4A          LSR    A
F143| 05 82       ORA    IBDRVN      ; FOR DRIVE 0 OR 1
F145| AA          TAX                ; INTO X FOR INDEX TO TABLE
F146| 68          PLA                ; RESTORE ACC.
F147| 60          RTS
F148|             ;
F148|             ; *****
F148|             ; NOTE: FORMATTING ROUTINES
F148|             ; NOTE INCLUDED FOR SOS
F148|             ; *****
F148|             ; *****
F148|             ; READ SUBROUTINE
F148|             ; (16-SECTOR FORMAT)
F148|             ; *****
F148|             ; READS ENCODED BYTES
F148|             ; INTO NBUF1 AND NBUF2
F148|             ; *****
F148|             ; FIRST READS NBUF2
F148|             ; HIGH TO LOW,
F148|             ; THEN READS NBUF1
F148|             ; LOW TO HIGH.
F148|             ; *****
F148|             ; ---- ON ENTRY ----
F148|             ; *****
F148|             ; X-REG: SLOTNUM
F148|             ; TIMES $10.
F148|             ; *****
F148|             ; READ MODE (Q6L, Q7L)
F148|             ; *****
F148|             ; ---- ON EXIT ----
F148|             ; *****
F148|             ; CARRY SET IF ERROR
F148|             ; *****
F148|             ; IF NO ERROR:
F148|             ; A-REG HOLDS $AA.
F148|             ; X-REG UNCHANGED.
F148|             ; Y-REG HOLDS $00.
F148|             ; CARRY CLEAR.
F148|             ; ---- CAUTION ----
F148|             ; *****
F148|             ; OBSERVE
F148|             ; 'NO PAGE CROSS'
F148|             ; WARNINGS ON
F148|             ; SOME BRANCHES!!
F148|             ; *****
F148|             ; ---- ASSUMES ----
F148|             ; *****
F148|             ; 1 USEC CYCLE TIME
F148|             ; *****
F148|             ; *****
F148| A0 20          READ16 LDY    #20      ; 'MUST FIND' COUNT.
F14A| 88          RSYNC  DEY            ; IF CAN'T FIND MARKS.
F14B| F06A         BEQ    RDERR        ; THEN EXIT WITH CARRY SET
F14D| BD 8CC0      LDA    Q6L,X      ; READ NIBL.
F150| 10FB         BPL    RD1         ; *** NO PAGE CROSS! ***
F152| 49 D5        EOR    #0D5       ; DATA MARK1?
F154| D0F4         BNE    RSYNC      ; LOOP IF NOT.
F156| EA          NOP                ; DELAY BETWEEN NIBLS.
F157| BD 8CC0      RD2    LDA    Q6L,X
F15A| 10FB         BPL    RD2         ; *** NO PAGE CROSS! ***
F15C| C9 AA        CMP    #0AA        ; DATA MARK 2?
F15E| D0F2         BNE    RSYNC1     ; (IF NOT, IS IT DM1?)
F160| A0 55        LDY    #055       ; INIT NBUF2 INDEX.
F162|             ; (ADDED NIBL DELAY)
F162| EA          NOP                ; DELAY BETWEEN NIBLS.
F163| BD 8CC0      RD3    LDA    Q6L,X
F166| 10FB         BPL    RD3         ; *** NO PAGE CROSS! ***
F168| C9 AD        CMP    #0AD        ; DATA MARK 3?
F16A| D0E6         BNE    RSYNC1     ; (IF NOT, IS IT DM1?)
F16C|             ; (CARRY SET IF DM3!)

```

← Seems like "Note"  
should be "Not"

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 6

```

F16C| EA          NOP          ; DELAY BETWEEN NIBLS.
F16D| EA          NOP          ; DELAY BETWEEN NIBLS.
F16E| BD 8CC0     RD4         LDA    Q6L,X
F171| 10FB        BPL    RD4    ; *** NO PAGE CROSS! ***
F173| 99 0203     STA    NBUF2,Y ; STORE BYTES DIRECTLY
F176| AD EFFF     LDA    INTERRUPT ; POLL INTERRUPT LINE
F179| 05 8B       ORA    IMASK    ; (THIS MAY BE USED TO INVALIDATE POLL)
F17B| 1037        BPL    GOSERV
F17D| 88         DEY          ; INDEX TO NEXT
F17E| 10EE        BPL    RD4
F180| C8         RD5         INY    ; (FIRST TIME Y=0)
F181| BD 8CC0     RD5A        LDA    Q6L,X ; GET ENCODED BYTES OF NBUF1
F184| 10FB        BPL    RD5A
F186| 99 0002     STA    NBUF1,Y
F189| AD EFFF     LDA    INTERRUPT ; POLL INTERRUPT LINE
F18C| 05 8B       ORA    IMASK    ; (THIS MAY BE USED TO INVALIDATE POLL)
F18E| 1024        BPL    GOSERV
F190| C0 E4        CPY    #0E4    ; WITHIN 1 MS OF COMPLETION?
F192| D0EC        BNE    RD5
F194| C8         RD6         INY    ; NO POLL FROM NOW ON
F195| BD 8CC0     RD6         LDA    Q6L,X
F198| 10FB        BPL    RD6
F19A| 99 0002     STA    NBUF1,Y
F19D| C8         RD6         INY    ; FINISH OUT NBUF1 PAGE
F19E| D0F5        BNE    RD6
F1A0| BD 8CC0     RDCKSUM     LDA    Q6L,X ; GET CHECKSUM BYTE.
F1A3| 10FB        BPL    RDCKSUM
F1A5| 85 96       STA    CKSUM
F1A7| 20 01F2     JSR    RDA6    ; CHECK BIT SLIP MARKS
F1AA|             ;
F1AA|             ; CHECK FOR INTERRUPTS
F1AA|             ;
F1AA| 24 8B       CHKINT     BIT    IMASK ; SHOULD INTERRUPTS BE ALLOWED?
F1AC| 1004        BPL    $010    ; YES, ALLOW THEM.
F1AE| 24 8F       BIT    $08F
F1B0| 1001        BPL    $020
F1B2| 58         $010        CLI
F1B3| 60         $020        RTS
F1B4|             ;
F1B4| 20 AAF2     GOSERV     JSR    SERVICE ; GO TO SERVICE INTERRUPT
F1B7| 38         RDERR      SEC
F1B8| 60         RTS
F1B9|             ;
F1B9|             ; *****
F1B9|             ;
F1B9|             ; READ ADDRESS FIELD
F1B9|             ; SUBROUTINE
F1B9|             ; (16-SECTOR FORMAT)
F1B9|             ;
F1B9|             ; *****
F1B9|             ;
F1B9|             ; READS VOLUME, TRACK
F1B9|             ; AND SECTOR
F1B9|             ;
F1B9|             ; ---- ON ENTRY ----
F1B9|             ;
F1B9|             ; XREG: SLOTNUM TIMES $10
F1B9|             ;
F1B9|             ; READ MODE (Q6L, Q7L)
F1B9|             ;
F1B9|             ; ---- ON EXIT ----
F1B9|             ;
F1B9|             ; CARRY SET IF ERROR
F1B9|             ;
F1B9|             ; IF NO ERROR:
F1B9|             ; A-REG HOLDS $AA.
F1B9|             ; Y-REG HOLDS $00.
F1B9|             ; X-REG UNCHANGED.
F1B9|             ; CARRY CLEAR.
F1B9|             ;
F1B9|             ; CSSTV HOLDS CHKSUM,
F1B9|             ; SECTOR, TRACK, AND
F1B9|             ; VOLUME READ.
F1B9|             ;
F1B9|             ; USES TEMPS COUNT,
F1B9|             ; LAST, CSUM, AND
F1B9|             ; 4 BYTES AT CSSTV.
F1B9|             ;
F1B9|             ; ---- EXPECTS ----
F1B9|             ;
F1B9|             ; ORIGINAL 10-SECTOR
F1B9|             ; NORMAL DENSITY NIBLS
F1B9|             ; (4-BIT), ODD BITS,
F1B9|             ; THEN EVEN
F1B9|             ;
F1B9|             ; ---- CAUTION ----
F1B9|             ;
F1B9|             ; OBSERVE
F1B9|             ;
F1B9|             ; 'NO PAGE CROSS'
F1B9|             ; WARNINGS ON

```



10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 7

```

F1B9|          ;      SOME BRANCHES!!      *
F1B9|          ;                          *
F1B9|          ;      ---- ASSUMES ----      *
F1B9|          ;                          *
F1B9|          ;      1 USEC CYCLE TIME      *
F1B9|          ;                          *
F1B9|          ;*****
F1B9| A0 FC      RDA16  LDY      #0FC
F1BB| 84 95      STY      COUNT      ; 'MUST FIND' COUNT.
F1BD| C8        RDASYN INY
F1BE| D004      BNE      RDA1      ; LOW ORDER OF COUNT
F1C0| E6 95      INC      COUNT      ; (2K NIBLS TO FIND
F1C2| F0F3      BEQ      RDERR      ; ADR MARK, ELSE ERR)
F1C4| BD 8CC0    LDA      RDA1      ; READ NIBL.
F1C7| 10FB      BPL      RDA1      ; *** NO PAGE CROSS! ***
F1C9| C9 D5      RDASN1 CMP      #0D5      ; ADR MARK 1?
F1CB| D0F0      BNE      RDASYN      ; (LOOP IF NOT)
F1CD| EA        NOP
F1CE| BD 8CC0    RDA2  LDA      Q6L,X      ; ADDED NIBL DELAY
F1D1| 10FB      BPL      RDA2      ; *** NO PAGE CROSS! ***
F1D3| C9 AA      CMP      #0AA      ; ADR MARK 2?
F1D5| D0F2      BNE      RDASN1      ; (IF NOT, IS IT AM1?)
F1D7| A0 03      LDY      #03      ; INDEX FOR 4-BYTE READ
F1D9|          ;      (ADDED NIBL DELAY)
F1D9| BD 8CC0    RDA3  LDA      Q6L,X
F1DC| 10FB      BPL      RDA3      ; *** NO PAGE CROSS! ***
F1DE| C9 96      CMP      #96      ; ADR MARK 3?
F1E0| D0E7      BNE      RDASN1      ; (IF NOT IS IT AM1?)
F1E2|          ;      (LEAVES CARRY SET!)
F1E2| 78        SEI
F1E3| A9 00      LDA      #00      ; DISABLE INTERRUPT SYSTEM
F1E5| 85 89      RDAFLD STA      CSUM      ; INIT CHECKSUM
F1E7| BD 8CC0    RDA4  LDA      Q6L,X      ; READ 'ODD BIT' NIBBL
F1EA| 10FB      BPL      RDA4      ; *** NO PAGE CROSS! ***
F1EC| 2A        ROL      A      ; ALIGN ODD BITS, 1' INTO LSB
F1ED| 85 95      STA      LAST      ; (SAVE THEM)
F1EF| BD 8CC0    RDA5  LDA      Q6L,X      ; READ 'EVEN BIT' NIBL
F1F2| 10FB      BPL      RDA5      ; *** NO PAGE CROSS ***
F1F4| 25 95      AND      LAST      ; MERGE ODD AND EVEN BITS
F1F6| 99 97 00   STA      CSSTV,Y      ; STORE DATA BYTE
F1F9| 45 89      EOR      CSUM
F1FB| 88        DEY
F1FC| 10E7      BPL      RDAFLD      ; LOOP ON 4 DATA BYTES.
F1FE| A8        TAY
F1FF| D0B6      BNE      RDERR      ; IF FINAL CHECKSUM
F201| BD 8CC0    RDA6  LDA      Q6L,X      ; NONZERO, THEN ERROR
F204| 10FB      BPL      RDA6      ; FIRST BIT SLIP NIBBL
F206| C9 DE      CMP      #0DE      ; *** NO PAGE CROSS! ***
F208| D0AD      BNE      RDERR      ; ERROR IF NONMATCH
F20A| EA        NOP
F20B| BD 8CC0    RDA7  LDA      Q6L,X      ; DELAY
F20E| 10FB      BPL      RDA7      ; SECOND BIT-SLIP NIBL
F210| C9 AA      CMP      #0AA      ; *** NO PAGE CROSS! ***
F212| D0A3      BNE      RDERR      ; ERROR IF NOMATCH
F214| 18        RDEXIT CLC
F215| 60        WEXIT  RTS      ; CLEAR CARRY ON
F216|          ;      NORMAL READ EXITS.
F216|          ;*****
F216|          ;      WRITE SUBR      *
F216|          ;      (16-SECTOR FORMAT) *
F216|          ;*****
F216|          ;      WRITES DATA FROM *
F216|          ;      NBUF1 AND NBUF2 *
F216|          ;      *
F216|          ;      FIRST NBUF2, *
F216|          ;      HIGH TO LOW. *
F216|          ;      THEN NBUF1, *
F216|          ;      LOW TO HIGH *
F216|          ;      *
F216|          ;      ---- ON ENTRY ---- *
F216|          ;      *
F216|          ;      X-REG  SLOTNUM *
F216|          ;      TIMES $10 *
F216|          ;      *
F216|          ;      ---- ON EXIT ---- *
F216|          ;      *
F216|          ;      CARRY SET IF ERROR. *
F216|          ;      (W PROT VIOLATION) *
F216|          ;      *
F216|          ;      IF NO ERROR: *
F216|          ;      *
F216|          ;      A-REG UNCERTAIN. *
F216|          ;      X-REG UNCHANGED. *
F216|          ;      Y-REG HOLDS $00. *
F216|          ;      CARRY CLEAR. *

```

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 8

```

F216|      ;      *
F216|      ; ---- ASSUMES ---- *
F216|      ;      *
F216|      ; 1 USEC CYCLE TIME *
F216|      ;      *
F216|      ;*****
F216|      ;
F216| 38      WRITE16  SEC      ; ANTICIPATE WPROT ERR.
F217| B8      CLV      ; TO INDICATE WRITE PROTECT ERROR INSTEAD OF
F218|      ; INTERRUPT
F218| BD 8DC0      LDA      Q6H,X
F21B| BD 8EC0      LDA      Q7L,X      ; SENSE WPROT FLAG.
F21E| 30F5      BMI      WEXIT      ; BRANCH IF WRITE PROTECTED
F220| A9 FF      LDA      #0FF      ; SYNC DATA.
F222| 9D 8FC0      STA      Q7H,X      ; (5) GOTO WRITE MODE
F225| 1D 8CC0      ORA      Q6L,X      ; (4)
F228| A0 04      LDY      #04      ; (2) FOR FIVE NIBLS.
F22A| EA      NOP      ; (2)
F22B| 48      PHA      ; (4)
F22C| 68      PLA      ; (3)
F22D| 48      WSYNC    PHA      ; (4) EXACT TIMING
F22E| 68      PLA      ; (3)
F22F| 20 BBF2      JSR      WNIBL7      ; (13,9,6) WRITE SYNC
F232| 88      DEY      ; (2)
F233| D0F8      BNE      WSYNC      ; (2*) MUST NOT CROSS PAGE!
F235| A9 D5      LDA      #0D5      ; (2) 1ST DATA MARK
F237| 20 BAF2      JSR      WNIBL9      ; (15,9,6)
F23A| A9 AA      LDA      #0AA      ; (2) 2ND DATA MARK
F23C| 20 BAF2      JSR      WNIBL9      ; (15,9,6)
F23F| A9 AD      LDA      #0AD      ; (2) 3RD DATA MARK
F241| 20 BAF2      JSR      WNIBL9      ; (15,9,6)
F244| A0 55      LDY      #55      ; (2) NBUF2 INDEX
F246| EA      NOP      ; (2) FOR TIMING
F247| EA      NOP      ; (2)
F248| EA      NOP      ; (2)
F249| D008      BNE      VRYFRST      ; (3) BRANCH ALWAYS
F24B| AD EFFF      WINTRPT LDA      INTERRUPT      ; (4) POLL INTERRUPT LINE
F24E| 05 8B      ORA      IMASK      ; (3)
F250| 38      SEC      ; (2)
F251| 1057      BPL      SERVICE      ; (2) BRANCH IF INTERRUPT HAS OCCURED
F253| 3000      BMI      WRTFRST      ; (3) FOR TIMING.
F255| B9 0203      LDA      NBUF2,Y      ; (4)
F258| 9D 8DC0      STA      Q6H,X      ; (5) STORE ENCODED BYTE
F25B| BD 8CC0      LDA      Q6L,X      ; (4) TIME MUST = 32 US PER BYTE!
F25E| 88      DEY      ; (2)
F25F| 10EA      BPL      WINTRPT      ; (3) (2 IF BRANCH NOT TAKEN)
F261| 98      TYA      ; (2) INSURE NO INTERRUPT THIS BYTE
F262| 3003      BMI      WMIDLE      ; (3) BRANCH ALWAYS.
F264| AD EFFF      WNTRPT1 LDA      INTERRUPT      ; (4) POLL INTERRUPT LINE
F267| 05 8B      WNTRPT1 ORA      IMASK      ; (3)
F269| 38      SEC      ; (2)
F26A| 3002      BMI      WDATA2      ; (3) BRANCH IF NO INTERRUPT
F26C| 103C      BPL      SERVICE      ; GO SERVICE INTERRUPT.
F26E| C8      INY      ; (2)
F26F| B9 0002      LDA      NBUF1,Y      ; (4)
F272| 9D 8DC0      STA      Q6H,X      ; (5) STORE ENCODED BYTE
F275| BD 8CC0      LDA      Q6L,X      ; (4)
F278| C0 E4      CPY      #0E4      ; (2) WITHIN 1 MS OF COMPLETION?
F27A| D0E8      BNE      WNTRPT1      ; (3) (2) NO KEEP WRITING AND POLLING.
F27C| EA      NOP      ; (2)
F27D| C8      INY      ; (2)
F27E| EA      WDATA3    NOP      ; (2)
F27F| EA      NOP      ; (2)
F280| 48      PHA      ; (4)
F281| 68      PLA      ; (3)
F282| B9 0002      LDA      NBUF1,Y      ; (4) WRITE LAST OF ENCODED BYTES
F285| 9D 8DC0      STA      Q6H,X      ; (5) WITHOUT POLLING INTERRUPTS.
F288| BD 8CC0      LDA      Q6L,X      ; (4)
F28B| A5 96      LDA      CKSUM      ; (3) NORMALLY FOR TIMING
F28D| C8      INY      ; (2)
F28E| D0EE      BNE      WDATA3      ; (3) (2)
F290| F000      BEQ      WRCKSUM      ; (3) BRANCH ALWAYS
F292| 20 BBF2      WRCKSUM JSR      WNIBL7      ; (13,9,6) GO WRITE CHECK SUM!!
F295| 48      PHA      ; (3)
F296| 68      PLA      ; (4)
F297| B9 C0F3      WRBITSLMK LDA      BITSIPMK,Y      ; (4) LOAD BIT SLIP MARK
F29A| 20 BDF2      JSR      WNIBL      ; (6,9,6)
F29D| C8      INY      ; (2)
F29E| C0 04      CPY      #04      ; (2)
F2A0| D0F5      BNE      WRBITSLMK      ; (2) (3)
F2A2| 18      CLC      ; (2)
F2A3| BD 8EC0      NOWRITE LDA      Q7L,X      ; OUT OF WRITE MODE.
F2A6| BD 8CC0      LDA      Q6L,X      ; TO READ MODE.
F2A9| 60      RTS      ; RETURN FROM WRITE.
F2AA|      ;
F2AA| 2C 54F3      SERVICE BIT      SEV      ; SET VFLAG TO INDICATE INTERRUPT
F2AD| 20 A3F2      JSR      NOWRITE      ; TAKE IT OUT OF WRITE MODE!
F2B0| A5 8F      LDA      #08F
F2B2| 1002      BPL      $010
F2B4| 85 8B      STA      IMASK

```

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 9

```

F2B6| C6 8F          $010    DEC    08F
F2B8| 58
F2B9| 60            CLI
F2BA|              RTS      ; COULD NOT HAVE GOT HERE WITHOUT CLI OK
F2BA|              ;
F2BA|              ;*****
F2BA|              ;
F2BA|              ; 7-BIT NIBL WRITE SUBRS *
F2BA|              ;
F2BA|              ; A-REG OR'D PRIOR EXIT *
F2BA|              ; CARRY CLEARED *
F2BA|              ;
F2BA|              ;*****
F2BA|              ;
F2BA| 18            WNIBL9   CLC      ; (2) 9 CYCLES, THEN WRITE
F2BB| 48            WNIBL7   PHA      ; (3) 7 CYCLES, THEN WRITE
F2BC| 68            PLA      ; (4)
F2BD| 9D 8DC0      WNIBL    STA     Q6H,X ; (5) NIBL WRITE SUB
F2C0| 1D 8CC0      ORA     Q6L,X ; (4) CLOBBERS ACC. NOT CARRY
F2C3| 60            RTS
F2C4|              ;
F2C4|              ;*****
F2C4|              ;
F2C4|              ; PRENIBILIZE SUBR *
F2C4|              ; (16-SECTOR FORMAT) *
F2C4|              ;
F2C4|              ;*****
F2C4|              ;
F2C4|              ; CONVERTS 256 BYTES OF *
F2C4|              ; USER DATA IN (BUF) INTO *
F2C4|              ; ENCODED BYTES TO BE *
F2C4|              ; WRITTEN DIRECTLY TO DISK *
F2C4|              ; ENCODED CHECK SUM IN *
F2C4|              ; ZERO PAGE 'CKSUM' *
F2C4|              ;
F2C4|              ; ---- ON ENTRY ---- *
F2C4|              ;
F2C4|              ; BUF IS 2-BYTE POINTER *
F2C4|              ; TO 256 BYTES OF USER *
F2C4|              ; DATA. *
F2C4|              ;
F2C4|              ; A-REG CHECK SUM. *
F2C4|              ; X-REG UNCERTAIN *
F2C4|              ; Y-REG HOLDS 0. *
F2C4|              ; CARRY SET. *
F2C4|              ;
F2C4|              ;*****
F2C4|              ;
F2C4| A2 02          PRENIB16 LDX     #02      ; START NBUF2 INDEX.
F2C6| A0 00          LDY     #00      ; START USER BUF INDEX.
F2C8| 88            DEY
F2C9| B1 9B          LDA     (BUF),Y ; NEXT USER BYTE
F2CB| 4A            LSR     A         ; SHIFT TWO BITS OF
F2CC| 3E 0103        ROL     NBUF2-1,X ; CURRENT USER BYTE
F2CF| 4A            LSR     A         ; INTO CURRENT NBUF2
F2D0| 3E 0103        ROL     NBUF2-1,X ; BYTE.
F2D3| 99 0102        STA     NBUF1+1,Y ; (6 BITS LEFT).
F2D6| E8            INX
F2D7| E0 56          CPX     #56      ; FROM 0 TO $55
F2D9| 90ED          BCC     PRENIB1   ; BR IF NO WRAPAROUND
F2DB| A2 00          LDX     #00      ; RESET NBUF2 INDEX
F2DD| 98            TYA
F2DE| D0E8          BNE     PRENIB1   ; USER BUF INDEX
F2E0| A0 56          LDY     #56      ; (DONE IF ZERO)
F2E2| 59 0003        EOR     NBUF2-2,Y ; (ACC=0 FOR CHECK SUM)
F2E5| 29 3F          AND     #03F     ; COMBINE WITH PREVIOUS
F2E7| AA            TAX
F2E8| BD 55F3        LDA     NIBL,X   ; STRIP GARBAGE BITS
F2EB| 99 0103        STA     NBUF2-1,Y ; TO FORM RUNNING CHECK SUM
F2EE| B9 0003        LDA     NBUF2-2,Y ; GET ENCODED EQUIV.
F2F1| 88            DEY
F2F2| D0EE          BNE     PRENIB3   ; REPLACE PREVIOUS
F2F4| 29 3F          AND     #3F      ; RESTORE ACTUAL PREVIOUS
F2F6| 59 0102        EOR     NBUF1+1,Y ; LOOP UNTIL ALL OF NBUF2 IS CONVERTED.
F2F9| AA            TAX
F2FA| BD 55F3        LDA     NIBL,X   ; NOW DO THE SAME FOR
F2FD| 99 0002        STA     NBUF1,Y  ; NIBBLE BUFFER 1
F300| B9 0102        LDA     NBUF1+1,Y ; TO DO ANY BACK TRACKING (NBUF1-1)
F303| C8            INY
F304| D0F0          BNE     PRENIB4   ; RECOVER THAT WHICH IS NOW 'PREVIOUS'
F306| AA            TAX
F307| BD 55F3        LDA     NIBL,X   ; USE LAST AS CHECK SUM
F30A| 85 96          STA     CKSUM
F30C| 4C 4CF3        JMP     SETMEG   ; ALL DONE.
F30F|              ;
F30F|              ;*****
F30F|              ;
F30F|              ; POSTNIBLIZE SUBR *
F30F|              ; 16-SECTOR FORMAT *
F30F|              ;
F30F|              ;*****

```

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 10

```

F30F| 38 ; POSTNIB16 SEC
F310| A0 55 LDY #55 ; FIRST CONVERT TO 6 BIT NIBBLES
F312| A9 00 LDA #00 ; INIT CHECK SUM
F314| BE 0203 PNIBL1 LDX NBUF2,Y ; GET ENCODED BYTE
F317| 5D 00F3 EOR DNIBL,X
F31A| 3030 BMI SET1MEG ; SET 1 MHZ
F31C| 99 0203 STA NBUF2,Y ; REPLACE WITH 6 BIT EQUIV.
F31F| 88 DEY
F320| 10A6 BPL PRENIB1 ; LOOP UNTIL DONE WITH NIBBLE BUFFER 2
F322| C8 INY ; NOW Y=0
F323| BE 0002 PNIBL2 LDX NBUF1,Y ; DO THE SAME WITH
F326| 5D 00F3 EOR DNIBL,X
F329| 99 0002 STA NBUF1,Y ; NIBBLE BUFFER 1
F32C| C8 INY ; DO ALL 256 BYTES
F32D| D0F4 BNE PNIBL2
F32F| A6 96 LDX CKSUM ; MAKE SURE CHECK SUM MATCHES
F331| 5D 00F3 EOR DNIBL,X ; BETTER BE ZERO
F334| D016 BNE POSTERR ; BRANCH IF IT IS
F336| A2 56 POST1 LDX #56 ; INIT NBUF2 INDEX
F338| CA POST2 DEX ; NBUF IDX $55 TO $00
F339| 30FB BMI POST1 ; WRAPAROUND IF NEG
F33B| B9 0002 LDA NBUF1,Y
F33E| 5E 0203 LSR NBUF2,X ; SHIFT 2 BITS FROM
F341| 2A ROL A ; CURRENT NBUF2 NIBL
F342| 5E 0203 LSR NBUF2,X ; CURRENT NBUF1
F345| 2A ROL A ; NIBL.
F346| 91 9B STA (BUF),Y ; BYTE OF USER DATA
F348| C8 INY ; NEXT USER BYTE
F349| D0ED BNE POST2
F34B| 18 CLC ; GOOD DATA
F34C| F34C POSTERR .EQU *
F34C| AD DFFF SET1MEG LDA ENVIRON
F34F| 09 80 ORA #ONEMEG ; SET TO ONE MEGAHERTZ CLOCK RATE
F351| 8D DFFF STA ENVIRON
F354| 60 SEV RTS ; (SEV USED TO SET VFLAG)
F355| ;
F355| ; *****
F355| ; 6-BIT TO 7-BIT *
F355| ; NIBL CONVERSION TABLE *
F355| ; *****
F355| ;
F355| ; CODES WITH MORE THAN *
F355| ; ONE PAIR OF ADJACENT *
F355| ; ZEROES OR WITH NO *
F355| ; ADJACENT ONES (EXCEPT *
F355| ; B7) ARE EXCLUDED. *
F355| ; *****
F355| ;
F355| NIBL .BYTE 96,97,9A,9B,9D,9E,9F,0A6,0A7,0AB,0AC,0AD,0AE,0AF,0B2,0B3,0B4,0B5
F35C| A6 A7 AB AC AD AE AF
F363| B2 B3 B4 B5
F367| B6 B7 B9 BA BB BC BD .BYTE 0B6,0B7,0B9,0BA,0BB,0BC,0BD,0BE,0BF,0CB,0CD,0CE,0CF,0D3,0D6,0D7
F36E| BE BF CB CD CE CF D3
F375| D6 D7
F377| D9 DA DB DC DD DE DF .BYTE 0D9,0DA,0DB,0DC,0DD,0DE,0DF,0E5,0E6,0E7,0E9,0EA,0EB,0EC,0ED,0EE
F37E| E5 E6 E7 E9 EA EB EC
F385| ED EE
F387| EF F2 F3 F4 F5 F6 F7 .BYTE 0EF,0F2,0F3,0F4,0F5,0F6,0F7,0F9,0FA,0FB,0FC,0FD,0FE,0FF
F38E| F9 FA FB FC FD FE FF
F395| ;
F395| ; *****
F395| ; 7-BIT TO 6-BIT *
F395| ; 'DENIBLIZE' TABL *
F395| ; (16-SECTOR FORMAT) *
F395| ;
F395| ; VALID CODES *
F395| ; $96 TO $FF ONLY. *
F395| ;
F395| ; CODES WITH MORE THAN *
F395| ; ONE PAIR OF ADJACENT *
F395| ; ZEROES OR WITH NO *
F395| ; ADJACENT ONES (EXCEPT *
F395| ; BIT 7) ARE EXCLUDED *
F395| ; *****
F395| ;
F395| DNIBL .EQU REGRWTS+300
F395| 01 00 01 .BYTE 01,00,01
F398| 98 99 02 03 9C 04 05 .BYTE 98,99,02,03,9C,04,05,06,0A0,0A1,0A2,0A3,0A4,0A5,07,08,0A8
F39F| 06 A0 A1 A2 A3 A4 A5
F3A6| 07 08 A8
F3A9| A9 AA 09 0A 0B 0C 0D .BYTE 0A9,0AA,09,0A,0B,0C,0D,0B0,0B1,0E,0F,10,11,12,13,0B8,14,15
F3B0| B0 B1 0E 0F 10 11 12
F3B7| 13 B8 14 15
F3BB| 16 17 18 19 1A .BYTE 16,17,18,19,1A

```

10/31/89 9:56

## HD:Apple ///:ROM - Disk I/O

Page 11

F3C0	DE	AA	EB	FF	C4	C5	C6	BITS LIPMK	.BYTE	0DE, 0AA, 0EB, 0FF, 0C4, 0C5, 0C6, 0C7, 0C8, 0C9, 0CA, 1B, 0CC, 1C, 1D, 1E		
F3C7	C7	C8	C9	CA	1B	CC	1C					
F3CE	1D	1E										
F3D0	D0	D1	D2	1F	D4	D5	20	.BYTE	0D0, 0D1, 0D2, 1F, 0D4, 0D5, 20, 21, 0D8, 22, 23, 24, 25, 26, 27, 28, 0E0, 0E1			
F3D7	21	D8	22	23	24	25	26					
F3DE	27	28	E0	E1								
F3E2	E2	E3	E4	29	2A	2B	E8	.BYTE	0E2, 0E3, 0E4, 29, 2A, 2B, 0E8, 2C, 2D, 2E, 2F, 30, 31, 32, 0F0, 0F1, 33, 34			
F3E9	2C	2D	2E	2F	30	31	32					
F3F0	F0	F1	33	34								
F3F4	35	36	37	38	F8	39	3A	.BYTE	35, 36, 37, 38, 0F8, 39, 3A, 3B, 3C, 3D, 3E, 3F			
F3FB	3B	3C	3D	3E	3F							

```

F400| *****
F400| ;
F400| ; FAST SEEK SUBROUTINE
F400| ;
F400| ; *****
F400| ;
F400| ; ---- ON ENTRY ----
F400| ;
F400| ; X-REG HOLDS SLOTNUM
F400| ; TIMES $10
F400| ;
F400| ; A-REG HOLDS DESIRED
F400| ; HALFTRACK.
F400| ;
F400| ; CURTRK HOLDS DESIRED
F400| ; HALFTRACK.
F400| ;
F400| ; ---- ON EXIT ----
F400| ;
F400| ; A-REG UNCERTAIN.
F400| ; Y-REG UNCERTAIN.
F400| ; X-REG UNDISTURBED.
F400| ;
F400| ; CURTRK AND TRKN HOLD
F400| ; FINAL HALFTRACK.
F400| ;
F400| ; PRIOR HOLDS PRIOR
F400| ; HALFTRACK IF SEEK
F400| ; WAS REQUIRED.
F400| ;
F400| ; MONTIMEL AND MONTIMEH
F400| ; ARE INCREMENTED BY
F400| ; THE NUMBER OF
F400| ; 100 USEC QUANTUMS
F400| ; REQUIRED BY SEEK
F400| ; FOR MOTOR ON TIME
F400| ; OVERLAP.
F400| ;
F400| ; --- VARIABLES USED ---
F400| ;
F400| ; CURTRK, TRKN, COUNT,
F400| ; PRIOR, SLOTTEMP
F400| ; MONTIMEL, MONTIMEH
F400| ; *****
F400| ;
F400| 85 9E SEEK STA TRKN ; SAVE TARGET TRACK
F400| C5 8C CMP CURTRK ; ON DESIRED TRACK?
F400| F042 BEQ SETPHASE ; YES, ENERGIZE PHASE AND RETURN
F400| A9 00 LDA #00
F400| 85 95 STA TRKCNT ; HALFTRACK COUNT.
F400| A5 8C SEEK2 LDA CURTRK ; SAVE CURTRK FOR
F400| 85 9D STA PRIOR ; DELAYED TURN OFF.
F400| 38 SEC
F400| E5 9E SBC TRKN ; DELTA-TRACKS.
F400| F031 BEQ SEEKEND ; BR IF CURTRK=DESTINATION
F400| B006 BCS OUT ; (MOVE OUT, NOT IN)
F400| 49 FF EOR #0FF ; CALC TRKS TO GO.
F400| E6 8C INC CURTRK ; DECR CURRENT TRACK (OUT)
F400| 9004 BCC MINTST ; (ALWAYS TAKEN).
F400| 69 FE OUT ADC #0FE ; CALC TRACKS TO GO.
F400| C6 8C DEC CURTRK ; DECR CURRENT TRACK (OUT)
F400| C5 95 MINTST CMP TRKCNT
F400| 9002 BCC MAXTST ; AND 'TRKS MOVED'
F400| A5 95 LDA TRKCNT
F400| C9 09 MAXTST CMP #09
F400| B002 BCS STEP2 ; IF TRKCNT>$08 LEAVE Y ALONE (Y=$08)
F400| A8 STEP TAY ; ELSE SET ACCELERATION INDEX IN Y
F400| 38 SEC
F400| 20 48F4 STEP2 JSR SETPHASE
F400| B9 67F4 LDA ONTABLE,Y ; FOR 'ONTIME'
F400| 20 56F4 JSR MSWAIT ; (100 USEC INTERVALS)
F400| A5 9D LDA PRIOR
F400| 18 CLC ; FOR PHASE OFF
F400| 20 4AF4 JSR CLRPHASE ; TURN OFF PRIOR PHASE
F400| B9 70F4 LDA OFFTABLE,Y ; THEN WAIT 'OFFTIME'
F400| 20 56F4 JSR MSWAIT ; (100 USEC INTERVALS)
F400| E6 95 INC TRKCNT ; 'TRACKS MOVED' COUNT.

```

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 12

```

F442| D0C6          BNE      SEEK2      ; (ALWAYS TAKEN)
F444| 20 56F4      SEEKEND JSR      MSWAIT ; SETTLE 25 MSEC
F447| 18            CLC                ; SET FOR PHASE OFF
F448| A5 8C          SETPHASE LDA      CURTRK ; GET CURRENT TRACK
F44A| 29 03          CLRPHASE AND      #03    ; MASK FOR 1 AND 4 PHASES
F44C| 2A            ROL                ; DOUBLE FOR PHASE ON/OFF INDEX
F44D| 05 81          ORA      IBSLOT
F44F| AA            TAX
F450| BD 80C0        LDA      PHASEOFF,X ; TURN ON/OFF ONE PHASE
F453| A6 81          LDX      IBSLOT      ; RESTORE X-REG
F455| 60            SEEKRTS RTS          ; AND RETURN
F456|                ;
F456|                ; *****
F456|                ;
F456|                ; MSWAIT SUBROUTINE
F456|                ;
F456|                ; *****
F456|                ;
F456|                ; DELAYS A SPECIFIED
F456|                ; NUMBER OF 100 USEC
F456|                ; INTERVALS FOR MOTOR
F456|                ; ON TIMING
F456|                ;
F456|                ; ---- ON EXIT ----
F456|                ;
F456|                ; A-REG HOLDS $00
F456|                ; X-REG HOLDS $00
F456|                ; Y-REG UNCHANGED
F456|                ; CARRY SET
F456|                ;
F456|                ; MONTIMEL, MONTIMEH
F456|                ; ARE INCREMENTED ONCE
F456|                ; PER 100 USEC INTERVAL
F456|                ; FOR MOTOR ON TIMING
F456|                ;
F456|                ; ---- ASSUMES ----
F456|                ;
F456|                ; 1 USEC CYCLE TIME
F456|                ;
F456|                ; *****
F456|                ;
F456| A2 11          MSWAIT LDX      #11
F458| CA            MSW1    DEX          ; DELAY 86 USEC
F459| D0FD          BNE      MSW1
F45B| E6 99          INC      MONTIMEL
F45D| D002          BNE      MSW2      ; DOUBLE BYTE INCREMENT
F45F| E6 9A          INC      MONTIMEH
F461| 38            MSW2    SEC
F462| E9 01          SBC      #01      ; DONE IN INTERVALS
F464| D0F0          BNE      MSWAIT    ; (A-REG COUNTS)
F466| 60            RTS
F467|                ;
F467|                ; *****
F467|                ;
F467|                ; PHASE ON-, OFF-TIME
F467|                ; TABLES IN 100-USEC
F467|                ; INTERVALS. (SEEK)
F467|                ;
F467|                ; *****
F467|                ;
F467| 01 30 28 24 20 1E 1D ONTABLE .BYTE 01,30,28,24,20,1E,1D,1C,1C
F46E| 1C 1C
F470| 70 2C 26 22 1F 1E 1D OFFTABLE .BYTE 70,2C,26,22,1F,1E,1D,1C,1C
F477| 1C 1C
F479|                ;
F479| 86 83          BLOCKIO STX      IBTRK
F47B| A0 05          LDY      #05
F47D| 48            PHA
F47E| 0A            TRKSEC ASL      A
F47F| 26 83          ROL      IBTRK
F481| 88            DEY
F482| D0FA          BNE      TRKSEC
F484| 68            PLA
F485| 29 07          AND      #07
F487| A8            TAY
F488| B9 A0F4        LDA      SECTABL,Y
F48B| 85 84          STA      IBSECT
F48D| 20 00F0       JSR      REGRWTS
F490| B00B          BCS      QUIT
F492| E6 86          INC      IBBUFP+1
F494| E6 84          INC      IBSECT
F496| E6 84          INC      IBSECT
F498| 20 00F0       JSR      REGRWTS
F49B| C6 86          DEC      IBBUFP+1
F49D| A5 88          QUIT   LDA      IBSTAT
F49F| 60            RTS
F4A0|                ;
F4A0| 00 04 08 0C 01 05 09 SECTABL .BYTE 00,04,08,0C,01,05,09,0D
F4A7| 0D
F4A8|                ; *****

```

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 13

```

F4A8|          ;          *
F4A8|          ; JOYSTICK READ ROUTINE          *
F4A8|          ;          *
F4A8|          ; *****
F4A8|          ; ENTRY ACC= COUNT DOWN HIGH *
F4A8|          ; X&Y= DON'T CARE          *
F4A8|          ;          *
F4A8|          ; EXIT ACC= TIMER HIGH BYTE *
F4A8|          ; Y= TIMER LOW BYTE          *
F4A8|          ; CARRY CLEAR          *
F4A8|          ;          *
F4A8|          ; IF CARRY SET, ROUTINE          *
F4A8|          ; WAS INTERRUPTED &          *
F4A8|          ; ACC & Y ARE INVALID          *
F4A8|          ; *****
F4A8|          ;
F4A8| FFD9      TIMLATCH .EQU 0FFD9
F4A8| FFD8      TIMER1L .EQU 0FFD8
F4A8| FFD9      TIMER1H .EQU 0FFD9
F4A8| C066      JOYRDY .EQU 0C066
F4A8|          ;
F4A8| F4A8      ANALOG .EQU *          ; CARRY SHOULD BE SET!
F4A8| 8D D9FF   STA      TIMLATCH          ; START THE TIMER!
F4AB| AD EFFF   ANLOG1  LDA      INTERRUPT
F4AE| 2D 66C0   AND      JOYRDY          ; WAIT FOR ONE OR THE OTHER TO GO LOW
F4B1| 30F8      BMI      ANLOG1
F4B3| AD 66C0   LDA      JOYRDY          ; WAS IT REALLY THE JOYSTICK?
F4B6| 300C      BMI      GOODTIME        ; NOPE, WHAT TIME IS IT?
F4B8| 18        CLC
F4B9| AD D9FF   LDA      TIMER1H          ; TIME'S A SLIP SLIDIN AWAY
F4BC| AC D8FF   LDY      TIMER1L          ; NOW, WHAT TIME IS IT?
F4BF| 1003      BPL      GOODTIME        ; TIME WAS VALID!
F4C1| AD D9FF   LDA      TIMER1H          ; HI BYTE CHANGED
F4C4| 60        GOODTIME RTS
F4C5|          .END

```

## SYMBOL TABLE DUMP

```

AB - Absolute    LB - Label    UD - Undefined    MC - Macro
RF - Ref         DF - Def      PR - Proc         FC - Func
PB - Public      PV - Private  CS - Consts

```

```

ALDONE1 LB F0EB | ALDONE LB F0E3 | ALLOFF LB F118 | ANALOG LB F4A8 | ANLOG1 LB F4AB |
BITSLIP LB F3C0 | BLOCKIO LB F479 | BUF AB 009B | CHKDRV LB F12B | CHKDRV1 LB F12D |
CHKINT LB F1AA | CKDRTS LB F13D | CKSUM AB 0096 | CLRPHASE LB F44A | CONWAIT LB F054 |
CORRECTV LB F0C4 | COUNT AB 0095 | CSSTV AB 0097 | CSUM AB 0089 | CSUM1 AB 0097 |
CURTRK AB 008C | DISKIO PR ---- | DNIBL LB F300 | DRIVSEL LB F035 | DRV1EN AB 008B |
DRVERR LB F0E8 | DRVINDX LB F13E | DRVOEN AB C08A | DRVOTRK AB 0085 | DRVWAIT LB F041 |
DVMT AB 00E0 | ENVIRON AB FFD9 | ENVTEMP AB 009F | GOCAL LB F0A7 | GOCAL1 LB F0A6 |
GOODTIME LB F4C4 | GOSSEEK LB F115 | GOSERV LB F1B4 | HNDLERR LB F0EA | HRDERRS AB 0080 |
IBBUFP AB 0085 | IBCMD AB 0087 | IBDEERR AB 0082 | IBDRVN AB 0082 | IBNODRV AB 0080 |
IBRERR AB 0083 | IBSECT AB 0084 | IBSLOT AB 0081 | IBSMOD AB 0089 | IBSTAT AB 0088 |
IBTRK AB 0083 | IBWPER AB 0081 | IMASK AB 008B | INTERRUPT AB FFEF | IOBPDN AB 008A |
JOYRDY AB C066 | LAST AB 0095 | MAXTST LB F425 | MINTST LB F41F | MONTIMEH AB 009A |
MONTIMEL AB 0099 | MOTOF LB F052 | MOTOROFF AB C088 | MOTORON AB C089 | MSW1 LB F458 |
MSW2 LB F461 | MSWAIT LB F456 | MYSEEK LB F104 | NBUF1 AB 0200 | NBUF2 AB 0302 |
NIBL LB F355 | NODRIVER LB F064 | NOWRITE LB F2A3 | NXOFF LB F11A | OFFTABLE LB F470 |
OK LB F048 | ONEMEG AB 0080 | ONTABLE LB F467 | OUT LB F41B | PHASEOFF AB C080 |
PHASEON AB C081 | PHASON AB C081 | PHSOFF AB C080 | PNIBL1 LB F314 | PNIBL2 LB F323 |
POST1 LB F336 | POST2 LB F338 | POSTERR LB F34C | POSTNIB1 LB F30F | PRENIB1 LB F2C8 |
PRENIB16 LB F2C4 | PRENIB2 LB F2E5 | PRENIB3 LB F2E2 | PRENIB4 LB F2F6 | PRIOR AB 009D |
Q6H AB C08D | Q6L AB C08C | Q7H AB C08F | Q7L AB C08E | QUIT LB F49D |
RD1 LB F14D | RD2 LB F157 | RD3 LB F163 | RD4 LB F16E | RD5 LB F180 |
RD5A LB F181 | RD6 LB F195 | RDA1 LB F1C4 | RDA2 LB F1CE | RDA3 LB F1D9 |
RDA4 LB F1E7 | RDA5 LB F1EF | RDA6 LB F201 | RDA7 LB F20B | RDADR16 LB F1B9 |
RDAFLD LB F1E5 | RDASN1 LB F1C9 | RDASYN LB F1BD | RDCKSUM LB F1A0 | RDERR LB F1B7 |
RDEXIT LB F214 | RDRIGHT LB F0AC | READ16 LB F148 | REGRWTS LB F000 | RETRYCNT AB 0093 |
RSYNC LB F14A | RSYNC1 LB F152 | RTTRK LB F0C0 | SECT AB 0098 | SECTABL LB F4A0 |
SEEK LB F400 | SEEK1 LB F105 | SEEK2 LB F40A | SEEKCNT AB 0094 | SEEKEND LB F444 |
SEEKRTS LB F455 | SERVICE LB F2AA | SETIMEG LB F34C | SETPHASE LB F448 | SETTRK LB F125 |
SEV LB F354 | STEP LB F429 | STEP2 LB F42B | TEMP AB 0097 | TIMER1H AB FFD9 |
TIMER1L AB FFD8 | TIMLATCH AB FFD9 | TRACK AB 0099 | TRKCNT AB 0095 | TRKN AB 009E |
TRKN1 AB 0099 | TRKSEC LB F47E | TRYADR LB F083 | TRYADR2 LB F08A | TRYTRK LB F069 |
TRYTRK2 LB F07F | TWOMEG AB 007F | VOLUME AB 009A | VRYFRST LB F253 | WDATA2 LB F26E |
WDATA3 LB F27E | WEXIT LB F215 | WINTREPT LB F24B | WMIDLE LB F267 | WNIBL LB F2BD |
WNIBL7 LB F2BB | WNIBL9 LB F2BA | WNTREPT1 LB F264 | WRBITSML LB F297 | WRCKSUM LB F292 |
WRIT LB F0F9 | WRIT1 LB F220 | WRITE16 LB F216 | WRTRFRST LB F255 | WSYNC LB F22D |

```

```

Assembly complete: 1076 lines
0 Errors flagged on this Assembly

```

## 6502 OPCODE STATIC FREQUENCIES

```

ADC : 1 m
AND : 8 | *****

```

10/31/89 9:56

HD:Apple ///:ROM - Disk I/O

Page 14

```

ASL : 3 | **
BCC : 10 | *****
BCS : 7 | *****
BEQ : 8 | *****
BIT : 3 | **
BMI : 10 | *****
BNE : 38 | *****
BPL : 28 | *****
BVC : 1 m
CLC : 9 | *****
CLI : 2 | *
CLV : 1 m
CMP : 14 | *****
CPX : 1 m
CPY : 4 | ***
DEC : 5 | ****
DEX : 2 | *
DEY : 13 | *****
EOR : 8 | *****
INC : 10 | *****
INX : 2 | *
INY : 12 | *****
JMP : 2 | *
JSR : 39 | *****
LDA : 86 M *****
LDX : 12 | *****
LDY : 18 | *****
LSR : 9 | *****
NOP : 13 | *****
ORA : 9 | *****
PHA : 10 | *****
PHP : 4 | ***
PLA : 11 | *****
PLP : 3 | **
ROL : 7 | *****
ROR : 6 | *****
RTS : 16 | *****
SBC : 2 | *
SEC : 9 | *****
SEI : 1 m
STA : 42 | *****
STX : 1 m
STY : 3 | **
TAX : 5 | ****
TAY : 3 | **
TXA : 1 m
TYA : 4 | ***

```

Minimum frequency = 1  
Maximum frequency = 86

Average frequency = 10

Unused opcodes:

BRK BVS CLD RTI SED TSX TXS

Program opcode usage: 87 %

-----  
(1.00) That's all, Folks ...  
-----



Source Code Listing  
for

Apple ///

# ROM Diagnostics

David T. Craig  
736 Edgewater  
Wichita, Kansas 67230

10/31/89 9:47

HD:Apple ///:ROM - Sara Tests

Page 1

```

0000| ;*****
0000| ; APPLE /// ROM - DIAGNOSTIC ROUTINES
0000| ; COPYRIGHT 1979 BY APPLE COMPUTER, INC.
0000| ;*****
0000| .ABSOLUTE
0000| .PROC SARATESTS
0000|
0000| ;*****
0000| ;
0000| ; SARA DIAGNOSTIC TEST ROUTINES
0000| ;
0000| ; DECEMBER 18, 1979
0000| ; BY
0000| ; W. BROEDNER & R. LASHLEY
0000| ;
0000| ; COPYRIGHT 1979 BY APPLE COMPUTER, INC.
0000| ;*****
0000| 0001 ROM .EQU 01
0000| 0000 ZRPG .EQU 00
0000| 0010 ZRPG1 .EQU 10
0000| 0018 PTRLO .EQU ZRPG1+08
0000| 0019 PTRHI .EQU ZRPG1+09
0000| 001A BNK .EQU ZRPG1+0A
0000| 0087 IBCMD .EQU 87
0000| 0085 IBBUFP .EQU 85
0000| 0091 PREVTRK .EQU 91
0000| 0000 F479 BLOCKIO .EQU 0F479
0000| 005D CV .EQU 5D
0000| 00FF STK0 .EQU 0FF
0000| 1419 IBNK .EQU 1400+PTRHI
0000| 1810 PHPR .EQU 1800+ZRPG1
0000| C000 KYBD .EQU 0C000
0000| C008 KEYBD .EQU 0C008
0000| C010 KBDSTRB .EQU 0C010
0000| C058 PDLEN .EQU 0C058
0000| C047 ADRS .EQU 0C047
0000| C050 GRMD .EQU 0C050
0000| C051 TXTMD .EQU 0C051
0000| C066 ADTO .EQU 0C066
0000| C0D0 DISKOFF .EQU 0C0D0
0000| C0F1 ACIAST .EQU 0C0F1
0000| C0F2 ACIACM .EQU 0C0F2
0000| C0F3 ACIACN .EQU 0C0F3
0000| C100 SLT1 .EQU 0C100
0000| C200 SLT2 .EQU 0C200
0000| C300 SLT3 .EQU 0C300
0000| C400 SLT4 .EQU 0C400
0000| CFFF EXPROM .EQU 0CFFF
0000| FFD0 ZPREG .EQU 0FFD0
0000| FFD1 SYSD1 .EQU 0FFD1
0000| FFD2 SYSD2 .EQU 0FFD2
0000| FFD3 SYSD3 .EQU 0FFD3
0000| FFE0 SYSE0 .EQU 0FFE0
0000| FFE1 BNKSW .EQU 0FFE1
0000| FFE2 SYSE2 .EQU 0FFE2
0000| FFE3 SYSE3 .EQU 0FFE3
0000| FC25 COUT .EQU 0FC25
0000| FD07 CROUT1 .EQU 0FD07
0000| FD0F KEYIN .EQU 0FD0F
0000| FBC7 SETCVH .EQU 0FBC7
0000| FD98 CLDSTRT .EQU 0FD98
0000| FD9D SETUP .EQU 0FD9D
0000| F901 MONITOR .EQU 0F901
0000| ;
0000| .ORG 0F4C5
F4C5| 00 B1 B2 BA B9 10 00 RAMTBL .BYTE 00,0B1,0B2,0BA,0B9,10,00,13
F4CC| 13
F4CD| F4CD CHPG .EQU *
F4CD| 52 41 .ASCII "RA"
F4CF| CD .BYTE 0CD ; M
F4D0| 52 4F .ASCII "RO"
F4D2| CD .BYTE 0CD ; M
F4D3| 56 49 .ASCII "VI"
F4D5| C1 .BYTE 0C1 ; A
F4D6| 41 43 49 .ASCII "ACI"
F4D9| C1 .BYTE 0C1 ; A
F4DA| 41 2F .ASCII "A/"
F4DC| C4 .BYTE 0C4 ; D
F4DD| 44 49 41 47 4E 4F 53 .ASCII "DIAGNOSTI"
F4E4| 54 49
F4E6| C3 .BYTE 0C3 ; C
F4E7| 5A .ASCII "Z"
F4E8| D0 .BYTE 0D0 ; P
F4E9| 52 45 54 52 .ASCII "RETR"
F4ED| D9 .BYTE 0D9 ; Y
F4EE| ;
F4EE| ; SETUP SYSTEM

```

W= Walt

Broedner later designed the hardware for the Apple IIe computer which was released in January 1983

10/31/89 9:47

HD:Apple ///:ROM - Sara Tests

Page 2

```

F4EE|      ;
F4EE|      ;
F4EE| A9 53      LDA    #52+ROM      ; TURN OFF SCREEN, SET 2MHZ SPEED
F4F0| 8D DFFF    STA    SYSD1      ; AND RUN OFF ROM
F4F3| A2 00      LDX    #00        ; SET BANK SWITCH TO ZERO
F4F5| 8E E0FF    STX    SYSE0
F4F8| 8E EFFF    STX    BNKSW
F4FB| 8E D0FF    STX    ZPREG      ; AND SET ZERO PAGE SAME
F4FE| CA        DEX
F4FF| 8E D2FF    STX    SYSD2      ; PROGRAM DDR'S
F502| 8E D3FF    STX    SYSD3
F505| 9A        TXS
F506| E8        INX
F507| A9 0F      LDA    #0F
F509| 8D E3FF    STA    SYSE3
F50C| A9 3F      LDA    #3F
F50E| 8D E2FF    STA    SYSE2
F511| A0 0E      LDY    #0E
F513| B9 D0C0    DISK1  LDA    DISKOFF,Y
F516| 88        DEY
F517| 88        DEY
F518| 10F9       BPL    DISK1
F51A| AD 08C0    LDA    KEYBD
F51D| 29 04      AND    #04
F51F| D003       BNE    NXBYT
F521| 4C 86F6    JMP    RECON
F524|      ;
F524|      ; VERIFY ZERO PAGE
F524|      ;
F524| A9 01      NXBYT  LDA    #01      ; ROTATE A 1 THROUGH
F526| 95 00      NXBIT  STA    ZRPG,X   ; EACH BIT IN THE 0 PG
F528| D5 00      CMP     ZRPG,X   ; TO COMPLETELY TEST
F52A| D0FE      NOGOOD  BNE    NOGOOD  ; THE PAGE. HANG IF NOGOOD.
F52C| 0A        ASL     A          ; TRY NEXT BIT OF BYTE
F52D| D0F7      BNE    NXBIT      ; UNTIL BYTE IS ZERO.
F52F| E8        INX             ; CONTINUE UNTIL PAGE
F530| D0F2      BNE    NXBYT      ; IS DONE.
F532| 8A        CNTWR  TXA          ; PUSH A DIFFERENT
F533| 48        PHA          ; BYTE ONTO THE
F534| E8        INX             ; STACK UNTIL ALL
F535| D0FB      BNE    CNTWR      ; STCK BYTES ARE FULL.
F537| CA        DEX             ; THEN PULL THEM
F538| 86 18      STX    PTRLO      ; OFF AND COMPARE TO
F53A| 68        PULBT  PLA          ; THE COUNTER GOING
F53B| C5 18      CMP     PTRLO      ; BACKWARDS. HANG IF
F53D| D0EB      BNE    NOGOOD      ; THEY DON'T AGREE.
F53F| C6 18      DEC     PTRLO      ; GET NEXT COUNTER BYTE
F541| D0F7      BNE    PULBT      ; CONTINUE UNTIL STACK
F543| 68        PLA          ; IS DONE. TEST LAST BYTE
F544| D0E4      BNE    NOGOOD      ; AGAINST ZERO.
F546|      ;
F546|      ; SIZE IN MEMORY
F546|      ;
F546| A2 08      NOMEM   LDX    #08      ; ZERO THE BYTES USED TO DISPLAY
F548| 95 10      STA    ZRPG1,X   ; THE BAD RAM LOCATIONS
F54A| CA        DEX             ; EACH BYTE= A CAS LINE
F54B| 10FB      BPL     NOMEM      ; ON THE SARA BOARD.
F54D| A2 02      LDX    #02      ; STARTING AT PAGE 2
F54F| 86 19      NMEM1  STX    PTRHI  ; TEST THE LAST BYTE
F551| A9 00      LDA    #00      ; IN EACH MEM PAGE TO
F553| A0 FF      LDY    #0FF     ; SEE IF THE CHIPS ARE
F555| 91 18      STA    (PTRLO),Y ; THERE.. (AVOID 0 & STK PAGES)
F557| D1 18      CMP     (PTRLO),Y ; CAN THE BYTE BE 0'D?
F559| F007      BEQ     NMEM2
F55B| 20 48F7    JSR     RAM       ; NO, FIND WHICH CAS IT IS.
F55E| 94 10      STY    ZRPG1,X   ; SET CORRES. BYTE TO $FF
F560| A6 19      LDX    PTRHI      ; RESTORE X REGISTER
F562| E8        NMEM2  INX          ; AND INCREMENT TO NEXT
F563| E0 C0      CPX     #0C0      ; PAGE UNTIL I/O IS REACHED.
F565| D0E8      BNE    NMEM1
F567| A2 20      LDX    #20      ; THEN RESET TO PAGE 20
F569| EE EFFF    INC     BNKSW     ; AND GOTO NEXT BANK TO
F56C| AD EFFF    LDA     BNKSW     ; CONTINUE. (MASK INPUTS
F56F| 29 0F      AND     #0F      ; FROM BANKSWITCH TO SEE
F571| C9 03      CMP     #03      ; WHAT SWITCH IS SET TO)
F573| D0DA      BNE    NMEM1      ; CONTINUE UNTIL BANK '3'
F575|      ;
F575|      ; SETUP SCREEN
F575|      ;
F575| 20 9DFD    ERRLP   JSR     SETUP  ; CALL SCRNM SETUP ROUTINE
F578| A2 00      LDX    #00      ; SETUP I/O AGAIN
F57A| 8E E0FF    STX    SYSE0      ; FOR VIA TEST
F57D| CA        DEX             ; PROGRAM DATA DIR
F57E| 8E D2FF    STX    SYSD2      ; REGISTERS
F581| 8E D3FF    STX    SYSD3
F584| A9 3F      LDA    #3F
F586| 8D E2FF    STA    SYSE2
F589| A9 0F      LDA    #0F
F58B| 8D E3FF    STA    SYSE3
F58E| A2 10      LDX    #10      ; HEADING OF 'DIAGNOSTICS' WITH

```

10/31/89 9:47

HD:Apple ///:ROM - Sara Tests

Page 3

```

F590| 20 38F7      JSR    STRWT    ; THIS SUBROUTINE
F593| A2 00      ERRLP1 LDX    #00    ; PRINT 'RAM'
F595| 86 5D      STX    CV        ; SET CURSOR TO 2ND LINE
F597| A9 04      LDA    #04    ; SPACE CURSOR OUT 3
F599| 20 C7FB      JSR    SETCVH   ; (X STILL=0 ON RETURN)
F59C| 20 38F7      JSR    STRWT    ; THE SAME SUBROUTINE
F59F| A2 07      LDX    #07    ; FOR BYTES 7 - 0 IN
F5A1| F5A1      RAMWT1 .EQU    *
F5A1| B5 10      LDA    ZRPG1,X   ; OUT EACH BIT AS A
F5A3| A0 08      LDY    #08    ; ' ' OR '1' FOR INDICATE BAD OR MISSING RAM
F5A5| 0A        RAMWT2 ASL    A    ; CHIPS SUBROUTINE 'RAM' RAM
F5A6| 48        PHA           ; SETS UP THESE BYTES
F5A7| A9 AE      LDA    #0AE    ; LOAD A '.' TO ACC.
F5A9| 9002      BCC    RAMWT4
F5AB| A9 31      LDA    #31    ; LOAD A '1' TO ACC.
F5AD| 20 25FC      JSR    COUT    ; AND PRINT IT
F5B0| 68        PLA           ; RESTORE BYTE
F5B1| 88        DEY           ; AND ROTATE ALL 8
F5B2| D0F1      BNE    RAMWT2    ; TIMES
F5B4| 20 07FD      JSR    CROUT1  ; CLEAR TO END OF LINE.
F5B7| CA        DEX           ;
F5B8| 10E7      BPL    RAMWT1
F5BA|          ;
F5BA|          ; ZPG & STK TEST
F5BA|          ;
F5BA| 9A        TXS           ;
F5BB| 8C EFFF      STY    BNKSW
F5BE| 98        ZP1    TYA
F5BF| 8D D0FF      STA    ZPREG
F5C2| 85 FF      STA    STK0
F5C4| C8        INY
F5C5| 98        TYA
F5C6| 48        PHA
F5C7| 68        PLA
F5C8| C8        INY
F5C9| C0 20      CPY    #20
F5CB| D0F1      BNE    ZP1
F5CD| A0 00      LDY    #00
F5CF| 8C D0FF      STY    ZPREG
F5D2| 86 18      STX    PTRLO
F5D4| E8        ZP2    INX
F5D5| 86 19      STX    PTRHI
F5D7| 8A        TXA
F5D8| D1 18      CMP    (PTRLO),Y
F5DA| D006      BNE    ZP3
F5DC| E0 1F      CPX    #1F
F5DE| D0F4      BNE    ZP2
F5E0| F005      BEQ    ROMTST
F5E2| F5E2      ZP3    .EQU    *
F5E2| A2 1A      LDX    #1A    ; CHIP IS THERE, BAD ZERO AND STACK
F5E4| 20 7BF7      JSR    MESSERR ; SO PRINT 'ZP' MESSAGE
F5E7|          ; & SET FLAG (2MHZ MODE)
F5E7|          ;
F5E7|          ; ROM TEST ROUTINE
F5E7|          ;
F5E7| A9 00      ROMTST LDA    #00    ; SET POINTERS TO
F5E9| A8        TAY           ; $F000
F5EA| A2 F0      LDX    #0F0
F5EC| 85 18      STA    PTRLO
F5EE| 86 19      STX    PTRHI   ; SET X TO $FF
F5F0| A2 FF      LDX    #0FF    ; FOR WINDOWING I/O
F5F2| 51 18      EOR    (PTRLO),Y ; COMPUTE CHKSUM ON
F5F4| E4 19      CPX    PTRHI   ; EACH ROM BYTE,
F5F6| D006      BNE    ROMTST2  ; WINDOW OUT
F5F8| C0 BF      CPY    #0BF    ; RANGES FFC0-FFEF
F5FA| D002      BNE    ROMTST2
F5FC| A0 EF      LDY    #0EF
F5FE| C8        ROMTST2 INY
F5FF| D0F1      BNE    ROMTST1
F601| E6 19      INC    PTRHI
F603| D0ED      BNE    ROMTST1
F605| A8        TAY           ; TEST ACC. FOR 0
F606| F005      BEQ    VIATST   ; YES, NEXT TEST
F608| A2 03      LDX    #03    ; PRINT 'ROM' AND
F60A| 20 7BF7      JSR    MESSERR ; SET ERROR
F60D|          ;
F60D|          ; VIA TEST ROUTINE
F60D|          ;
F60D| 18        VIATST CLC           ; SET UP FOR ADDING BYTES
F60E| D8        CLD
F60F| AD E0FF      LDA    SYSE0   ; MASK OFF INPUT BITS
F612| 29 3F      AND    #3F    ; AND STORE BYTE IN
F614| 85 18      STA    PTRLO   ; TEMPOR. LOCATION
F616| AD EFFF      LDA    BNKSW  ; MASK OFF INPUT BITS
F619| 29 4F      AND    #4F    ; AND ADD TO STORED
F61B| 65 18      ADC    PTRLO   ; BYTE IN TEMP. LOC.
F61D| 6D D0FF      ADC    ZPREG  ; ADD REMAINING
F620| 85 18      STA    PTRLO   ; REGISTERS OF THE
F622| AD DFFF      LDA    SYSD1  ; VIA'S
F625| 29 5F      AND    #5F    ; (MASK THIS ONE)
F627| 65 18      ADC    PTRLO   ; AND TEST

```

10/31/89 9:47

HD:Apple ///:ROM - Sara Tests

Page 4

```

F629| 6D D2FF      ADC    SYSD2      ; TO SEE
F62C| 6D D3FF      ADC    SYSD3      ; IF THEY AGREE
F62F| 6D E2FF      ADC    SYSE2      ; WITH THE RESET
F632| 6D E3FF      ADC    SYSE3      ; CONDITION.
F635| C9 E1         CMP    #0E0+ROM    ; =E1?
F637| F005         BEQ    ACIA        ; YES, NEXT TEST
F639| A2 06         LDX    #06        ; NO, PRINT 'VIA' MESS
F63B| 20 7BF7      JSR    MESSERR     ; AND SET ERROR FLAG
F63E|               ;
F63E|               ; ACIA TEST
F63E|               ;
F63E| 18            ACIA    CLC          ; SET UP FOR ADDITION
F63F| A9 9F         LDA    #9F        ; MASK INPUT BITS
F641| 2D F1C0      AND    ACIAST      ; FROM STATUS REG
F644| 6D F2C0      ADC    ACIACM      ; AND ADD DEFAULT STATES
F647| 6D F3C0      ADC    ACIACN      ; OIF CONTROL AND COMMAND
F64A| C9 10         CMP    #10        ; REGS. =10?
F64C| F005         BEQ    ATD         ; YES, NEXT TEST
F64E| A2 09         LDX    #09        ; NO, 'ACIA' MESSAGE AND
F650| 20 7BF7      JSR    MESSERR     ; THEN SET ERROR FLAG
F653|               ;
F653|               ; A/D TEST ROUTINE
F653|               ;
F653| A9 C0         ATD    LDA    #0C0   ;
F655| 8D DCFE      STA    0FFDC        ;
F658| AD 5AC0      LDA    PDLEN+2      ;
F65B| AD 5EC0      LDA    PDLEN+6      ;
F65E| AD 5CC0      LDA    PDLEN+4      ;
F661| A0 20        LDY    #20         ;
F663| 88           ADCTST1  DEY          ; WAIT FOR 40 USEC
F664| D0FD        BNE    ADCTST1      ;
F666| AD 5DC0      LDA    PDLEN+5      ; SET A/D RAMP
F669| C8           ADCTST3  INY          ; COUNT FOR CONVERSION
F66A| F00A        BEQ    ADCERR      ;
F66C| AD 66C0      LDA    ADTO         ; IF BIT 7=1?
F66F| 30F8        BMI    ADCTST3     ; YES, CONTINUE
F671| 98           TYA          ; NO, MOVE COUNT TO ACC
F672| 29 E0        AND    #0E0        ; ACC<32
F674| F005        BEQ    KEYPLUG     ;
F676| F676        ADCERR  .EQU    *    ; NO,
F676| A2 0D        LDX    #0D         ; PRINT 'A/D' MESS
F678| 20 7BF7      JSR    MESSERR     ; AND SET ERROR FLAG
F67B|               ;
F67B|               ; KEYBOARD PLUGIN TEST
F67B|               ;
F67B| AD 08C0      KEYPLUG  LDA    KEYBD  ; IS KYBD PLUGGED IN?
F67E| 0A           ASL    A           ; (IS LIGHT CURRENT
F67F| 1041         BPL    SEX         ; PRESENT?) NO, BRANCH
F681| AD DFFF      LDA    SYSD1      ; IS ERROR FLAG SET?
F684| 303C        BMI    SEX         ; ERROR HANG
F686|               ;
F686|               ; RECONFIGURE THE SYSTEM
F686|               ;
F686| A9 77        RECON    LDA    #77   ; TURN ON SCREEN
F688| 8D DFFF      STA    SYSD1      ;
F68B| 20 98FD      JSR    CLDSTRT    ; INITIALIZE MONITOR AND DEFAULT CHARACTER SET
F68E| 2C 10C0      BIT    KBDSTRB    ; CLEAR KEYBOARD
F691| AD FFCF      LDA    EXPROM     ; DISABLE ALL SLOTS
F694| AD 20C0      LDA    0C020     ;
F697| A9 10        LDA    #10       ; TEST FOR "APPLE 1"
F699| 2D 08C0      AND    KEYBD     ;
F69C| D003        BNE    BOOT       ; NO, DO REGULAR BOOT
F69E| 20 01F9      JSR    MONITOR    ; AND NEVER COME BACK
F6A1| A2 01        LDX    #01       ; READ BLOCK 0
F6A3| 86 87        STX    IBCMD     ;
F6A5| CA           DEX             ;
F6A6| 86 85        STX    IBBUFP     ; INTO RAM AT $A000
F6A8| A9 A0        LDA    #0A0      ;
F6AA| 85 86        STA    IBBUFP+1 ;
F6AC| 4A           LSR    A         ; FOR TRACK 80
F6AD| 85 91        STA    PREVTRK   ; MAKE IT RECALIBRATE TOO!
F6AF| 8A           TXA             ;
F6B0| 20 79F4      JSR    BLOCKIO   ;
F6B3| 900A        BCC    GOBOOT     ; IF WE'VE SUCCEEDED. DO IT UP
F6B5| A2 1C        LDX    #1C       ;
F6B7| 20 38F7      JSR    STRWT     ; 'RETRY'
F6BA| 20 0FFD      JSR    KEYIN     ;
F6BD| B0E2        BCS    BOOT       ;
F6BF| 4C 00A0      GOBOOT  JMP    0A000 ; GO TO IT FOOL...
F6C2|               ;
F6C2|               ; SYSTEM EXERCISER
F6C2|               ;
F6C2| A0 7F        SEX     LDY    #7F   ; TRY FROM
F6C4| 98           SEX1    TYA         ; $7F TO 0
F6C5| 29 FE        AND    #0FE        ; ADD.=
F6C7| 49 4E        EOR    #4E        ; $4E OR $4F
F6C9| F003        BEQ    SEX2        ; YES, SKP
F6CB| B9 00C0      LDA    KYBD, Y    ; NO, CONT
F6CE| 88           SEX2    DEY         ; NEXT ADD
F6CF| D0F3        BNE    SEX1

```

10/31/89 9:47

HD:Apple ///:ROM - Sara Tests

Page 5

```

F6D1| AD 51C0          LDA      TXTMD      ; SET TXT
F6D4| B9 00C1          SEX3     LDA      SLT1,Y  ; EXERCISE
F6D7| B9 00C2          LDA      SLT2,Y  ; ALL
F6DA| B9 00C3          LDA      SLT3,Y  ; SLOTS
F6DD| B9 00C4          LDA      SLT4,Y
F6E0| AD FFCF          LDA      EXPROM      ; DISABLE EXPANSION ROM AREA
F6E3| C8              INY
F6E4| D0EE          BNE      SEX3
F6E6|                ;
F6E6|                ; RAM TEST ROUTINE
F6E6|                ;
F6E6| A9 73          USRENTY   LDA      #72+ROM
F6E8| 8D DFFF          STA      SYS1
F6EB| A9 18          LDA      #18
F6ED| 8D D0FF          STA      ZPREG
F6F0| A9 00          LDA      #00
F6F2| A2 07          LDX      #07
F6F4| 95 10          RAMTST0   STA      ZRPG1,X
F6F6| CA              DEX
F6F7| 10FB          BPL      RAMTST0
F6F9| 20 84F7        JSR      RAMSET
F6FC| 08              PHP
F6FD| 20 F6F7        RAMTST1   JSR      RAMWT
F700| 20 F6F7        JSR      RAMWT
F703| 28              PLP
F704| 6A              ROR      A
F705| 08              PHP
F706| 20 A1F7        JSR      PTRINC
F709| D0F2          BNE      RAMTST1
F70B| 20 84F7        JSR      RAMSET
F70E| 08              PHP
F70F| 20 FAF7        RAMTST4   JSR      RAMRD
F712| 48              PHA
F713| A9 00          LDA      #00
F715| 91 18          STA      (PTRLO),Y
F717| 68              PLA
F718| 28              PLP
F719| 6A              ROR      A
F71A| 08              PHP
F71B| 20 A1F7        JSR      PTRINC
F71E| D0EF          BNE      RAMTST4
F720|                ;
F720|                ; RETURN TO START
F720|                ;
F720| A9 00          LDA      #00
F722| 8D EFFF          STA      BNKSW
F725| 8D D0FF          STA      ZPREG
F728| A2 07          LDX      #07
F72A| BD 1018        RAMTST6   LDA      PHPR,X
F72D| 95 10          STA      ZRPG1,X
F72F| CA              DEX
F730| 10F8          BPL      RAMTST6
F732| 20 7EF7        JSR      ERROR
F735| 4C 75F5        JMP      ERRLP
F738|                ;
F738|                ; *****
F738|                ; SARA TEST SUBROUTINES
F738|                ; *****
F738|                ;
F738| BD CDF4          STRWT     LDA      CHPG,X
F73B| 48              PHA
F73C| 09 80          ORA      #80      ; NORMAL VIDEO
F73E| 20 25FC        JSR      COUT      ; & PRINT
F741| E8              INX              ; NXT
F742| 68              PLA              ; CHR
F743| 10F3          BPL      STRWT
F745| 4C 07FD        JMP      CROUT1   ; CLR TO END OF LINE
F748|                ;
F748|                ; SUBROUTINE RAM
F748|                ;
F748| 48              RAM        PHA      ; SV ACC
F749| 8A              TXA      ; CONVRT
F74A| 4A              LSR      A      ; ADD TO
F74B| 4A              LSR      A      ; USE FOR
F74C| 4A              LSR      A      ; 8 ENTRY
F74D| 4A              LSR      A
F74E| 08              PHP
F74F| 4A              LSR      A
F750| 28              PLP
F751| AA              TAX      ; LOOKUP
F752| BD C5F4        LDA      RAMTBL,X ; IF VAL
F755| 1014          BPL      RAM0      ; <0, GET
F757| 48              PHA      ; WHICH
F758| AD EFFF        LDA      BNKSW
F75B| 29 0F          AND      #0F
F75D| AA              TAX
F75E| 68              PLA
F75F| E0 00          CPX      #00
F761| F013          BEQ      RAM1      ; BANK?
F763| 4A              LSR      A      ; SET

```

10/31/89 9:47

HD:Apple ///:ROM - Sara Tests

Page 6

```

F764| 4A          LSR    A          ; PROPER
F765| 4A          LSR    A          ; RAM
F766| CA          DEX     ; VAL
F767| D00D         BNE     RAM1
F769| 29 05        AND     #05        ; CONVERT
F76B| D009         BNE     RAM1        ; TO VAL
F76D| 8A          TXA
F76E| F002        BEQ     RAM00
F770| A9 03        LDA     #03
F772| 9002        BCC     RAM1
F774| 49 03        EOR     #03
F776| 29 07        AND     #07        ; BANKSW
F778| AA          TAX
F779| 68          PLA
F77A| 60          RTS
F77B|              ;
F77B|              ; SUBROUTINE ERROR
F77B|              ;
F77B| 20 38F7      MESSERR JSR     STRWT    ; PRINT MESSAGE FIRST
F77E| A9 F3        ERROR  LDA     #0F2+ROM ; SET 1
F780| 8D DFFF      STA     SYSDB1    ; MHZ MO
F783| 60          RTS
F784|              ;
F784|              ; SUBROUTINE RAMSET
F784|              ;
F784| A2 01        RAMSET LDX     #01
F786| 86 1A        STX     BNK
F788| A0 00        LDY     #00
F78A| A9 AA        LDA     #0AA
F78C| 38          SEC
F78D| 48          RAMSET1 PHA
F78E| 08          PHP
F78F| A5 1A        LDA     BNK
F791| 09 80        ORA     #80
F793| 8D 1914      STA     IBNK
F796| A9 02        LDA     #02
F798| 85 19        STA     PTRHI
F79A| A2 00        LDX     #00
F79C| 86 18        STX     PTRLO
F79E| 28          PLP
F79F| 68          PLA
F7A0| 60          RTS
F7A1|              ;
F7A1|              ; SUBROUTINE PTRINC
F7A1|              ;
F7A1| 48          PTRINC  PHA
F7A2| E6 18        INC     PTRLO
F7A4| D01D         BNE     RETS
F7A6| A5 1A        LDA     BNK
F7A8| 100E        BPL     PINC1
F7AA| A5 19        LDA     PTRHI
F7AC| C9 13        CMP     #13
F7AE| F006        BEQ     PINC2
F7B0| C9 17        CMP     #17
F7B2| D004        BNE     PINC1
F7B4| E6 19        INC     PTRHI
F7B6| E6 19        PINC2  INC     PTRHI
F7B8| E6 19        PINC1  INC     PTRHI
F7BA| D007        BNE     RETS
F7BC| C6 1A        DEC     BNK
F7BE| C6 1A        DEC     BNK
F7C0| 20 8DF7      JSR     RAMSET1
F7C3| 68          RETS    PLA
F7C4| A6 1A        LDX     BNK
F7C6| E0 FD        CPX     #0FD
F7C8| 60          RTS
F7C9|              ;
F7C9|              ; SUBROUTINE RAMERR
F7C9|              ;
F7C9| 48          RAMERR  PHA
F7CA| A6 19        LDX     PTRHI
F7CC| A4 1A        LDY     BNK
F7CE| 3019        BMI     RAMERR4
F7D0| 8A          TXA
F7D1| 301D        BMI     RAMERR5
F7D3| 18          CLC
F7D4| 69 20        ADC     #20
F7D6| 8C EFFF      RAMERR2 STY     BNKSW
F7D9| AA          TAX
F7DA| 20 48F7      RAMERR3 JSR     RAM
F7DD| 68          PLA
F7DE| 48          PHA
F7DF| A0 00        LDY     #00
F7E1| 51 18        EOR     (PTRLO),Y
F7E3| 15 10        ORA     ZRPG1,X
F7E5| 95 10        STA     ZRPG1,X
F7E7| 68          PLA
F7E8| 60          RTS
F7E9| A9 00        RAMERR4 LDA     #00
F7EB| 8D EFFF      STA     BNKSW

```

10/31/89 9:47

HD:Apple ///:ROM - Sara Tests

Page 7

```

F7EE| F0EA          BEQ      RAMERR3
F7F0| 38          RAMERR5 SEC
F7F1| E9 60        SBC      #60
F7F3| C8          INY
F7F4| D0E0        BNE      RAMERR2
F7F6|             ;
F7F6|             ; SUBROUTINE RAMWT
F7F6|             ;
F7F6| 49 FF        RAMWT    EOR      #0FF
F7F8| 91 18        STA      (PTRLO),Y
F7FA| D1 18        RAMRD    CMP      (PTRLO),Y
F7FC| D0CB        BNE      RAMERR
F7FE| 60          RET1     RTS
F7FF|             .END
F7FF|

```

## SYMBOL TABLE DUMP

```

AB - Absolute    LB - Label    UD - Undefined    MC - Macro
RF - Ref         DF - Def       PR - Proc         FC - Func
PB - Public      PV - Private    CS - Consts

```

ACIA	LB F63E	ACIACM	AB C0F2	ACIACN	AB C0F3	ACIAST	AB C0F1	ADCERR	LB F676
ADCTST1	LB F663	ADCTST3	LB F669	ADRS	AB C047	ADTO	AB C066	ATD	LB F653
BLOCKIO	AB F479	BNK	AB 001A	BNKSW	AB FFEF	BOOT	LB F6A1	CHPG	LB F4CD
CLDSRT	AB FD98	CNTWR	LB F532	COUT	AB FC25	CROUT1	AB FD07	CV	AB 005D
DISK1	LB F513	DISKOFF	AB C0D0	ERRLP	LB F575	ERRLP1	LB F593	ERROR	LB F77E
EXPROM	AB CFFF	GOBOOT	LB F6BF	GRMD	AB C050	IBBUFP	AB 0085	IBCMD	AB 0087
IBNK	AB 1419	KBDSTRB	AB C010	KEYBD	AB C008	KEYIN	AB FD0F	KEYPLUG	LB F67B
KYBD	AB C000	MESSERR	LB F77B	MONITOR	AB F901	NMEM1	LB F54F	NMEM2	LB F562
NOGOOD	LB F52A	NOMEM	LB F548	NXBIT	LB F526	NXBYT	LB F524	PDLEN	AB C058
PHPR	AB 1810	PINC1	LB F7B8	PINC2	LB F7B6	PREVTRK	AB 0091	PTRHI	AB 0019
PTRINC	LB F7A1	PTRLO	AB 0018	PULBT	LB F53A	RAM	LB F748	RAM0	LB F76B
RAM00	LB F772	RAM1	LB F776	RAMERR	LB F7C9	RAMERR2	LB F7D6	RAMERR3	LB F7DA
RAMERR4	LB F7E9	RAMERR5	LB F7F0	RAMRD	LB F7FA	RAMSET	LB F784	RAMSET1	LB F78D
RAMTBL	LB F4C5	RAMTST0	LB F6F4	RAMTST1	LB F6FD	RAMTST4	LB F70F	RAMTST6	LB F72A
RAMWT	LB F7F6	RAMWT1	LB F5A1	RAMWT2	LB F5A5	RAMWT4	LB F5AD	RECON	LB F686
RET1	LB F7FE	RETS	LB F7C3	ROM	AB 0001	ROMTST	LB F5E7	ROMTST1	LB F5F2
ROMTST2	LB F5FE	SARATEST	PR ----	SETCVH	AB FBC7	SETUP	AB FD9D	SEX	LB F6C2
SEX1	LB F6C4	SEX2	LB F6CE	SEX3	LB F6D4	SLT1	AB C100	SLT2	AB C200
SLT3	AB C300	SLT4	AB C400	STK0	AB 00FF	STRWT	LB F738	SYS01	AB FFD0
SYS02	AB FFD2	SYS03	AB FFD3	SYSE0	AB FFE0	SYSE2	AB FFE2	SYSE3	AB FFE3
TXTMD	AB C051	USRENTY	LB F6E6	VIATST	LB F60D	ZP1	LB F5BE	ZP2	LB F5D4
ZP3	LB F5E2	ZPREG	AB FFD0	ZRPG	AB 0000	ZRPG1	AB 0010		

Assembly complete: 545 lines  
 0 Errors flagged on this Assembly

## 6502 OPCODE STATIC FREQUENCIES

```

ADC : 10 | *****
AND : 12 | *****
ASL : 3  | ***
BCC : 3  | ***
BCS : 1  | *
BEQ : 12 | *****
BIT : 1  | *
BMI : 4  | ****
BNE : 31 | *****
BPL : 9  | *****
CLC : 3  | ***
CLD : 1  | *
CMP : 10 | *****
CPX : 5  | *****
CPY : 2  | **
DEC : 3  | ***
DEX : 9  | *****
DEY : 5  | *****
EOR : 5  | *****
INC : 6  | *****
INX : 6  | *****
INY : 6  | *****
JMP : 4  | ****
JSR : 29 | *****
LDA : 56 | M *****
LDX : 24 | *****
LDY : 10 | *****
LSR : 9  | *****
ORA : 3  | ***
PHA : 11 | *****
PHP : 6  | *****
PLA : 12 | *****
PLP : 4  | ****
ROR : 2  | **
RTS : 6  | *****
SBC : 1  | m *

```



10/31/89 9:47

HD:Apple ///:ROM - Sara Tests

Page 8

```
SEC :    2 | **
STA :   30 | *****
STX :   18 | *****
STY :    4 | ****
TAX :    4 | ****
TAY :    2 | **
TXA :    6 | *****
TXS :    2 | **
TYA :    4 | ****
```

```
Minimum frequency =    1
Maximum frequency =   56
```

```
Average frequency =    8
```

```
Unused opcodes:
```

```
BRK  BVC  BVS  CLI  CLV  NOP  ROL  RTI  SED  SEI  TSX
```

```
Program opcode usage: 80 %
```

-----  
(1.00) That's all, Folks ...  
-----

Source Code Listing  
for

**Apple ///**

**ROM - Monitor**

David T. Craig  
736 Edgewater  
Wichita, Kansas 67230

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 1

```

0000| ;*****
0000| ; APPLE /// ROM - MONITOR
0000| ; COPYRIGHT 1979 BY APPLE COMPUTER, INC.
0000| ;*****
0000|
0000| .ABSOLUTE
0000| .PROC MONITOR
0000| .ORG 0F7FE
F7FE| ;
F7FE| ;
F7FE| 60 RET1 RTS
F7FF| 3F .BYTE 03F
F800| E9 01 SBC #01
F802| F0FA BEQ RET1
F804| E9 01 SBC #01
F806| F0F6 BEQ RET1
F808| E9 01 SBC #01
F80A| F0F2 BEQ RET1
F80C| E9 01 SBC #01
F80E| F0EE BEQ RET1
F810| E9 01 SBC #01
F812| F0EA BEQ RET1
F814| E9 01 SBC #01
F816| F0E6 BEQ RET1
F818| E9 01 SBC #01
F81A| F0E2 BEQ RET1
F81C| E9 01 SBC #01
F81E| F0DE BEQ RET1
F820| E9 01 SBC #01
F822| F0DA BEQ RET1
F824| E9 01 SBC #01
F826| F0D6 BEQ RET1
F828| E9 01 SBC #01
F82A| F0D2 BEQ RET1
F82C| E9 01 SBC #01
F82E| F0CE BEQ RET1
F830| E9 01 SBC #01
F832| F0CA BEQ RET1
F834| E9 01 SBC #01
F836| F0C6 BEQ RET1
F838| E9 01 SBC #01
F83A| F0C2 BEQ RET1
F83C| E9 01 SBC #01
F83E| F0BE BEQ RET1
F840| E9 01 SBC #01
F842| F0BA BEQ RET1
F844| E9 01 SBC #01
F846| F0B6 BEQ RET1
F848| E9 01 SBC #01
F84A| F0B2 BEQ RET1
F84C| E9 01 SBC #01
F84E| F0AE BEQ RET1
F850| E9 01 SBC #01
F852| F0AA BEQ RET1
F854| E9 01 SBC #01
F856| F0A6 BEQ RET1
F858| E9 01 SBC #01
F85A| F0A2 BEQ RET1
F85C| E9 01 SBC #01
F85E| F09E BEQ RET1
F860| E9 01 SBC #01
F862| F09A BEQ RET1
F864| E9 01 SBC #01
F866| F096 BEQ RET1
F868| E9 01 SBC #01
F86A| F092 BEQ RET1
F86C| E9 01 SBC #01
F86E| F08E BEQ RET1
F870| E9 01 SBC #01
F872| F08A BEQ RET1
F874| E9 01 SBC #01
F876| F086 BEQ RET1
F878| E9 01 SBC #01
F87A| F082 BEQ RET1
F87C| E9 01 SBC #01
F87E| F002 BEQ RET3
F880| E9 01 SBC #01
F882| F07C BEQ RET2
F884| E9 01 SBC #01
F886| F078 BEQ RET2
F888| E9 01 SBC #01
F88A| F074 BEQ RET2
F88C| E9 01 SBC #01
F88E| F070 BEQ RET2
F890| E9 01 SBC #01
F892| F06C BEQ RET2
F894| E9 01 SBC #01
F896| F068 BEQ RET2
F898| E9 01 SBC #01
F89A| F064 BEQ RET2

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 2

F89C	E9 01	SBC	#01
F89E	F060	BEQ	RET2
F8A0	E9 01	SBC	#01
F8A2	F05C	BEQ	RET2
F8A4	E9 01	SBC	#01
F8A6	F058	BEQ	RET2
F8A8	E9 01	SBC	#01
F8AA	F054	BEQ	RET2
F8AC	E9 01	SBC	#01
F8AE	F050	BEQ	RET2
F8B0	E9 01	SBC	#01
F8B2	F04C	BEQ	RET2
F8B4	E9 01	SBC	#01
F8B6	F048	BEQ	RET2
F8B8	E9 01	SBC	#01
F8BA	F044	BEQ	RET2
F8BC	E9 01	SBC	#01
F8BE	F040	BEQ	RET2
F8C0	E9 01	SBC	#01
F8C2	F03C	BEQ	RET2
F8C4	E9 01	SBC	#01
F8C6	F038	BEQ	RET2
F8C8	E9 01	SBC	#01
F8CA	F034	BEQ	RET2
F8CC	E9 01	SBC	#01
F8CE	F030	BEQ	RET2
F8D0	E9 01	SBC	#01
F8D2	F02C	BEQ	RET2
F8D4	E9 01	SBC	#01
F8D6	F028	BEQ	RET2
F8D8	E9 01	SBC	#01
F8DA	F024	BEQ	RET2
F8DC	E9 01	SBC	#01
F8DE	F020	BEQ	RET2
F8E0	E9 01	SBC	#01
F8E2	F01C	BEQ	RET2
F8E4	E9 01	SBC	#01
F8E6	F018	BEQ	RET2
F8E8	E9 01	SBC	#01
F8EA	F014	BEQ	RET2
F8EC	E9 01	SBC	#01
F8EE	F010	BEQ	RET2
F8F0	E9 01	SBC	#01
F8F2	F00C	BEQ	RET2
F8F4	E9 01	SBC	#01
F8F6	F008	BEQ	RET2
F8F8	E9 01	SBC	#01
F8FA	F004	BEQ	RET2
F8FC	E9 01	SBC	#01
F8FE	F000	BEQ	RET2
F900	60	RET2	RTS
F901		;	
F901		;	
F901	0058	SCRNLOC	.EQU 58
F901		;	
F901	0058	LMARGIN	.EQU SCRNLLOC
F901	0059	RMARGIN	.EQU SCRNLLOC+1
F901	005A	WINTOP	.EQU SCRNLLOC+2
F901	005B	WINBTM	.EQU SCRNLLOC+3
F901	005C	CH	.EQU SCRNLLOC+4
F901	005D	CV	.EQU SCRNLLOC+5
F901	005E	BAS4L	.EQU SCRNLLOC+6
F901	005F	BAS4H	.EQU SCRNLLOC+7
F901	0060	BAS8L	.EQU SCRNLLOC+8
F901	0061	BAS8H	.EQU SCRNLLOC+9
F901	0058	TBAS4L	.EQU SCRNLLOC+A
F901	0063	TBAS4H	.EQU SCRNLLOC+0B
F901	0064	TBAS8L	.EQU SCRNLLOC+0C
F901	0065	TBAS8H	.EQU SCRNLLOC+0D
F901	0066	FORGND	.EQU SCRNLLOC+0E
F901	0067	BKGND	.EQU SCRNLLOC+0F
F901	0068	MODES	.EQU SCRNLLOC+10
F901	0069	CURSOR	.EQU SCRNLLOC+11
F901	006A	STACK	.EQU SCRNLLOC+12
F901	006B	PROMPT	.EQU SCRNLLOC+13
F901	006C	TEMPX	.EQU SCRNLLOC+14
F901	006D	TEMPY	.EQU SCRNLLOC+15
F901	006E	CSWL	.EQU SCRNLLOC+16
F901	006F	CSWH	.EQU SCRNLLOC+17
F901	0070	KSWL	.EQU SCRNLLOC+18
F901	0071	KSWH	.EQU SCRNLLOC+19
F901	0072	PCL	.EQU SCRNLLOC+1A
F901	0073	PCH	.EQU SCRNLLOC+1B
F901	0074	A1L	.EQU SCRNLLOC+1C
F901	0075	A1H	.EQU A1L+1
F901	0076	A2L	.EQU A1L+2
F901	0077	A2H	.EQU A1L+3
F901	0078	A3L	.EQU A1L+4
F901	0079	A3H	.EQU A1L+5
F901	007A	A4L	.EQU A1L+6

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 3

```

F901| 007B      A4H      .EQU  A1L+7
F901| 007C      STATE    .EQU  A1L+8
F901| 007D      YSAV      .EQU  A1L+9
F901| 007E      INBUF     .EQU  A1L+0A
F901| 0080      TEMP      .EQU  A1L+0C
F901| 0069      MASK      .EQU  CURSOR
F901|           ;
F901| C000      KBD        .EQU  0C000
F901| C010      KBDSTRB    .EQU  0C010
F901|           ;
F901| 0358      USERADR     .EQU  358
F901| F479      BLOCKIO     .EQU  0F479
F901| F686      RECON       .EQU  0F686      ; AS OF 12/20/1979
F901| F4EE      DIAGN       .EQU  0F4EE
F901| 0050      INBUFLN     .EQU  50      ; ONLY 80 BYTES ($3A0-$3EF)
F901| 0081      IBSLOT      .EQU  81
F901| 0082      IBDRVN      .EQU  IBSLOT+1
F901| 0085      IBBUFP      .EQU  IBSLOT+4
F901| 0087      IBCMD       .EQU  IBSLOT+6
F901|           ;
F901| F901      ENTRY       .EQU  *
F901| BA         TSX
F902| 86 6A      STX        STACK
F904| D8         MON        CLD      ; MUST BE HEX MODE
F905| 20 4EFC     JSR        BELL
F908| A6 6A      MONZ       LDX      STACK ; RESTORE STACK TO ORIGINAL LOCATION
F90A| 9A         TXS
F90B| A9 DF      LDA        #0DF     ; PROMPT (APPLE) FOR SARA MONITOR
F90D| 85 6B      STA        PROMPT
F90F| 20 D5FC     JSR        GETLNZ  ; GET A LINE OF INPUT
F912| 20 67F9     JSR        ZSTATE   ; SET REGULAR SCAN
F915| 20 2CF9     JSR        GETNUM   ; ATTEMPT TO READ HEX BYTE
F918| 84 7D      STY        YSAV      ; STORE CURRENT INPUT POINTER
F91A| A0 12      LDY        #12      ; 18 COMMANDS
F91C| 88         CMDSRCH    DEY
F91D| 30E5      BMI        MON        ; GIVE UP IF UNRECOGNIZABLE
F91F| D9 6CF9     CMP        CMDTAB,Y ; FOUND?
F922| D0F8      BNE        CMDSRCH    ; NO KEEP LOOKING
F924| 20 5EF9     JSR        TOSUB    ; PERFORM FUNCTION
F927| A4 7D      LDY        YSAV      ; GET NEXT POINTER
F929| 4C 15F9     JMP        NXTINP    ; DO NEXT COMMAND
F92C|           ;
F92C| A2 00      GETNUM     LDX        #00      ; CLEAR A2
F92E| 86 76      STX        A2L
F930| 86 77      STX        A2H
F932| B1 7E      NXTCHR    LDA        (INBUF),Y
F934| C8         INY
F935| 49 B0      EOR        #0B0
F937| C9 0A      CMP        #0A      ; TEST FOR DIGIT
F939| 9006      BCC        DIGIT     ; SAVE IT IF 1-9
F93B| 69 88      ADC        #88      ; TEST FOR HEX A-F
F93D| C9 FA      CMP        #0FA
F93F| 902A      BCC        DIGRET
F941| A2 03      DIGIT     LDX        #03
F943| 0A         ASL        A
F944| 0A         ASL        A
F945| 0A         ASL        A
F946| 0A         ASL        A
F947| 0A         ASL        A
F948| 26 76      NXTBIT    ROL        A2L ; SHIFT HEX DIGITS INTO A2
F94A| 26 77      ROL        A2H
F94C| CA         DEX
F94D| 10F8      BPL        NXTBIT    ; SHIFTED ALL YET?
F94F| A5 7C      NXTBAS     LDA        STATE
F951| D006      BNE        NXTBS2    ; IF ZERO THEN COPY TO A1,3
F953| B5 77      LDA        A2H,X
F955| 95 75      STA        A1H,X
F957| 95 79      STA        A3H,X
F959| E8         NXTBS2    INX
F95A| F0F3      BEQ        NXTBAS
F95C| D0D4      BNE        NXTCHR
F95E|           ; SWITCH ROUTINE FOR CHARACTER
F95E|           ;
F95E| A9 FA      TOSUB      LDA        #0FA    ; PUSH ADDRESS OR FUNCTION
F960| 48         PHA
F961| B9 7DF9     LDA        CMDVEC,Y ; AND RETURN IT
F964| 48         PHA
F965| A5 7C      LDA        STATE ; PASS MODE VIA ACC.
F967| A0 00      ZSTATE     LDY        #00
F969| 84 7C      STY        STATE ; RESET STATE OF SCAN
F96B| 60         DIGRET     RTS
F96C| F96C      CMDTAB      .EQU  *
F96C| 00          .BYTE      00      ; G =GP (CALL) SUBROUTINE
F96D| 03          .BYTE      03      ; J =JUMP (CONT) PROGRAM
F96E| 06          .BYTE      06      ; M =MOVE MEMORY
F96F| EB          .BYTE      0EB     ; R =READ DISK BLOCK
F970| EC          .BYTE      0EC     ; S =MEMORY SEARCH
F971| EE          .BYTE      0EE     ; U =USER FUNCTION
F972| EF          .BYTE      0EF     ; V =VERIFY MEMORY BLOCKS

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 4

```

F973| F0      .BYTE 0F0      ; W  =WRITE DISK BLOCK
F974| F1      .BYTE 0F1      ; X  =REPEAT COMMAND LINE
F975| 99      .BYTE 99       ; SP =SPACE (BYTE SEPARATOR)
F976| 9B      .BYTE 9B       ; "  =ASCII (HI BIT ON)
F977| A0      .BYTE 0A0      ; '  =ASCII (HI BIT OFF)
F978| 93      .BYTE 93       ; :  =SET STORE MODE
F979| A7      .BYTE 0A7      ; .  =RANGE SEPARATOR
F97A| A8      .BYTE 0A8      ; /  =COMMAND SEPARATOR
F97B| 95      .BYTE 95       ; <  =DEST/SOURCE SEPARATOR
F97C| C6      .BYTE 0C6      ; CR =CARRIAGE RETURN
F97D|          ;
F97D| F97D    CMDVEC      .EQU *
F97D| 90      .BYTE 90       ; GO-1
F97E| 8E      .BYTE 8E       ; JUMP-1
F97F| 3F      .BYTE 3F       ; MOVE-1
F980| D3      .BYTE 0D3      ; READ-1
F981| 08      .BYTE 08       ; SEARCH-1
F982| 8B      .BYTE 8B       ; USER-1
F983| 4E      .BYTE 4E       ; VRFY-1
F984| D6      .BYTE 0D6      ; WRTE-1
F985| 2C      .BYTE 2C       ; REPEAT-1
F986| B7      .BYTE 0B7      ; SPCE-1
F987| 1A      .BYTE 1A       ; ASCII-1
F988| 1C      .BYTE 1C       ; ASCII0-1
F989| CB      .BYTE 0CB      ; SETMODE-1
F98A| CB      .BYTE 0CB      ; SETMODE-1
F98B| AD      .BYTE 0AD      ; SEP-1
F98C| A4      .BYTE 0A4      ; DEST-1
F98D| 39      .BYTE 39       ; CRMON-1
F98E|          ;
F98E|          ;
F98E| E6 7A    NXTA4      INC  A4L      ; BUMP 16 BIT POINTERS
F990| D002    BNE  NXTA1
F992| E6 7B    INC  A4H
F994| E6 74    NXTA1      INC  A1L      ; BUMP A1
F996| D005    BNE  TSTA1
F998| E6 75    INC  A1H
F99A| 38      SEC
F99B| F010    BEQ  RETA1      ; IN CASE OF ROLL OVER
F99D| A5 74    TSTA1      LDA  A1L
F99F| 38      SEC
F9A0| E5 76    SBC  A2L
F9A2| 85 80    STA  TEMP
F9A4| A5 75    LDA  A1H
F9A6| E5 77    SBC  A2H
F9A8| 05 80    ORA  TEMP
F9AA| D001    BNE  RETA1      ; IF A1 LESS THAN OR EQUAL TO A2
F9AC| 18      CLC
F9AD| 60      RETA1      RTS      ; THEN CARRY CLEAR ON RETURN
F9AE|          ;
F9AE|          ;
F9AE| 48      PRBYTE      PHA      ; SAVE LOW NIBBLE
F9AF| 4A      LSR  A
F9B0| 4A      LSR  A      ; SHIFT HI NIBBLE TO PRINT.
F9B1| 4A      LSR  A
F9B2| 4A      LSR  A
F9B3| 20 B9F9 JSR  PRHEXZ
F9B6| 68      PLA
F9B7| 29 0F    PRHEX      AND  #0F      ; STRIP HI NIBBLE
F9B9| 09 B0    PRHEXZ     ORA  #0B0      ; MAKE IT NUMERIC
F9BB| C9 BA    CMP  #0BA      ; IS IT '>'9'
F9BD| 9002    BCC  PRHEX2
F9BF| 69 06    ADC  #06      ; MAKE IT 'A'-'F'
F9C1| 4C 39FC PRHEX2     JMP  COUT
F9C4|          ;
F9C4| 20 AEF9 PRBYCOL     JSR  PRBYTE
F9C7|          ;
F9C7| A9 BA    PRCOLON     LDA  #0BA      ; PRINT A COLON
F9C9| D0F6    BNE  PRHEX2      ; BRANCH ALWAYS
F9CB|          ;
F9CB| A9 07    TST80WID    LDA  #07      ; ANTICIPATE
F9CD| 24 68    BIT  MODES      ; TEST FOR 80
F9CF| 5002    BVC  SVMASK
F9D1| A9 0F    LDA  #0F
F9D3| 85 69    SVMASK     STA  MASK
F9D5| 60      RTS
F9D6|          ;
F9D6| 8A      A1PC      TXA      ; TEST FOR NEW PC
F9D7| F007    BEQ  OLDPC
F9D9| B5 74    A1PC1      LDA  A1L,X
F9DB| 95 72    STA  PCL,X
F9DD| CA      DEX
F9DE| 10F9    BPL  A1PC1
F9E0| 60      OLDPC      RTS
F9E1|          ;
F9E1| 85 69    ASCII1     STA  MASK      ; SAVE HI BIT STATUS
F9E3| A4 7D    ASCII2     LDY  YSAV      ; MOVE ASCII TO MEMORY
F9E5| B1 7E    LDA  (INBUF),Y
F9E7| E6 7D    INC  YSAV      ; BUMP FOR NEXT THING.
F9E9| A0 00    LDY  #00

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 5

```

F9EB| C9 A2          CMP    #0A2      ; ASCII " ?
F9ED| D005          BNE    ASCII3   ; NOPE, CONTINUE.
F9EF| A5 69          LDA          MASK
F9F1| 1032          BPL     BITON    ; HE'S CHANGED MODES.
F9F3| 60             RTS
F9F4| C9 A7          ASCII3  CMP    #0A7      ; ASCII ' ?
F9F6| D005          BNE    CRCHK    ; NO, TEST FOR EOL.
F9F8| A5 69          LDA          MASK
F9FA| 302D          BMI    BITOFF   ; CHANGE MODES.
F9FC| 60             RTS
F9FD|
F9FD| C9 8D          CRCHK  CMP    #8D      ; END OF LINE?
F9FF| F007          BEQ     ASCDONE  ; YES, FINISHED
FA01| 25 69          AND     MASK
FA03| 20 C3FA        JSR     STOR1    ; GO STORE IT!
FA06| D0DB          BNE    ASCII2    ; DO NEXT.
FA08| 60             RTS
FA09|
FA09|
FA09| B1 74          SEARCH  LDA    (A1L),Y ; LOAD SEARCH BYTE
FA0B| C5 7A          CMP     A4L
FA0D| D006          BNE    SRCH1
FA0F| 20 75FA        JSR     PRINTA1 ; DUMP MEMORY
FA12| 20 EFFC        JSR     CROUT
FA15| 20 94F9        SRCH1  JSR     NXTA1 ; INCREMENT POINTER
FA18| 90EF          BCC     SEARCH   ; CONTINUE SEARCH
FA1A| 60             RTS          ; RETURN
FA1B|
FA1B|
FA1B| 38           ASCII  SEC          ; INDICATE HI ON.
FA1C| 90           .BYTE  90          ; (BCC - NEVER TAKEN)
FA1D| 18           ASCII0 CLC          ; INDICATE HI OFF
FA1E| AA           CKMDE  TAX          ; SAVE STATE
FA1F| 86 7C          STX     STATE     ; RETAIN STATE
FA21| 49 BA          EOR     #0BA     ; ARE WE IN STORE MODE?
FA23| D07D          BNE    ERROR
FA25| A9 FF          BITON  LDA    #0FF ; SET HI BIT UNMASKED
FA27| B0B8          BCS     ASCII1
FA29| A9 7F          BITOFF LDA    #7F ; MASK HI BIT
FA2B| 10B4          BPL     ASCII1   ; ALWAYS BRANCHES
FA2D| 2C 00C0        REPEAT BIT     KBD ; REPEAT UNTIL KEYPRESS
FA30| 1003          BPL     REPEAT1
FA32| 4C 0FFD        JMP     KEYIN
FA35| 68           REPEAT1 PLA
FA36| 68           LFA36  PLA
FA37| 4C 12F9        JMP     SCAN
FA3A|
FA3A|
FA3A| 20 B4FA        CRMON  JSR     BL1
FA3D| 4C 08F9        JMP     MONZ
FA40|
FA40|
FA40| 20 9DF9        MOVE   JSR     TSTA1 ; TEST VALID RANGE
FA43| B05D          BCS     ERROR
FA45| B1 74          MOVNXT LDA    (A1L),Y ; COMPARE BYTE FOR BYTE
FA47| 91 7A          STA    (A4L),Y
FA49| 20 8EF9        JSR     NXTA4 ; BUMP BOTH A1 AND A4
FA4C| 90F7          BCC     MOVNXT
FA4E| 60             RTS          ; ALL DONE WITH MOVE
FA4F|
FA4F|
FA4F| 20 9DF9        VRFY   JSR     TSTA1 ; TEST VALID RANGE
FA52| B04E          BCS     ERROR
FA54| B1 74          VRFY1  LDA    (A1L),Y ; COMPARE BYTE FOR BYTE
FA56| D1 7A          CMP     (A4L),Y ; MATCH?
FA58| F006          BEQ     VRFY2    ; YES, DO NEXT.
FA5A| 20 66FA        JSR     MISMATCH ; PRINT BOTH BYTES
FA5D| 20 EFFC        JSR     CROUT    ; GOTO NEWLINE
FA60| 20 8EF9        VRFY2  JSR     NXTA4 ; BUMP BOTH A1 AND A4
FA63| 90EF          BCC     VRFY1
FA65| 60             RTS          ; VERIFY DONE.
FA66|
FA66|
FA66| A5 7B          MISMATCH LDA    A4H ; PRINT ADDRESS OF A4
FA68| 20 AEF9        JSR     PRBYTE
FA6B| A5 7A          LDA     A4L
FA6D| 20 C4F9        JSR     PRBYCOL ; OUTPUT A COLON FOR SEPARATOR
FA70| B1 7A          LDA     (A4L),Y ; AND THE DATA
FA72| 20 84FA        JSR     PRBYTSP ; PRINT THE BYTE AND A SPACE
FA75| 20 87FA        PRINTA1 JSR     PRSPC ; LEAD WITH A SPACE
FA78| A5 75          LDA     A1H ; OUTPUT ADDRESS A1
FA7A| 20 AEF9        JSR     PRBYTE
FA7D| A5 74          LDA     A1L
FA7F| 20 C4F9        JSR     PRBYCOL ; SEPARATE WITH A COLON
FA82| B1 74          PRA1BYTE LDA    (A1L),Y ; PRINT BYTE POINTED TO BY A1
FA84| 20 AEF9        PRBYTSP JSR     PRBYTE
FA87| A9 A0          PRSPC  LDA     #0A0 ; PRINT A SPACE
FA89| 4C 39FC        JMP     COUT     ; END VIA OUTPUT ROUTINE.
FA8C|
FA8C|
FA8C| 4C 5803        USER   JMP     USERADR
FA8F|

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 6

```

FA8F| 68          JUMP    PLA          ; LEAVE STACK WITH NOTHIN' ON IT.
FA90| 68          PLA          ; STUFF PROGRAM COUNTER
FA91| 20 D6F9     GO      JSR      A1PC    ; JUMP TO USER PROG.
FA94| 6C 7200     JMP      @PCL
FA97|             ;
FA97| FA97       ; RWERROR .EQU      *      ; PRINT ERROR NUMBER
FA97| 20 AEF9     JSR      PRBYTE    ; PRINT THE OFFENDER
FA9A| A9 A1      LDA      #0A1     ; FOLLOWED BY A "!"
FA9C| 20 39FC     JSR      COUT      ;
FA9F| 20 07FD     JSR      NOSTOP    ; OUTPUT A CARRIAGE RETURN (NO STOPLST)
FAA2| 4C 04F9     JMP      MON
FAA5|             ;
FAA5| A5 76      DEST    LDA      A2L     ; COPY A2 TO A4 FOR DESTINATION OP
FAA7| 85 7A      STA      A4L
FAA9| A5 77      LDA      A2H
FAAB| 85 7B      STA      A4H
FAAD| 60        RTS
FAAE|             ;
FAAE| 20 B8FA     SEP      JSR      SPCE    ; SEPARATOR TEST STORE MODE OR DUMP.
FAB1| 98        TYA          ; ZERO MODE.
FAB2| F01D      BEQ      SETMDZ    ; BRANCH ALWAYS
FAB4|             ;
FAB4| C6 7D      BL1      DEC      YSAV    ; TEST FOR NO LINE
FAB6| F045      BEQ      DUMP8      ; IF NO LINE, GIVEN A ROW OF BYTES
FAB8| CA        SPCE     DEX          ; TEST IF AFTER ANOTHER SPACE
FAB9| D016      BNE      SETMDZ
FABB| C9 BA     CMP      #0BA      ; STORE MODE?
FABD| D04B      BNE      TSTDUMP
FABF| 85 7C      STOR     STA      STATE  ; KEEP IT IN STORE STATE
FAC1| A5 76      LDA      A2L      ; GET BYTE TO BE STORED
FAC3| 91 78      STOR1    STA      (A3L),Y ; PUT IT IN MEMORY.
FAC5| E6 78      INC      A3L      ; BUMP POINTER
FAC7| D002      BNE      DUMMY
FAC9| E6 79      INC      A3H
FACB| 60        DUMMY    RTS          ; ALSO USED FOR '/' TO CLEAR MODE
FACC|             ;
FACC| A4 7D      SETMODE LDY      YSAV    ; USE INPUT CHARACTER
FACE| 88        DEY
FACF| B1 7E      LDA      (INBUF),Y ; TO SET MODE
FAD1| 85 7C      SETMDZ   STA      STATE
FAD3| 60        RTS
FAD4|             ;
FAD4| A9 01      READ     LDA      #01     ; GET DISK COMMAND TO READ
FAD6| 2C        .BYTE    2C          ; DUMMY BIT TO SKIP 2 BYTES
FAD7| A9 02      WRTE     LDA      #02     ; SET DISK COMMAND TO WRITE
FAD9| 85 87      SAVCMD   STA      IBCMD
FADB| A5 74      RWLOOP  LDA      A1L
FADD| 85 85      STA      IBBUFF    ; COMMAND FORMAT IS
FADF| A5 75      LDA      A1H      ; BLOCKNUMBER <ADDRESS END ADDRESS
FAE1| 85 86      STA      IBBUFF+1
FAE3| A6 7B      LDX      A4H      ; SEND BLOCK NUMBER VIA X & A
FAE5| A5 7A      LDA      A4L
FAE7| 78        SEI          ; NO INTERRUPTS WHILE IN MONITOR
FAE8| 20 79F4     JSR      BLOCKIO   ; DO DISKO FEVER
FAEB| B0AA      BCS      RWERROR    ; GIVE UP IF ERROR ENCOUNTERED
FAED| E6 7A      INC      A4L      ; BUMP BLOCK NUMBER
FAEF| D002      BNE      NOVER
FAF1| E6 7B      INC      A4H
FAF3| E6 75      NOVER   INC      A1H  ; BUMP RAM ADDRESS BY 512 BYTES
FAF5| E6 75      INC      A1H
FAF7| 20 9DF9     JSR      TSTA1     ; TEST FOR FINISHED
FAFA| 90DF      BCC      RWLOOP    ; NOT DONE, DO NEXT BLOCK
FAFC| 60        RTS
FAFD|             ;
FAFD| A5 75      DUMP8    LDA      A1H
FAFF| 85 77      STA      A2H
FB01| 20 CBF9     JSR      TST80WID  ; GET WIDTH MASK INTO ACC
FB04| 05 74      ORA      A1L
FB06| 85 76      STA      A2L
FB08| D006      BNE      DUMP0      ; BRANCH ALWAYS
FB0A|             ;
FB0A| 4A        TSTDUMP  LSR      A      ; DUMP?
FB0B| B095      ERROR1   BCS      ERROR
FB0D| 20 CBF9     DUMP    JSR      TST80WID ; SET FOR EITHER 80 OR 40 COLUMNS
FB10| A5 74      DUMP0    LDA      A1L
FB12| 85 7A      STA      A4L
FB14| A5 75      LDA      A1H
FB16| 85 7B      STA      A4H
FB18| 20 9DF9     JSR      TSTA1     ; TEST FOR VALID RANGE
FB1B| B0EE      BCS      ERROR1
FB1D| 20 75FA     DUMP1   JSR      PRINT1  ; PRINT ADDRESS AND FIRST BYTE
FB20| 20 94F9     DUMP2   JSR      NXTA1
FB23| B010      BCS      DUMPASC    ; END WITH ASCII
FB25| A5 74      LDA      A1L      ; TEST END OF LINE
FB27| 25 69      AND      MASK      ; FOR 40/80 COLUMN
FB29| D005      BNE      DUMP3
FB2B| 20 35FB     JSR      DUMPASC
FB2E| D0ED      BNE      DUMP1      ; BRANCH ALWAYS
FB30| 20 82FA     DUMP3   JSR      PRA1BYTE ; GO PRINT NEXT BYTE AND A SPACE
FB33| D0EB      BNE      DUMP2      ; ALWAYS (ACC JUST PULLED AS $A0)

```



10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 7

```

FB35|                                     ;
FB35| A5 7A                               DUMPASC LDA A4L           ; RESET TO BEGINNING OF LINE
FB37| 85 74                               STA A1L
FB39| A5 7B                               LDA A4H
FB3B| 85 75                               STA A1H
FB3D| 20 87FA                             JSR PRSPC           ; PRINT AN EXTRA SPACE
FB40| A0 00                               ASC1 LDY #00        ; TO INDEX MEMORY INDIRECT
FB42| B1 74                               LDA (A1L),Y
FB44| 09 80                               ORA #80           ; SET NORMAL VIDEO
FB46| C9 A0                               CMP #0A0         ; TEST FOR CONTROL CHARACTERS
FB48| B002                               BCS ASC2         ; OK TO PRINT NON CONTROLS
FB4A| A9 AE                               LDA #0AE         ; OTHERWISE PRINT A SPACE
FB4C| 20 39FC                             ASC2 JSR COUT      ; PUT IT OUT
FB4F| 20 8EF9                             JSR NXTA4        ; BUMP BOTH A1 AND A4
FB52| B006                               BCS ASC3         ; FINISHED
FB54| A5 74                               LDA A1L          ; TEST END OF LINE
FB56| 25 69                               AND MASK
FB58| D0E6                               BNE ASC1         ; NOT DONE, PRINT NEXT
FB5A| 4C EFFC                             ASC3 JMP CROUT
FB5D|                                     ;
FB5D| 38                               COL80 SEC          ; INDICATE 80 COLUMNS
FB5E| AD 53C0                             LDA #0C053       ; GOTO 80 COLUMN MODE
FB61| B004                               BCS SET80        ; BRANCH ALWAYS
FB63|                                     ;
FB63| 18                               COL40 CLC          ; INDICATE 40 COLUMNS DESIRED
FB64| AD 52C0                             LDA #0C052       ; GOTO 40 COLUMN MODE
FB67| A5 68                               SET80 LDA MODES
FB69| 09 40                               ORA #40          ; ASSUME 80
FB6B| B002                               BCS SET80A       ; AND BRANCH IF IT IS
FB6D| 29 BF                               AND #0BF         ; BUT FIX FOR 40 IF NOT
FB6F| 85 68                               SET80A STA MODES
FB71| 09 7F                               ORA #7F          ; ISOLATE BIT 7
FB73| 29 A0                               AND #0A0         ; (BIT 7 SETS NORMAL/INVERSE)
FB75| 85 66                               STA FORGND
FB77| B002                               BCS SET80B       ; AGAIN ASSUMES 80 COLUMNS
FB79| A9 F0                               LDA #0F0         ; IF NOT, SET FOR/BACKGROUND COLOR
FB7B| 85 67                               SET80B STA BKGND
FB7D|                                     ;
FB7D| A5 58                               CLSCRN LDA LMARGIN ; SET CURSOR TO TOP LEFT OF WINDOW
FB7F| 85 5C                               STA CH
FB81| A5 5A                               LDA WINTOP
FB83| 85 5D                               STA CV           ; NOW DROP INTO CLEAR END OF PAGE
FB85|                                     ;
FB85| A5 5C                               CLEOP LDA CH      ; SAVE CURRENT CURSOR POSITION
FB87| 48                               PHA
FB88| A5 5D                               LDA CV
FB8A| 48                               PHA
FB8B| 20 C5FB                             JSR SETCV
FB8E| 20 A2FB                             CLEOP1 JSR CLEOL   ; CLEAR TO END OF FIRST LINE
FB91| A5 58                               LDA LMARGIN
FB93| 85 5C                               STA CH
FB95| 20 DDFB                             JSR CURDOWN      ; GOTO NEXT LINE
FB98| 90F4                               BCC CLEOP1
FB9A| 68                               PLA
FB9B| A8                               TAY
FB9C| 68                               PLA              ; RESTORE CURSOR POSITION
FB9D| 85 5C                               STA CH
FB9F| 98                               TYA              ; GET OLD CV IN ACC AGAIN
FBA0| B023                               BCS SETCV        ; BRANCH ALWAYS
FBA2|                                     ;
FBA2| A5 5C                               CLEOL LDA CH      ; CLEAR TO END OF LINE FIRST
FBA4| 4C 89FC                             JMP CLEOL1
FBA7|                                     ;
FBA7| C9 80                               CONTROL CMP #80    ; IF INVERSE
FBA9| 9065                               BCC DISPLAYX
FBAB| C9 8D                               TSTCR CMP #8D    ; IF CARRIAGE RETURN THEN NEW LINE
FBAD| D03A                               BNE TSTBACK
FBAF| 20 A2FB                             CARRAGE JSR CLEOL ; FIRST CLEAR TO THE END OF THIS LINE
FBB2| 20 D7FB                             JSR SETCHZ       ; RESET CURSOR AND GOTO NEXT LINE (CARRY IS SET)
FBB5| 4C 16FC                             JMP NXTLIN       ; THEN GOTO THE NEXT LINE.
FBB8|                                     ;
FBB8| A5 5D                               CURUP LDA CV      ; TEST FOR TOP OF SCREEN
FBBA| C6 5D                               DEC CV           ; ANTICIPATE 'NOT' TOP
FBBC| C5 5A                               CMP WINTOP
FBBE| D002                               BNE CURUP1      ; IT'S NOT TOP, CONTINUE
FBC0| A5 5B                               LDA WINBTM      ; WRAP AROUND TO BOTTOM
FBC2| 38                               CURUP1 SEC       ; DECREMENT BY ONE
FBC3| E9 01                               SBC #01
FBC5| 85 5D                               SETCV STA CV     ; SAVE NEW VERTICAL LINE
FBC7| FBC7                               BASCALC .EQU *
FBC7| FBC7                               CURDN1 .EQU *
FBC7| A5 5D                               LDA CV          ; GET VALUES FOR FIRST PAGE ($400)
FBC9| 104E                               BPL BASCALC1    ; ALWAYS
FBCB|                                     ;
FBCB| 24 68                               CURIGHT BIT MODES ; TEST FOR 80 OR 40
FBCD| 7002                               BVS RIGHT1
FBCF| E6 5C                               INC CH
FBD1| E6 5C                               RIGHT1 INC CH    ; BUMP CURSOR HORIZONTAL

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 8

```

FBD3| A5 5C          LDA      CH          ; TEST FOR NEW LINE
FBD5| C5 59          CMP      RMARGIN
FBD7| A5 58          SETCHZ   LDA      LMARGIN      ; JUST IN CASE WE HAVE.
FBD9| 905D          BCC      CTRLRET
FBD8| 85 5C          SETCVH   STA      CH          ; CURSOR AT START OF NEXT LINE
FBD0|              ; DROP INTO CURDOWN FOR WRAP AROUND
FBD0|              ;
FBD0| E6 5D          CURDOWN  INC      CV          ; MOVE CURSOR DOWN ONE LINE
FBD5| A5 5D          LDA      CV          ; ANTICIPATE NOT BOTTOM
FBE1| C5 5B          CMP      WINBTM      ; TEST FOR BOTTOM
FBE3| 90E2          BCC      CURDN1
FBE5| A5 5A          LDA      WINTOP
FBE7| B0DC          BCS      SETCV      ; BRANCH ALWAYS
FBE9|              ;
FBE9| C9 88          TSTBACK  CMP      #88         ; BACKSPACE?
FBE8| D05D          BNE      TSTBELL
FBE1| 24 68          CURLEFT  BIT      MODES      ; TEST FOR FOURTY OR EIGHTY MODE
FBEF| 7002          BVS      LEFT80
FBF1| C6 5C          DEC      CH
FBF3| C6 5C          LEFT80  DEC      CH
FBF5| 3006          BMI      LEFTUP
FBF7| A5 5C          LDA      CH          ; TEST FOR WRAP AROUND
FBF9| C5 58          CMP      LMARGIN
FBFB| 103B          BPL      CTRLRET
FBFD| 20 B8FB          LEFTUP  JSR      CURUP
FC00| A5 59          LDA      RMARGIN
FC02| 85 5C          STA      CH          ; SAVE NEW CURSOR POSITION
FC04| D0E7          BNE      CURLEFT      ; BRANCH ALWAYS
FC06|              ;
FC06| C9 A0          COUT2    CMP      #0A0        ; IS IT CONTROL CHARACTER
FC08| 909D          BCC      CONTROL
FC0A| 24 68          BIT      MODES      ; TEST FOR INVERSE
FC0C| 3002          BMI      DISPLAYX      ; NO PUT IT OUT
FC0E| 29 7F          AND      #7F         ; STRIP HI BIT
FC10| 20 9DFC          DISPLAYX JSR      DISPLAY
FC13|              ;
FC13| 20 CFBF          INCHORZ JSR      CURRIGHT      ; MOVE CURSOR RIGHT
FC16| B043          NXTLIN   BCS      SCROLL      ; IT'S BOTTOM, RESET CH=0 AND SCROLL
FC18| 60            RTS          ; RESET CH ONLY
FC19|              ;
FC19| 08            BASCALC1 PHP      ; CALC BASE ADR IN BAS4L,H
FC1A| 48            PHA
FC1B| 4A            LSR      A          ; FOR GIVEN LINE NO.
FC1C| 29 03          AND      #03         ; 0<=LINE NO.<$17
FC1E| 09 04          ORA      #04         ; ARG=000ABCDE, GENERATE
FC20| 85 5F          STA      BAS4H      ; BAS4H=000001CD
FC22| 49 0C          EOR      #0C
FC24| 85 61          STA      BAS8H
FC26| 68            PLA          ; AND
FC27| 29 18          AND      #18         ; BAS4L=EABAB000
FC29| 9002          BCC      BSCLC2
FC2B| 69 7F          ADC      #7F
FC2D| 85 5E          STA      BAS4L
FC2F| 0A            ASL      A
FC30| 0A            ASL      A
FC31| 05 5E          ORA      BAS4L
FC33| 85 5E          STA      BAS4L
FC35| 85 60          STA      BAS8L      ; SAME FOR PAGE 2
FC37| 28            PLP
FC38| 60            CTRLRET  RTS
FC39|              ;
FC39| 48            COUT      PHA          ; SAVE CHARACTER
FC3A| 84 6D          STY      TEMPY
FC3C| 86 6C          STX      TEMPX
FC3E| 20 47FC          JSR      COUT1
FC41| A4 6D          LDY      TEMPY
FC43| A6 6C          LDX      TEMPX
FC45| 68            PLA
FC46| 60            RTS
FC47| 6C 6E00          COUT1   JMP      @CSWL      ; NORMALLY COUT1
FC4A|              ;
FC4A| C9 87          TSTBELL  CMP      #87         ; BELL?
FC4C| D004          BNE      LNFD         ; NO TEST FOR FORM FEED
FC4E| AE 40C0          BELL    LDX      0C040      ; SOUND BELL
FC51| 60            RTS
FC52| C9 8A          LNFD     CMP      #8A         ; LINE FEED?
FC54| D0E2          BNE      CTRLRET
FC56| 20 DDFB          JSR      CURDOWN      ; MOVE CURSOR DOWN A LINE
FC59| 90DD          BCC      CTRLRET      ; BRANCH IF NO SCROLL NECESSARY.
FC5B|              ;
FC5B| A5 5A          SCROLL   LDA      WINTOP      ; START WITH TOP LINE
FC5D| 48            PHA          ; SAVE IT FOR NOW
FC5E| 20 C5FB          JSR      SETCV      ; GET BASCALC FOR THIS LINE
FC61| A2 03          SCRL1    LDX      #03         ; MOVE CURRENT BASCALC AS DESTINATION
FC63| B5 5E          SCRL2    LDA      BAS4L,X
FC65| 95 58          STA      TBAS4L,X      ; (TEMPORARY BASE ADDR.)
FC67| CA            DEX
FC68| 10F9          BPL      SCRL2
FC6A| 68            PLA          ; GET DESTINATION LINE
FC6B| 18            CLC

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 9

```

FC6C| 69 01      ADC    #01      ; CALCULATE SOURCE LINE.
FC6E| C5 5B      CMP    WINBTM    ; IS IT THE LAST LINE?
FC70| B015      BCS    LASTLN    ; YES, CLEAR IT
FC72| 48         PHA          ; SAVE AS NEXT DESTINATION LINE
FC73| 20 C5FB    JSR    SETCV     ; GET BASE ADDR FOR SOURCE LINE
FC76| A5 59      LDA    RMARGIN   ; MOVE SOURCE TO DESTINATION
FC78| 4A         LSR    A         ; DIVIDE BY 2
FC79| A8         TAY
FC7A| 88         SCRL3    DEY     ; DONE YET
FC7B| 30E4      BMI    SCRL1    ; YES, DO NEXT LINE
FC7D| B1 5E      LDA    (BAS4L),Y
FC7F| 91 58      STA    (TBAS4L),Y
FC81| B1 60      LDA    (BAS8L),Y
FC83| 91 64      STA    (TBAS8L),Y
FC85| 90F3      BCC    SCRL3    ; BRANCH ALWAYS
FC87| A5 58      LASTLN    LDA    LMARGIN ; BLANK FILL THE LAST LINE
FC89| 4A         CLEOL1    LSR    A         ; DIVIDE BY 2
FC8A| A8         TAY
FC8B| B004      BCS    CLEOL2
FC8D| A5 66      LDA    FORGND    ; (NORMALLY A SPACE)
FC8F| 91 5E      STA    (BAS4L),Y
FC91| A5 67      CLEOL2    LDA    BKGND    ; (IF 80 COLUMNS, ALSO A SPACE)
FC93| 91 60      STA    (BAS8L),Y
FC95| C8         INY
FC96| 98         TYA
FC97| 0A         ASL    A         ; TEST FOR END OF LINE
FC98| C5 59      CMP    RMARGIN   ; MULT BY 2 AGAIN
FC9A| 90ED      BCC    CLEOL1    ; CONTINUE IF MORE TO DO.
FC9C| 60         RTS          ; ALL DONE.
FC9D|           ;
FC9D| 24 68      DISPLAY  BIT    MODES    ; TEST FOR 40 OR 80
FC9F| 700C      BVS    DSPL80    ; STORE THE SINGLE CHARACTERS AND RETURN
FCA1| 46 5C      LSR    CH        ; INSURE PROPER 40 COLUMN DISPLAY
FCA3| 06 5C      ASL    CH        ; BY DROPPING BIT 0
FCA5| 20 ADFF    JSR    DSPL80    ; DISPLAY IN $400 PAGE.
FCA8| A5 67      LDA    BKGND    ; ALSO SET BACKGROUND COLOR
FCAA| 91 60      DSPBKGND STA    (BAS8L),Y
FCAC| 60         RTS
FCAD|           ;
FCAD| 48      DSPL80    PHA          ; PRESERVE CHARACTER
FCAE| A5 5C      LDA    CH        ; DETERMINE WHICH PAGE
FCB0| 4A         LSR    A
FCB1| A8         TAY
FCB2| 68         PLA
FCB3| B0F5      BCS    DSPBKGND ; BRANCH IF $900 PAGE
FCB5| 91 5E      STA    (BAS4L),Y
FCB7| 60         RTS
FCB8|           ;
FCB8| B1 7E      NOTCR   LDA    (INBUF),Y ; ECHO CHARACTER
FCBA| 20 39FC    JSR    COUT
FCBD| C9 88      CMP    #88      ; BACKSPACE
FCBF| F01D      BEQ    BKSPCE
FCC1| C9 98      CMP    #98      ; CANCEL?
FCC3| F008      BEQ    CANCEL
FCC5| E6 80      INC    TEMP
FCC7| A5 80      LDA    TEMP
FCC9| C9 50      CMP    #INBUFLN
FCCB| D017      BNE    NXTCHAR    ; NO WRAP AROUND ALLOWED.
FCCD| A9 DC      CANCEL  LDA    #0DC ; OUTPUT BACKSLASH
FCCF| 20 39FC    JSR    COUT
FCD2| 20 EFFC    JSR    CROUT
FCD5| FCD5      GETLNZ   .EQU    *
FCD5| A5 6B      GETLN   LDA    PROMPT
FCD7| 20 39FC    JSR    COUT
FCDA| A0 01      LDY    #01
FCDC| 84 80      STY    TEMP     ; START AT BEGINNING OF INBUF
FCDE| A4 80      BKSPCE  LDY    TEMP
FCE0| F0F3      BEQ    GETLN
FCE2| C6 80      DEC    TEMP     ; BACK UP INPUT BUFFER
FCE4| 20 60FD    NXTCHAR JSR    RDCHAR ; GET INPUT
FCE7| A4 80      LDY    TEMP
FCE9| 91 7E      STA    (INBUF),Y
FCEB| C9 8D      CMP    #8D
FCED| D0C9      BNE    NOTCR
FCEF| FCEF      CROUT   .EQU    *
FCEF| 2C 00C0    BIT    KBD       ; TEST FOR START/STOP
FCF2| 1013      BPL    NOSTOP
FCF4| 20 2EFD    JSR    KEYIN3    ; READ KBD
FCF7| C9 A0      CMP    #0A0     ; IS IT A SPACE?
FCF9| F007      BEQ    STOPLST    ; YES, PAUSE TIL NEXT KEYPRESS.
FCFB| C9 89      CMP    #89      ; QUIT THIS OPERATION
FCFD| D008      BNE    NOSTOP    ; NO, IGNORE THIS KEY.
FCFF| 4C 9FFA    JMP    ERROR2    ; YES, RESTART
FD02| AD 00C0    STOPLST LDA    KBD
FD05| 10FB      BPL    STOPLST
FD07| A9 8D      NOSTOP  LDA    #8D
FD09| 4C 39FC    JMP    COUT
FD0C|           ;
FD0C| 6C 7000    RDKEY   JMP    @KSWL
FD0F|           ;

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 10

```

FD0F| A9 7F          KEYIN   LDA    #7F          ; MAKE SURE FIRST IS CURSOR
FD11| 85 63          STA    TBAS4H
FD13| 20 88FD        JSR    PICK          ; GO READ SCREEN
FD16| 48            KEYIN1  PHA          ; SAVE CHR AT CURSOR POSITION
FD17| 20 35FD        JSR    KEYWAIT       ; TEST FOR KEYPRESS
FD1A| B008          BCS    KEYIN2       ; GO GET IT
FD1C| A5 69          LDA    CURSOR       ; GIVE THEM AN UNDERScore FOR A TIME
FD1E| 20 9DFC        JSR    DISPLAY
FD21| 20 35FD        JSR    KEYWAIT       ; GO SEE IF KEYPRESSED
FD24| 68            KEYIN2  PLA          ;
FD25| 08            PHP          ; SAVE KEYPRESS STATUS
FD26| 48            PHA
FD27| 20 9DFC        JSR    DISPLAY
FD2A| 68            PLA
FD2B| 28            PLP
FD2C| 90E8        BCC    KEYIN1
FD2E| AD 00C0      KEYIN3  LDA    KBD          ; READ KEYBOARD
FD31| 2C 10C0      KEYIN4  BIT    KBDSTRB       ; CLEAR KEYBOARD STROBE
FD34| 60            RTS
FD35| E6 58        KEYWAIT INC    TBAS4L       ; JUST KEEP COUNTING
FD37| D009        BNE    KWAIT2
FD39| E6 63        INC    TBAS4H
FD3B| A9 7F        LDA    #7F          ; TEST FOR DONE
FD3D| 18            CLC
FD3E| 25 63        AND    TBAS4H
FD40| F005        BEQ    KEYRET       ; RETURN IF TIMED OUT
FD42| 0E 00C0      KWAIT2  ASL    KBD
FD45| 90EE        BCC    KEYWAIT
FD47| 60            KEYRET  RTS
FD48| ;
FD48| ;
FD48| FD48        ESC3    .EQU    *
FD48| 20 77FD        JSR    GOESC
FD4B| A5 68        ESCAPE  LDA    MODES       ; SET TO + SIGN FOR CURSOR MOVES
FD4D| 29 80        AND    #80
FD4F| 49 AB        EOR    #0AB
FD51| 85 69        STA    CURSOR
FD53| 20 0CFD      ESC1    JSR    RDKEY       ; READ NEXT CHARACTER
FD56| A0 08        LDY    #08          ; TEST FOR ESCAPE COMMAND
FD58| D9 F0FF      ESC2    CMP    ESCTABL,Y
FD5B| F0EB        BEQ    ESC3
FD5D| 88            DEY
FD5E| 10F8        BPL    ESC2          ; LOOP TIL FOUND OR DONE
FD60| ;
FD60| A9 80        RDCHAR  LDA    #80          ; GO READ A CHARACTER
FD62| 25 68        AND    MODES
FD64| 85 69        STA    CURSOR       ; SAVE STANDARD CURSOR
FD66| 20 0CFD      JSR    RDKEY
FD69| C9 9B        CMP    #9B          ; ESCAPE CHARACTER?
FD6B| F0DE        BEQ    ESCAPE
FD6D| C9 95        CMP    #95          ; FORWARD COPY?
FD6F| D0D6        BNE    KEYRET
FD71| 20 88FD      JSR    PICK          ; GET CHARACTER FROM SCREEN
FD74| 09 80        ORA    #80          ; SET TO NORMAL ASCII
FD76| 60            RTS
FD77| ;
FD77| A9 FB        GOESC   LDA    #0FB
FD79| 48            PHA
FD7A| B9 7FFD      LDA    ESCVECT,Y
FD7D| 48            PHA
FD7E| 60            RTS
FD7F| A1            ESCVECT .BYTE    0A1       ; CLEOL-1
FD80| 84            .BYTE    84          ; CLEOP-1
FD81| 7C            .BYTE    7C          ; CLSCRN-1
FD82| 62            .BYTE    62          ; COL40-1
FD83| 5C            .BYTE    5C          ; COL80-1
FD84| EC            .BYTE    0EC        ; CURLEFT-1
FD85| CA            .BYTE    0CA        ; CURRIGHT-1
FD86| DC            .BYTE    0DC        ; CURDOWN-1
FD87| B7            .BYTE    0B7        ; CURUP-1
FD88| ;
FD88| A5 5C        PICK    LDA    CH          ; GET A CHARACTER AT CURRENT CURSOR POSITION
FD8A| 4A            LSR    A          ; DETERMINE WHICH PAGE.
FD8B| A8            TAY
FD8C| 24 68        BIT    MODES       ; AND IF 80 COLUMN MODE
FD8E| 5005        BVC    PICK40       ; FORGET CARRY IF 40 COLUMNS
FD90| 9003        BCC    PICK40       ; GET CHARACTER FROM $400
FD92| B1 60        LDA    (BAS8L),Y
FD94| 60            RTS
FD95| B1 5E        PICK40  LDA    (BAS4L),Y
FD97| 60            RTS
FD98| ;
FD98| FD98        CLDSTRT .EQU    *
FD98| A9 03        LDA    #03
FD9A| 8D D0FF      STA    0FFD0       ; ZERO PAGE IS ON 3!
FD9D| FD9D        SETUP   .EQU    *
FD9D| D8            CLD          ; OF COURSE!
FD9E| A2 03        LDX    #03
FDA0| 86 7F        STX    INBUF+1
FDA2| BD BCFF      SETUP1  LDA    NMIRQ,X

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 11

```

FDA5| 9D CAFF          STA      0FFCA,X
FDA8| BD B4FF          LDA      HOOKS,X
FDAB| 95 6E            STA      CSWL,X
FDAD| BD B8FF          LDA      VBOUNDS,X
FDB0| 95 58            STA      LMARGIN,X
FDB2| CA              DEX
FDB3| 10ED            BPL      SETUP1
FDB5| 85 82            STA      IBDRVN
FDB7| A9 A0            LDA      #0A0      ; INPUT BUFFER AT $3A0
FDB9| 85 7E            STA      INBUF
FDBB| A9 60            LDA      #60
FDBD| 85 81            STA      IBSLOT
FDBF| A9 FF            LDA      #0FF
FDC1| 85 68            STA      MODES
FDC3| 20 63FB          JSR      COL40      ; SET 40 COLUMNS, CLEAR SCREEN
FDC6|                  ;
FDC6| 00A0              ADR      .EQU      0A0
FDC6| 00A0              CPORTL  .EQU      ADR
FDC6| 00A1              CPORTH  .EQU      ADR+1
FDC6| 00A2              CTEMP   .EQU      ADR+2
FDC6| 00A3              CTEMP1  .EQU      ADR+3
FDC6| 00A4              YTEMP    .EQU      ADR+4
FDC6| 00C0              ROWTEMP .EQU      ADR+20
FDC6| C0DB              CWRTON   .EQU      0C0DB
FDC6| C0DA              CWRTOFF  .EQU      0C0DA
FDC6| FFEC              CB2CTRL  .EQU      0FFEC
FDC6| FFED              CB2INT   .EQU      0FFED
FDC6|                  ;
FDC6|                  ;
FDC6| A9 78              GENENTR LDA      #78      ; INIT SCREEN INDX LOCATIONS
FDC8| 85 A0              STA      CPORTL
FDCA| A9 08              LDA      #08
FDCC| 85 A1              STA      CPORTH
FDCE| A9 F0              LDA      #0F0      ; SET UP INDEX TO CHRSET
FDD0| 85 A4              STA      YTEMP
FDD2| A9 00              LDA      #00
FDD4| AA                TAX
FDD5| 95 C0              ZIPTemps STA      ROWTEMP,X
FDD7| E8                INX
FDD8| E0 20              CPX      #20
FDDA| D0F9              BNE      ZIPTemps
FDDC| A9 05              LDA      #05      ; FAKE THE FIRST BIT PATTERN
FDE0| 18                CLC      ; (PHANTOM 9TH BIT SHIFTED AS BIT 0)
FDE2| 08                PHP
FDE4| 48                PHA
FDE6| 86 A2              GENASC  STX      CTEMP      ; GENERATE THE ASCII
FDE8| A0 07              GASC11  LDY      #07      ; CODES FOR THE FIRST PASS
FDEA| A6 A2              GASC12  LDX      CTEMP
FDEC| 8A                GASC13  TXA
FDEE| 91 A0              STA      (CPORTL),Y ; $XXF=CHR 0 / 4
FDEE| E8                INX      ; $XXE=CHR 1 / 5
FDEE| 88                DEY      ; $XXD=CHR 2 / 6
FDEC| 3006              BMI      GASC14 ; $XXC=CHR 3 / 7
FDEE| C0 03              CPY      #03      ; $XXB=CHR 0 / 4
FDF0| D0F5              BNE      GASC13 ; $XXA=CHR 1 / 5
FDF2| F0F1              BEQ      GASC12 ; $XX9=CHR 2 / 6
FDF4| 20 99FE          GASC14  JSR      NXTPORT ; $XX8=CHR 3 / 7
FDF7| B008              BCS      CBYTES ; GO DECODE CHARACTER TABLE
FDF9| C9 0A              CMP      #0A      ; SECOND SET OF 4?
FDFB| D0E6              BNE      GASC11
FDFD| A2 24              LDX      #24
FDFD| D0E0              BNE      GENASC ; BRANCH ALWAYS
FE01| 68                CBYTES  PLA      ; RESTORE BIT PATTERN
FE02| 28                PLP
FE03| A2 17              LDX      #17      ; (4 CHARACTERS OF 6 ROWS)
FE05| A0 05              LDY      #05      ; (FIVE COLUMNS)
FE07| 36 C4              CSHFT   ROL      ROWTEMP+4,X ; BREAK BYTE INTO
FE09| 0A                ASL      A      ; 5 BIT GROUPS
FE0A| D00E              BNE      SHFTCNT ; BRANCH IF MORE BITS IN THIS BYTE
FE0C| 84 A2              STY      CTEMP
FE0E| C6 A4              DEC      YTEMP ; (NOTE. CARRY IS SET)
FE10| F016              BEQ      DONE ; BRANCH IF ALL DONE
FE12| A4 A4              LDY      YTEMP ; GET CHARACTER TABLE INDEX
FE14| B9 C4FE          LDA      CHRSET-1,Y
FE17| 2A                ROL      A ; (CARRY KEEPS BYTE NON-ZERO UNTIL ALL 8 ARE
FE18|                  ; ARE SHIFTED)
FE18| A4 A2              LDY      CTEMP ; RESTORE COLUMN COUNT
FE1A| 88                DEY      ; GOT ALL FIVE BITS?
FE1B| D0EA              BNE      CSHFT ; NO, DO NEXT
FE1D| CA                DEX      ; ALL ROWS DONE
FE1E| 10E5              BPL      CCOLMS ; NO, DO NEXT
FE20| 08                PHP      ; SAVE REMAINING BIT PATTERN AND CARRY
FE21| 48                PHA
FE22| 20 28FE          JSR      STORCHRS ; MOVE EM TO NON DISPLAYED VIDEO AREA
FE25| 4C 01FE          JMP      CBYTES
FE28|                  ;
FE28| FE28              DONE     .EQU      *
FE28|                  ;
FE28| A2 1F              STORCHRS LDX      #1F      ; MOVE CHARACTER PATTERNS TO VIDEO AREA
FE2A| A0 00              STORSET  LDY      #00

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 12

```

FE2C| B5 C0          STOROW LDA ROWTEMP,X
FE2E| 0A           ASL A ; SHIFT TO CENTER
FE2F| 29 3E        AND #3E ; STRIP EXTRA GARBAGE
FE31| 91 A0        STA (CPORTL),Y
FE33| CA          DEX
FE34| C8          INY
FE35| C0 08        CPY #08 ; THIS GROUP DONE
FE37| D0F3        BNE STOROW ; NO, NEXT ROW
FE39| 20 99FE     JSR NXTPORT
FE3C| C9 08        CMP #08
FE3E| F004        BEQ GENDONE ; ALL ROWS STORED?
FE40| 8A          TXA
FE41| 10E7        BPL STORSET
FE43| 60          RTS ; PARTIAL SET ($478-$5FF)
FE44|
FE44| A9 01        ; GENDONE LDA #01 ; SET NORMAL MODE
FE46| 85 A2        STA CTEMP
FE48| A9 60        GEN1 LDA #60 ; PREPARE TO SEND BYTES TO CHARACTER
FE4A| 2C DBC0      BIT CWRTON ; GENERATOR RAM
FE4D| 20 AEFE     JSR VRETRCE ; WAIT FOR NEXT VERTICAL RETRACE
FE50| A9 20        LDA #20 ; WAIT AGAIN
FE52| 20 AEFE     JSR VRETRCE
FE55| 2C DAC0      BIT CWRTOFF ; CHARACTERS ARE NOW LOADED
FE58| 20 88FE     JSR ALTCHR ; REPEAT THIS SET FOR OTHER 64 CHARACTERS
FE5B| C6 A2        DEC CTEMP ; HAVE WE DONE ALTERNATES YET?
FE5D| 1016        BPL GEN2 ; NO, DO IT!
FE5F| A9 08        LDA #08 ; BUMP ASCII VALUES FOR NEXT SET
FE61| 85 A1        STA CPORTH
FE63| A0 07        NXTASCI LDY #07 ; THE USUAL COUNTDOWN
FE65| B1 A0        NXTASC2 LDA (CPORTL),Y
FE67| 18          CLC
FE68| 69 08        ADC #08
FE6A| 91 A0        STA (CPORTL),Y
FE6C| 88          DEY
FE6D| 10F6        BPL NXTASC2
FE6F| 20 99FE     JSR NXTPORT
FE72| 90EF        BCC NXTASCI
FE74| 60          RTS
FE75| A0 03        GEN2 LDY #03 ; SETUP ALTERNATE WITH UNDERLINES
FE77| A9 7F        LDA #7F
FE79| 99 FC05     UNDER STA 05FC,Y
FE7C| 99 FC07     STA 07FC,Y
FE7F| 88          DEY
FE80| 10F7        BPL UNDER
FE82| A9 08        LDA #08
FE84| 85 A1        STA CPORTH
FE86| D0C0        BNE GEN1
FE88|
FE88| A0 07        ; ALTCHR LDY #07 ; ADJUST ASCII FOR ALTERNATE SET
FE8A| B1 A0        ALTCL1 LDA (CPORTL),Y
FE8C| 49 20        EOR #20 ; $20--> $40-->$60
FE8E| 91 A0        STA (CPORTL),Y
FE90| 88          DEY
FE91| 10F7        BPL ALTCL1 ; ADJUST THEM ALL
FE93| 20 99FE     JSR NXTPORT
FE96| 90F0        BCC ALTCHR
FE98| 60          RTS
FE99|
FE99| A5 A0        ; NXTPORT LDA CPORTL ; CONVERT $78->$F8 OR $F8-$78
FE9B| 49 80        EOR #80
FE9D| 85 A0        STA CPORTL
FE9F| 3002        BMI NOHIGH
FEA1| E6 A1        INC CPORTH
FEA3| A5 A1        NOHIGH LDA CPORTH
FEA5| C9 0C        CMP #0C
FEA7| D004        BNE PORTDN
FEA9| A9 04        LDA #04
FEAB| 85 A1        STA CPORTH
FEAD| 60          PORTDN RTS
FEAE|
FEAE| 85 A3        ; VRETRCE STA CTEMP1 ; SAVE BITS TO BE STORED
FEB0| AD ECFE     LDA CB2CTRL ; CONTROL PORT FOR 'CB2'
FEB3| 29 3F        AND #3F ; RESET HI BITS TO 0
FEB5| 05 A3        ORA CTEMP1
FEB7| 8D ECFE     STA CB2CTRL
FEBA| A9 08        LDA #08 ; TEST VERTICAL RETRACE
FEBB| 8D EDFF     STA CB2INT
FEBF| 2C EDFF     VWAIT BIT CB2INT ; WAIT FOR RETRACE
FEC2| F0FB        BEQ VWAIT
FEC4| 60          RTS
FEC5|
FEC5| FEC5        ; CHRSET .EQU *
FEC5|
FEC5| F0 01 82 18 40 84 81 .BYTE 0F0,01,82,18,40,84,81,2F,58,44,81,29,02,1E,01,91,7C,1F,49,30
FEC5| 2F 58 44 81 29 02 1E
FEC5| 01 91 7C 1F 49 30
FEC5| 8A 08 43 14 31 2A 22 .BYTE 8A,08,43,14,31,2A,22,13,0E3,0F7,0C4,91,48,0A2,0DA,24,0C6,4A
FEC5| 13 E3 F7 C4 91 48 A2
FEC5| DA 24 C6 4A
FEC5| 62 8C 24 C6 F8 63 8C .BYTE 62,8C,24,C6,0F8,63,8C,0C1,46,17,52,8A,0AF,16,14,0E3,33,31

```

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 13

```

FEF2| C1 46 17 52 8A AF 16
FEF9| 14 E3 33 31
FEFD| C6 F8 DC 73 3F 46 17
FF04| 62 8C 21 E6 18 6A 8D
FF0B| 61 CF 18 62
FF0F| 74 D1 B9 18 49 4C 91
FF16| C0 F3 09 2C 91 C0 14
FF1D| 1D 8C EF 07
FF21| 17 43 88 31 84 1E DF
FF28| 0B 31 84 F8 FE 77 3E
FF2F| 3E 17 62 8C FD
FF34| C7 50 E3 0B 51 C5 E8
FF3B| C8 73 18 0C 42 3E 01
FF42| 02 20 42 3E
FF46| 41 18 8C 08 00 70 EE
FF4D| 00 11 11 21 11 02 E0
FF54| 3C 21 31 02 E0
FF59| 1C 00 C8 B9 80 62 14
FF60| 1F 46 A2 DE 43 2C 04
FF67| 88 BE FF CE
FF6B| 7D 37 49 88 95 18 98
FF72| 09 62 D1 44 E8 88 FB
FF79| 02 90 40 00 10
FF7E| E0 03 02 00 40 00 00
FF85| 08 00 00 28 10 42 44
FF8C| 25 82 B8 2F 48
FF91| 25 44 10 82 02 00 2F
FF98| 5A 40 45 02 8E 64 50
FF9F| 90 01 3E 26 42 80
FFA5| 21 80 00 05 00 F8 80
FFAC| 00 05 08 F8 80 28 05
FFB3| 88
FFB4|
FFB4| FFB4 ;
FFB4| 06FC HOOKS .EQU *
FFB6| 0FFD .WORD COUT2
FFB8| FFB8 VBOUNDS .WORD KEYIN
FFB8| 00 50 00 18 .EQU *
FFB8| .BYTE 00,50,00,18
FFBC|
FFBC| 4C 86F6 ; NMIRQ JMP RECON ; IN DIAGNOSTICS
FFBF| 40 RTI
FFC0|
FFC0| 43 4F 50 59 52 49 47 ;
FFC7| 48 54 20 4A 41 4E 55 .ASCII "COPYRIGHT JANUARY, 1980 APPLE COMPUTER INC..JRH"
FFCE| 41 52 59 2C 20 31 39
FFD5| 38 30 20 20 41 50 50
FFDC| 4C 45 20 43 4F 4D 50
FFE3| 55 54 45 52 20 49 4E
FFEA| 43 2E 2E 4A 52 48
FFF0|
FFF0| CC D0 D3 B4 B8 88 95 ;
FFF7| 8A 8B 00 ESCTABL .BYTE 0CC,0D0,0D3,0B4,0B8,88,95,8A,8B,00
FFFA|
FFFA| CAFF ;
FFFC| EEF4 RESET .WORD 0FFCA
FFFE| CDFD IRQ .WORD DIAGN ; NOTHING
      .WORD 0FFCD
      .END

```

## SYMBOL TABLE DUMP

```

AB - Absolute    LB - Label    UD - Undefined    MC - Macro
RF - Ref         DF - Def      PR - Proc         FC - Func
PB - Public      PV - Private  CS - Consts

```

```

A1H  AB 0075 | A1L  AB 0074 | A1PC  LB F9D6 | A1PC1  LB F9D9 | A2H  AB 0077 |
A2L  AB 0076 | A3H  AB 0079 | A3L  AB 0078 | A4H  AB 007B | A4L  AB 007A |
ADR  AB 00A0 | ALTC1 LB FE8A | ALTCHR LB FE88 | ASC1  LB FB40 | ASC2  LB FB4C |
ASC3  LB FB5A | ASCDONE LB FA08 | ASCII LB FA1B | ASCII0 LB FA1D | ASCII1 LB F9E1 |
ASCII2 LB F9E3 | ASCII3 LB F9F4 | BAS4H  AB 005F | BAS4L  AB 005E | BAS8H  AB 0061 |
BAS8L  AB 0060 | BASCALC LB FBC7 | BASCALC1 LB FC19 | BELL  LB FC4E | BITOFF  LB FA29 |
BITON  LB FA25 | BKGND  AB 0067 | BKSPCE  LB FCDE | BL1  LB FAB4 | BLOCKIO AB F479 |
BSCLC2 LB FC2D | CANCEL  LB FCCD | CARRAGE  LB FBAB | CB2CTRL AB FFEC | CB2INT  AB FFED |
CBYTES  LB FE01 | CCOLMS  LB FE05 | CH  AB 005C | CHRSET  LB FEC5 | CKMDE  LB FA1E |
CLDSTRT LB FD98 | CLEOL  LB FBA2 | CLEOL1  LB FC89 | CLEOL2  LB FC91 | CLEOP  LB FB85 |
CLEOP1  LB FB8E | CLSCRN  LB FB7D | CMDSRCH  LB F91C | CMDTAB  LB F96C | CMDVEC  LB F97D |
COL40  LB FB63 | COL80  LB FB5D | CONTROL  LB FBA7 | COUT  LB FC39 | COUT1  LB FC47 |
COUT2  LB FC06 | CPORTH  AB 00A1 | CPORTL  AB 00A0 | CRCHK  LB F9FD | CRMON  LB FA3A |
CROUT  LB FCEF | CSHFT  LB FE07 | CSWH  AB 006F | CSWL  AB 006E | CTEMP  AB 00A2 |
CTEMP1  AB 00A3 | CTRLRET LB FC38 | CURDN1  LB FBC7 | CURDOWN LB FBDD | CURIGHT  LB FBCB |
CURLEFT LB FBED | CURSOR  AB 0069 | CURUP  LB FBB8 | CURUP1  LB FBC2 | CV  AB 005D |
CWRTOFF AB C0DA | CWRTON  AB C0DB | DEST  LB FFA5 | DIAGN  AB F4EE | DIGIT  LB F941 |
DIGRET  LB F96B | DISPLAY LB FC9D | DISPLAYX LB FC10 | DONE  LB FE28 | DSPBKGNL LB FCAA |
DSPL80  LB FCAD | DUMMY  LB FABC | DUMP  LB FB0D | DUMP0  LB FB10 | DUMP1  LB FB1D |
DUMP2  LB FB20 | DUMP3  LB FB30 | DUMP8  LB FAFD | DUMPASC LB FB35 | ENTRY  LB F901 |
ERROR  LB FFA2 | ERROR1  LB FB0B | ERROR2  LB FAF9 | ESC1  LB FD53 | ESC2  LB FD58 |
ESC3  LB FD48 | ESCAPE  LB FD4B | ESCTABL  LB FFF0 | ESCVECT  LB FD7F | FORGND  AB 0066 |
GASCI1  LB FDE3 | GASCI2  LB FDE5 | GASCI3  LB FDE7 | GASCI4  LB FDF4 | GEN1  LB FE48 |

```

↑  
J. R. Huston  
(also worked  
on SOS)

J=James  
R=Richard  
aka  
Dick  
Huston

10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 14

GEN2	LB FE75	GENASC	LB FDE1	GENDONE	LB FE44	GENENTR	LB FDC6	GETLN	LB FCD5
GETLNZ	LB FCD5	GETNUM	LB F92C	GO	LB FA91	GOESC	LB FD77	HOOKS	LB FFB4
IBBUF	AB 0085	IBCMD	AB 0087	IBDRVN	AB 0082	IBSLOT	AB 0081	INBUF	AB 007E
INBUFLN	AB 0050	INCHORZ	LB FC13	IRQ	LB FFEE	JUMP	LB FA8F	KBD	AB C000
KBDSTRB	AB C010	KEYIN	LB FD0F	KEYIN1	LB FD16	KEYIN2	LB FD24	KEYIN3	LB FD2E
KEYIN4	LB FD31	KEYRET	LB FD47	KEYWAIT	LB FD35	KSWH	AB 0071	KSWL	AB 0070
KWAIT2	LB FD42	LASTLN	LB FC87	LEFT80	LB FBF3	LEFTUP	LB FBFD	LFA36	LB FA36
LMARGIN	AB 0058	LNFD	LB FC52	MASK	AB 0069	MISMATCH	LB FA66	MODES	AB 0068
MON	LB F904	MONITOR	PR ----	MONZ	LB F908	MOVE	LB FA40	MOVNXT	LB FA45
NMI	LB FFFA	NMIRQ	LB FFBC	NOHIGH	LB FEA3	NOSTOP	LB FD07	NOTCR	LB FCB8
NOVER	LB FAF3	NXTA1	LB F994	NXTA4	LB F98E	NXTASC2	LB FE65	NXTASCI	LB FE63
NXTBAS	LB F94F	NXTBIT	LB F947	NXTBS2	LB F959	NXTCHAR	LB FCE4	NXTCHR	LB F932
NXTINP	LB F915	NXTLIN	LB FC16	NXTPORT	LB FE99	OLDPC	LB F9E0	PCH	AB 0073
PCL	AB 0072	PICK	LB FD88	PICK40	LB FD95	PORTDN	LB FEAD	PRA1BYTE	LB FA82
PRBYCOL	LB F9C4	PRBYTE	LB F9AE	PRBYTSP	LB FA84	PRCOLON	LB F9C7	PRHEX	LB F9B7
PRHEX2	LB F9C1	PRHEXZ	LB F9B9	PRINTA1	LB FA75	PROMPT	AB 006B	PRSPC	LB FA87
RCHAR	LB FD60	RKEY	LB FD0C	READ	LB FAD4	RECON	AB F686	REPEAT	LB FA2D
REPEAT1	LB FA35	RESET	LB FFFC	RET1	LB F7FE	RET2	LB F900	RET3	LB F882
RETA1	LB F9AD	RIGHT1	LB FBD1	RMARGIN	AB 0059	ROWTEMP	AB 00C0	RWERROR	LB FA97
RWLOOP	LB FADB	SAVCM	LB FAD9	SCAN	LB F912	SCRL1	LB FC61	SCRL2	LB FC63
SCRL3	LB FC7A	SCRNLOC	AB 0058	SCROLL	LB FC5B	SEARCH	LB FA09	SEP	LB FAAE
SET80	LB FB67	SET80A	LB FB6F	SET80B	LB FB7B	SETCHZ	LB FBD7	SETCV	LB FBC5
SETCVH	LB FBDB	SETMDZ	LB FAD1	SETMODE	LB FACC	SETUP	LB FD9D	SETUP1	LB FDA2
SHFTCNT	LB FE1A	SPCE	LB FAB8	SRCH1	LB FA15	STACK	AB 006A	STATE	AB 007C
STOPLST	LB FD02	STOR	LB FABF	STOR1	LB FAC3	STORCHRS	LB FE28	STOROW	LB FE2C
STORSET	LB FE2A	SVMASK	LB F9D3	TBAS4H	AB 0063	TBAS4L	AB 0058	TBAS8H	AB 0065
TBAS8L	AB 0064	TEMP	AB 0080	TEMPX	AB 006C	TEMPY	AB 006D	TOSUB	LB F95E
TST80WID	LB F9CB	TSTA1	LB F99D	TSTBACK	LB FBE9	TSTBELL	LB FC4A	TSTCR	LB FBAB
TSTDUMP	LB FB0A	UNDER	LB FE79	USER	LB FA8C	USERADR	AB 0358	VBOUNDS	LB FFB8
VRETRCE	LB FEAE	VRFY	LB FA4F	VRFY1	LB FA54	VRFY2	LB FA60	VWAIT	LB FEBF
WINBTM	AB 005B	WINTOP	AB 005A	WRTE	LB FAD7	YSAV	AB 007D	YTEMP	AB 00A4
ZIPTEMPS	LB FDD5	ZSTATE	LB F967						

Assembly complete: 1129 lines  
 0 Errors flagged on this Assembly

## 6502 OPCODE STATIC FREQUENCIES

```

ADC : 5 | ***
AND : 14 | *****
ASL : 12 | *****
BCC : 21 | *****
BCS : 20 | *****
BEQ : 82 | *****
BIT : 12 | *****
BMI : 7 | ****
BNE : 41 | *****
BPL : 18 | *****
BVC : 2 | *
BVS : 3 | *
CLC : 7 | ****
CLD : 2 | *
CMP : 35 | *****
CPX : 1 m
CPY : 2 | *
DEC : 7 | ****
DEX : 7 | ****
DEY : 9 | *****
EOR : 6 | ***
INC : 18 | *****
INX : 3 | *
INY : 3 | *
JMP : 18 | *****
JSR : 79 | *****
LDA : 117 M *****
LDX : 12 | *****
LDY : 20 | *****
LSR : 11 | *****
ORA : 10 | *****
PHA : 16 | *****
PHP : 4 | **
PLA : 14 | *****
PLP : 3 | *
ROL : 4 | **
RTI : 1 m
RTS : 34 | *****
SBC : 67 | *****
SEC : 5 | ***
SEI : 1 m
STA : 72 | *****
STX : 7 | ****
STY : 5 | ***
TAX : 2 | *
TAY : 5 | ***
TSX : 1 m
TXA : 2 | *
TXS : 1 m
TYA : 3 | *

```



10/31/89 10:04

HD:Apple ///:ROM - Monitor

Page 15

Minimum frequency = 1  
Maximum frequency = 117

Average frequency = 17

Unused opcodes:

BRK CLI CLV NOP ROR SED

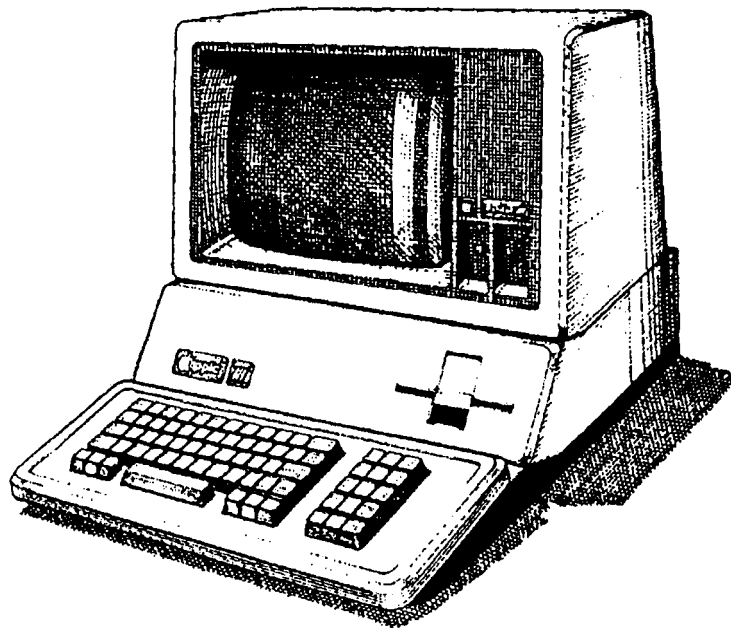
Program opcode usage: 89 %

-----  
(1.00) That's all, Folks ...  
-----

≡FINIS≡



# Apple III Computer Information



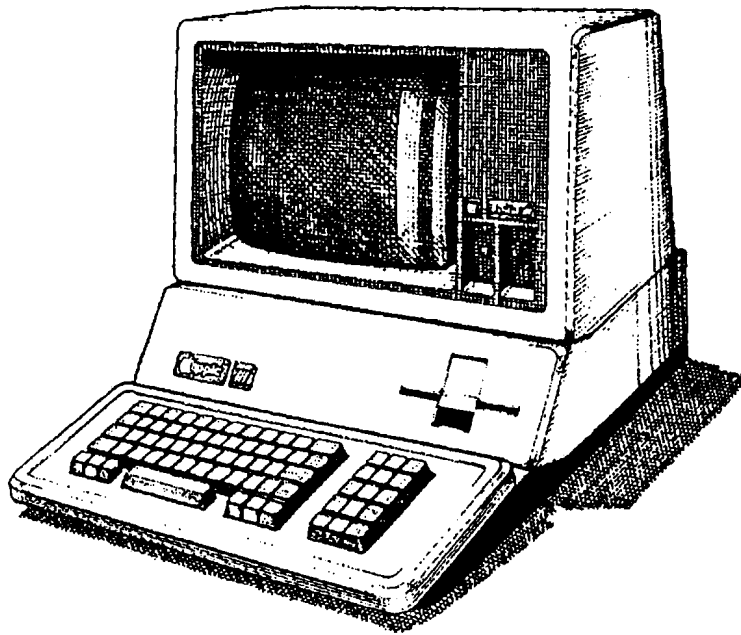
## Inside the Apple III ROM

### Document Table of Contents

Revision 2	•	04 Dec 1997
Revision 1	•	30 Nov 1997



## Apple III Computer Information



# Inside the Apple III ROM

Revision 2 • 04 Dec 1997

---

# Inside the Apple /// Computer ROM

---

David T. Craig • 04 December 1997  
71533.606@compuserve.com

## TABLE OF CONTENTS

- 1 INTRODUCTION
- 2 ROM SECTIONS
- 3 IMPORTANT ROM ROUTINES
- 4 ROM TABLES
- 5 ROM USAGE BY SOS
- 6 MONITOR COMMANDS
- 7 A FEW COMMENTS
- 8 REFERENCES
- 9 DOCUMENT MODIFICATION HISTORY

## 1 INTRODUCTION

This document provides a general overview of the contents of the Apple /// computer ROM revision 1. This information should be used in conjunction with a copy of the ROM source code listing. The audience of this document is anyone with an interest in the technology of the Apple /// computer's hardware and software.

### NOTE

There were two revisions of the Apple /// ROM, revision 0 and revision 1. Revision 0 ROMs had at address F1B9 the value 60. Revision 1 ROMs had at address F1B9 the value A0.

This ROM contains 4 KB of 6502 programming and several data tables. The ROM occupies memory addresses F000–FFFF. The basic purpose of the ROM is to test the Apple /// computer hardware and boot an operating system from the ///'s built-in floppy disk drive. The ROM also contains a simple Monitor program whose purpose is to allow the user to interact with the /// at the hexadecimal level.

Apple planned from an architectural perspective to support two 4K ROMs. But only one ROM was ever created. The Environment Register let you control which ROM was active. Both ROMs shared the same address space so you could only have one ROM active at a time. This feature would have doubled the ROM's effective size providing Apple with more room for ROM-based features that higher-level /// software (e.g. SOS) could have used.

When the Apple /// computer is turned on the ROM's flow of execution is as follows:

- 1) The ROM starts execution at the address contained in FFFC-FFFD (RESET) which is address F4EE (DIAGN).
- 2) Diagnostics (DIAGN/F4EE) starts. The diagnostic first initializes some memory for the ROM's use. If the Open Apple and the Control keys are held down then enter the ROM Monitor. Otherwise run several diagnostic checks of the /// hardware (tests zero page, sizes memory, initializes screen text buffer, tests stack memory, tests ROM checksum, tests VIA chip, tests ACIA chip, tests A/D circuitry, tests keyboard connection). Any diagnostic failures display an error message and the user has to reset the computer.
- 3) Read block 0 (512 bytes) to address A000 from the floppy disk in the built-in disk drive (BOOT/F6A1). If no disk is found or block 0 cannot be read then display "RETRY" and wait for the user to reset the computer. If the block is successfully read then execute the block contents (this is called the SOS Bootstrap Loader: see section ROM USAGE BY SOS).

## 2 ROM SECTIONS

Section	Address	Purpose
Disk I/O	F000-F4C4	Read and write floppy disk blocks (512 bytes each)
Diagnostics	F4C5-F7FE	Diagnose the /// hardware
Monitor	F7FF-FFFF	Interacts with user so user can do simple things

## 3 IMPORTANT ROM ROUTINES

BLOCKIO / F479	Reads or write a disk block (512 bytes), calls routine REGRWTS (F000) which reads a sector (256 bytes) from the disk.
BOOT / F6A1	Read floppy disk block *0 into address A000, execute the block.
ENTRY / F901	Monitor entry point.
DIAGN / F4EE	Diagnostic entry point.
USREENTRY/F6E6	Tests RAM and displays a table showing chip failures (users may execute this routine from the Monitor). This test is aimed at Apple ///s with 128K of RAM that exists on the older 12-Volt RAM boards. Though this routine will work with the newer 5-Volt RAM boards (256K) this test shows wrong information when RAM errors occur since the two RAM boards contain a different number of RAM chips. You can identify the different RAM boards as follows: The 5V boards have a large gray ceramic resistor near the edge and the 12V boards have a small blue tubular capacitor. To test the ///'s RAM you really should use Apple's /// Diagnostics Disk which lets you specify which RAM board you have.

## 4 ROM TABLES

Here's a list of the important data tables in the ROM. This list does not include disk I/O tables.

## Table Name / Address    Contents

---

CHRSET / FEC5-FFB3	Default character set (overridden when SOS loads the character set from SOS.DRIVER)
Copyright / FFC0-FFEF	Copyright message (contains the initials "JRH" for J. R. "Dick" Huston who was a key player behind the /// and SOS)
NMI / FFFA-FFFB	Jump address for the Non-Maskable Interrupt signal
RESET / FFFC-FFFD	Jump address when the /// is powered on
IRQ / FFFE-FFFF	Jump address for the Interrupt Request signal

---

**5    ROM USAGE BY SOS**

The Apple /// operating system (SOS = Sophisticated Operating System or Sara's OS) uses several ROM routines. These routines seem to all be related to disk block I/O. The following discussion is based on SOS version 1.3.

When the ROM loads block 0 from a SOS disk the ROM is loading the SOS Bootstrap Loader program. This program, which is at most 512 bytes in length, uses the ROM routine REGRWTS (F000) to read the SOS Loader into memory. This program does not test the ROM revision. It is interesting to note that ROM routine BLOCKIO is not used, instead a lower-level routine (REGRWTS) is used.

The SOS Loader determines if the ROM is revision 1 by comparing address F1B9's contents against A0 (reference: SOS source file SOSLDR.D.SRC). If this comparison fails then SOS displays on the screen the error "ROM ERROR: PLEASE NOTIFY YOUR DEALER." If the ROM revision is correct then the SOS loader uses the ROM's disk I/O routines to read more of SOS into memory.

The disk /// driver that is built into SOS also uses the ROM to perform disk block I/O (reference: DISK3.SRC). It is interesting to note that when the disk driver is initialized the driver checks if the ROM revision is 0 or 1. A revision of 0 is detected if address F1B9 contains 60. If neither revision is found then the disk driver returns an error to SOS (I don't think this will ever happen since the SOS loader has already determined that the ROM is revision 1). For a valid ROM revision the disk driver sets up several jump vectors which point to the appropriate addresses in the ROM for the various ROM routines needed by the disk driver. Therefore, the disk driver seems compatible with either ROM revision whereas the SOS loader likes only revision 1.

The .CONSOLE driver source listing appears to not use any ROM routines even though the ROM contains 40 and 80 column text routines and keyboard input routines. I assume the console driver was much more sophisticated than the ROM's text features and so using the ROM routines would not have worked well for this driver. I also assume that if the console driver used the ROM that when ROM

revision 1 was built the console driver would have had to be changed and Apple (smartly) did not want to do this.

## 6 MONITOR COMMANDS

Holding down the Open Apple and Control keys when the /// starts or when you press the Reset key activates the /// ROM Monitor. The screen will display in the upper left corner a small right-facing arrow with a blinking underscore character as the cursor. The Monitor's commands are based on the Apple ]['s Monitor commands but some commands have changed slightly and others are new for the (newer) ///.

The Monitor supports the following commands:

addr1.addr2	Dump memory data to screen from address 1 to address 2 and display ASCII character at the right of the screen.
CARRIAGE RETURN	Dump next line of addresses to the screen.
SPACE	Pause current memory dump. Press again to continue.
addr:byte_list	Store starting at the address the list of bytes.
addr:'text'	Store text starting at address with high bit clear.
addr:"text"	Store text starting at address with high bit set.
addr3<addr1.addr2M	Move data in addresses 1-2 to address 3.
addr3<addr1.addr2V	Verify data in addresses 1-2 is the same as data starting at address 3.
byte<addr1.addr2S	Search memory in address range 1-2 for the byte.
block<addr1.addr2W	Write address range to disk starting at the disk block.
block<addr1.addr2R	Read disk starting at block to the address range.
addrG	Call subroutine at the address.
addrJ	Jump to the address.
U	Call user routine starting at address \$03F8.
X	Repeat last command line until you press the SPACE BAR.
ESC-8	Display 80 columns of text.
ESC-4	Display 40 columns of text.
/	Seperate multiple commands on the same line.
CTRL-I	Interrupt current operation, return to Monitor command line.

Note: See Wells' *Apple /// Entry Points* article for a great overview of the ROM Monitor, its commands (with some syntax errors), and the memory locations that need setting up for the key ROM routines to work. Apple's */// Service Reference Manual* (p. 13.57) has a list of Monitor commands. Anderson's *The Apple Nobody Knows* also has good Monitor command info.

To obtain a binary dump of the /// ROM you can do the following:

1. Initialize a disk on either the /// or an Apple ][ computer.
2. Insert the new disk in the ///.
3. Start the /// and hold down the Open Apple and Control keys.
4. You should be in the /// Monitor.
5. Type 0<F000.FFW to write the ROM to disk blocks 0 to 7
6. Use a disk block reader on the /// or the ][ to read the ROM blocks and save them to a real file.

This disk writing is needed since the ROM does not provide a command for redirecting screen output to the ///'s serial port. But, I've read that you can output the ROM contents to the ///'s serial port but this involves using the Monitor to write a small program. If anyone has such a program please send a copy my way.

## 7 A FEW COMMENTS

I find it interesting, at least from a software engineering perspective, to note that in my opinion the /// ROM is missing several key features which I thought any system ROM would need. The ROM is missing two features which I think would have been useful to Apple and outside /// programmers:

- 1) The ROM does not have an explicit version number which exists at a specific ROM address. This version number could be used to validate the ROM in case there were several different ROMs (as there were). Apple uses a pseudo ROM version number (called the revision number) during the loading of SOS but this is somewhat lame in my opinion.
- 2) The ROM does not have a dispatch routine for use by the OS or applications that want to use ROM routines. This dispatch routine would reside at a specific address (e.g., F000) and it would take as input a command number and a set of parameters. These parameters could be passed via registers or on the stack. This routine would allow Apple to change the ROM and ROM "users" would not need to change their programming as long as they used the selector routine. The Apple ][ ROM did not have such a routine which caused Apple many headaches when it wanted to change the Apple ][ ROM and had to keep lots of routines in their same place.
- 3) The ROM source code is rather sparse concerning comments. It would be nice if the ROM contained detailed information about what each routine did and how to call the routines. Obviously, Apple did not expect anyone but Apple's own programmers to ever see the ROM source or use the ROM routines. (I've seen the Lisa computer's ROM listing which is much better documented than the ///'s and both are comparable in terms of age).



## 8 REFERENCES

### *Apple /// ROM Listing - Revision 0*

This can be found in the Apple /// patent (#4,383,296) dated May 1983. Note that in places this ROM listing is not always readable.

### *Apple /// ROM Listing - Revision 1*

I have a very readable listing of the revision 1 ROM that was printed on a laser printer.

### *Apple /// Service Reference Manual (Level 2)*

This almost 500 page document by Apple has everything you would want to know about the ///'s hardware, low-level software, and how to service a broken ///. Includes descriptions of the System Monitor (a.k.a. Development Monitor) [page 17.3] and the built-in RAM test routine [page 13.51].

### *Apple /// SOS Bootstrap Loader Listing*

Shows how 512 bytes of code are used to load SOS from disk into the ///'s memory.

The following articles provide good ROM information:

*Apple /// Entry Points*, Andy Wells, Call-APPLE, October 1981

*Apple /// Dabbling*, Rick Smith, Apple Orchard, Summer 1981

*/// Bits: John Jeppson's Guided Tour of Highway ///*, John Jeppson, Softalk, May 1983

*The Apple Nobody Knows*, Alan Anderson, Apple Orchard, Fall 1981

*Unlocking the Apple /// - Part 3*, Alan Anderson, Apple Orchard, September 1982

*Apple ///: 12-Volt 128K Internal Diagnostics*, Apple Technical Information Library

## 9 DOCUMENT MODIFICATION HISTORY

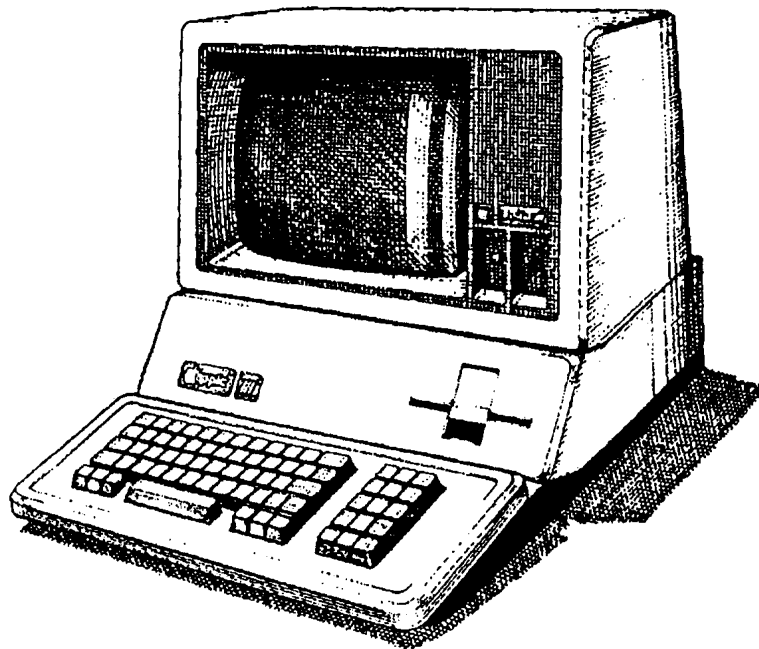
30 Nov 1997      Created this document.

04 Dec 1997      Corrected a few problems, extended the Reference section to include more /// articles pertaining to the /// ROM, added this section, added section MONITOR COMMANDS.

###



# Apple III Computer Information



## Inside the Apple III ROM

Revision 1 • 30 Nov 1997

---

## Inside the Apple /// Computer ROM

---

David T. Craig • 30 November 1997  
71533.606@compuserve.com

### TABLE OF CONTENTS

- 1 INTRODUCTION
- 2 ROM SECTIONS
- 3 IMPORTANT ROM ROUTINES
- 4 ROM TABLES
- 5 ROM USAGE BY SOS
- 6 A FEW COMMENTS
- 7 REFERENCES

### 1 INTRODUCTION

This document provides a general overview of the contents of the Apple /// computer ROM revision 1. This information should be used in conjunction with a copy of the ROM source code listing. The audience of this document is anyone with an interest in the technology of the Apple /// computer's hardware and software.

#### NOTE

There were two revisions of the Apple /// ROM, revision 0 and revision 1. Revision 0 ROMs had at address F1B9 the value 60. Revision 1 ROMs had at address F1B9 the value A0.

This ROM contains 4 KB of 6502 programming and several data tables. The ROM occupies memory addresses F000-FFFF. The basic purpose of the ROM is to test the Apple /// computer hardware and boot an operating system from the ///'s built-in floppy disk drive. The ROM also contains a simple Monitor program whose purpose is to allow the user to interact with the /// at the hexadecimal level.

When the Apple /// computer is turned on the ROM's flow of execution is as follows:

- 1) The ROM starts execution at the address contained in FFFC-FFFD (RESET) which is address F4EE (DIAGN).
- 2) Diagnostics (DIAGN/F4EE) starts. The diagnostic first initializes some memory for the ROM's use. If the Open Apple key is held down then enter the ROM Monitor. Otherwise run several diagnostic checks of the /// hardware (tests zero page, sizes memory, initializes screen text buffer,

tests stack memory, tests ROM checksum, tests VIA chip, tests ACIA chip, tests A/D circuitry, tests keyboard connection). Any diagnostic failures display an error message and the user has to reset the computer.

- 3) Read block 0 (512 bytes) to address A000 from the floppy disk in the built-in disk drive (BOOT/F6A1). If no disk is found or block 0 cannot be read then display "RETRY" and wait for the user to reset the computer. If the block is successfully read then execute the block contents (this is called the SOS Bootstrap Loader: see section ROM USAGE BY SOS).

## 2 ROM SECTIONS

Section	Address	Purpose
Disk I/O	F000-F4C4	Read and write floppy disk blocks (512 bytes each)
Diagnostics	F4C5-F7FE	Diagnose the /// hardware
Monitor	F7FF-FFFF	Interacts with user so user can do simple things

## 3 IMPORTANT ROM ROUTINES

BLOCKIO / F479	Reads or write a disk block (512 bytes), calls routine REGRWTS (F000) which reads a sector (256 bytes) from the disk
BOOT / F6A1	Read floppy disk block #0 into address A000, execute the block
ENTRY / F901	Monitor entry point
DIAGN / F4EE	Diagnostic entry point
USRENTY/F6E6	Tests RAM and displays a table showing chip failures (users may execute this routine from the Monitor)

## 4 ROM TABLES

Here's a list of the important data tables in the ROM. This list does not include disk I/O tables.

Table Name / Address	Contents
CHRSET / FEC5-FFB3	Default character set (overridden when SOS loads the character set from SOS.DRIVER)
Copyright / FFC0-FFEF	Copyright message (contains the initials "JRH" for J. R. Huston who was a key player behind the /// and SOS)
NMI / FFFA-FFFB	Jump address for the Non-Maskable Interrupt signal
RESET / FFFC-FFFD	Jump address when the /// is powered on

IRQ / FFFE-FFFF

Jump address for the Interrupt Request signal

## 5 ROM USAGE BY SOS

The Apple /// operating system (SOS) uses several ROM routines. These routines seem to all be related to disk block I/O. The following discussion is based on SOS version 1.3.

When the ROM loads block 0 from a SOS disk the ROM is loading the SOS Bootstrap Loader program. This program, which is at most 512 bytes in length, uses the ROM routine REGRWTS (F000) to read the SOS Loader into memory. This program does not test the ROM revision. It is interesting to note that ROM routine BLOCKIO is not used, instead a lower-level routine (REGRWTS) is used.

The SOS Loader determines if the ROM is revision 1 by comparing address F1B9's contents against A0 (reference: SOS source file SOSLDR.D.SRC). If this comparison fails then SOS displays on the screen the error "ROM ERROR: PLEASE NOTIFY YOUR DEALER." If the ROM revision is correct then the SOS loader uses the ROM's disk I/O routines to read more of SOS into memory.

The disk /// driver that is built into SOS also uses the ROM to perform disk block I/O (reference: DISK3.SRC). It is interesting to note that when the disk driver is initialized the driver checks if the ROM revision is 0 or 1. A revision of 0 is detected if address F1B9 contains 60. If neither revision is found then the disk driver returns an error to SOS (I don't think this will ever happen since the SOS loader has already determined that the ROM is revision 1). For a valid ROM revision the disk driver sets up several jump vectors which point to the appropriate addresses in the ROM for the various ROM routines needed by the disk driver. Therefore, the disk driver seems compatible with either ROM revision whereas the SOS loader likes only revision 1.

## 6 A FEW COMMENTS

I find it interesting, at least from a software engineering perspective, that the ROM is missing some key features which I thought any system ROM would need. The ROM is missing two features which I think would have been useful to Apple and outside /// programmers:

- 1) The ROM does not have an explicit version number which exists at a specific ROM address. This version number could be used to validate the ROM in case there were several different ROMs (as there were). Apple uses a pseudo ROM version number (called the revision number) during the loading of SOS but this is somewhat lame in my opinion.
- 2) The ROM does not have a selector routine for use by the OS or applications that want to use ROM routines. This selector would reside at a specific address (e.g., F000) and it would take as input a command number and a set of parameters. These parameters could be passed via registers or on the stack. This routine would allow Apple to change the ROM and ROM

"users" would not need to change their programming as long as they used the selector routine. The Apple ][ ROM did not have such a routine which caused Apple many headaches when it wanted to change the Apple ][ ROM and had to keep lots of routines in their same place.

## 7 REFERENCES

### *Apple /// ROM Listing*

I have a very nice listing of revision 1 ROM. A listing (that is somewhat readable) for the earlier revision 0 ROM may be found in the Apple /// patent.

### *Apple /// Service Reference Manual (Level 2)*

This almost 500 page book by Apple has everything you would want to know about the ///'s hardware, low-level software, and how to service a broken ///. Includes descriptions of the System Monitor (a.k.a. Development Monitor) [page 17.3] and the built-in RAM test routine [page 13.51].

### *Apple /// SOS Bootstrap Loader Listing*

Shows how 512 bytes of code is used to load SOS from disk into the ///'s memory.

\*\*\*



Apple /// Computer Technical Information

## **SOME COMMENTS ABOUT THE APPLE /// COMPUTER BOOT ROM**

David T Craig -- 27 February 2004

### **BACKGROUND**

The Apple /// computer was introduced by Apple Computer in 1980 and was discontinued in 1985.

This computer was a microcomputer with originally 128 KB of RAM memory expandable to 256 KB of RAM. It featured a 4 KB ROM (addressed from \$F000 to \$FFFF hexadecimal) which housed the initial programming that executed when the user turned on the computer. This ROM contained programming for the following functions:

- + diagnose hardware circuitry and memory
- + load and run a disk operating system (i.e. "boot")
- + provide an interface to a simple monitor program

The author wrote these comments after looking at the Apple /// ROM listing as found in Apple Computer's patent number 4,383,296 dated 10 May 1983. This analysis occurred during a scanning of the Apple /// patent.

### **ROM COMMENTS**

The Apple /// patent's ROM program listing is terrible in terms of printed quality. Many parts are very faint and impossible to read. I assume this was done on purpose by Apple's legal department so that Apple's competitors would not be able to duplicate this ROM programming easily.

The ROM programming does not seem to have been built for expansion. By this I mean the programming seems to have been written to just make it work and no long term thought was given to the ROM programming's organization.

There were two versions of the ROM. The Apple /// operating system (OS) programming needed to differentiate between the ROM versions since the ROM contained several routines which the OS used. This version determination was not done in a logical way. A memory location was chosen at random (at least it seems this way to me) to serve as the ROM's "version number". The OS had to test this "version number" when it needed to use specific ROM services.

The ROM version also determined the location of several ROM routines which the Apple /// OS used.

The ROM's organization could have been improved greatly in my opinion if it was organized differently. At the beginning of the ROM address space (\$F000) include a short header containing the following:

- \$F000 - ROM version number
- \$F001 - ROM size (K bytes)
- \$F002 - ROM checksum (2 bytes)
- \$F003 - ROM routine dispatch jump vector (3 bytes)
- \$F006 - ROM copyright notice (e.g. "(c) Apple Computer 1980")

The remainder of the ROM would have contained whatever programming and table data was needed.

The routine dispatch jump vector would be a standard jump instruction to a routine in the ROM whose purpose would be to let outside programs such as the operating system, device drivers, or even application programs access ROM routines in a ROM version independent manner. The dispatch routine would take as input a command number (in say the CPU's A register) and return result information in the CPU's X and Y registers. The A register on return would contain an error result with 0 meaning no error. Or, some fixed memory area could be use to handle ROM routine parameters. This dispatch mechanism could be seen as a BIOS (basic input output system).



Possible dispatch routines could be:

- + Restart or Cold start or Warm start the computer
- + Read a block from a disk drive
- + Write a block to a disk drive
- + Return size in blocks of a disk drive
- + Checksum the ROM for diagnostic purposes
- + Test computer's RAM memory for diagnostic purposes
- + Enter the Apple /// Monitor program

This dispatch mechanism would have simplified the Apple /// OS use of the ROM services since the ROM would always be accessed from just one address (\$F003). If the OS requested a ROM service which was unavailable (e.g. an old ROM was installed) then the ROM would tell the OS that the service did not exist via a dispatch error result.

## CONCLUSION

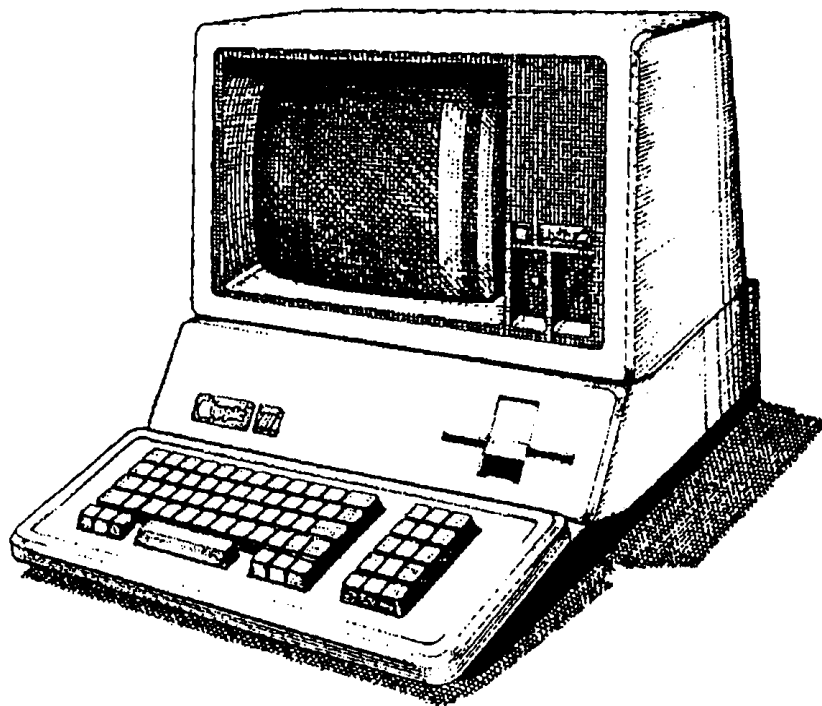
Hopefully this little commentary provides some useful information to its reader. If you are interested in the Apple /// computer you should see its patents (one is for the Apple ///, the other is for the Apple /// Plus). The first patent contains the full ROM listing, but the author has a real digital copy which is much more readable.

Enjoy.

###



# Apple III Computer Information



## Apple III Emulator Ideas

Version 4 • 12 Dec 1997



## SOME IDEAS ABOUT AN APPLE /// COMPUTER EMULATOR

David T. Craig -- 12 December 1997 -- Version 4

941 Calle Mejia #1006, Santa Fe, NM 87501 USA  
e-mail: 71533.606@compuserve.com

### TABLE OF CONTENTS

1.0	PURPOSE
2.0	EMULATOR GOALS
3.0	EMULATOR USER INTERFACE
4.0	DISK IMAGES
5.0	6502 CPU EMULATION
6.0	ROM EMULATION
7.0	MEMORY-MAPPED I/O EMULATION
8.0	MEMORY BANK SWITCHING EMULATION
9.0	SOS SYSTEM CALL EMULATION
10.0	DEVICE DRIVER EMULATION
11.0	KEYBOARD SUPPORT
12.0	MONITOR SUPPORT
13.0	APPLE ][ EMULATION DISK SUPPORT
14.0	WHAT LANGUAGE SHOULD THE /// EMULATOR BE WRITTEN IN?
15.0	WHAT TARGET MACHINES SHOULD BE SUPPORTED?
16.0	EMULATOR DEBUGGING FACILITIES
17.0	EMULATOR MEMORY STRUCTURE
18.0	WHAT'S NEXT?
19.0	REFERENCES

### MODIFICATION HISTORY

28 Nov 1997 -- Version 1  
Created by David T. Craig.

#### 04 Dec 1997 -- Version 2

New sections: MONITOR SUPPORT, EMULATOR DEBUGGING FACILITIES.

Updated sections: DISK IMAGES, MEMORY BANK SWITCHING EMULATOR, SOS SYSTEM CALL EMULATION, REFERENCES.

Added several good comments by Chris Smolinski (he's writing a /// emulator called SARA).

#### 09 Dec 1997 -- Version 3

DISK IMAGES: Updated info about DTCMake3///DiskImage Mac application, made disk image file an all-text file.

SOS SYSTEM CALL EMULATION: typo Silentypr --> Silentye.

WHAT TARGET MACHINES SHOULD BE SUPPORTED: More pre-68040 Mac comments.

EMULATOR DEBUGGING FACILITIES: typo affects --> affect, added info about enabling/disabling SOS BRK disassembly, same for ProDOS, added list of emulator debugging commands.

EMULATOR MEMORY STRUCTURE: New section.

#### 12 Dec 1997 -- Version 4

EMULATOR DEBUGGING FACILITIES: Added examples to every debugging command. Added commands SNAPSHOTW, SNAPSHOTR, ZPAGE, SPAGE, EPAGE, DRIVERS, macro commands.

disk:  
new: READFILES

read all  
files from  
disk to  
host computer  
disk

Corrections -- by page #

1 - Mod. History -- add 2 spaces

9 - "C" char set bank --> "K"

10 - -- : add extra space between  
all build command  
line words.

11 - HAPP name

10 - change RD cmd to show

(15) P and E bit names (upper case  
= 1, lower = 0)

12 - ZPAGE - show offset byte line,  
same for adr1, adr2

13 - S cmd

14 - DISK BUFFER

15 - SS

16 - BPE more general

16-17 del some BPE words

21 - BPNE FONT FONTROM



## 1.0 PURPOSE

This document describes some ideas about implementing a software emulator for the Apple /// computer. These ideas are based on my experiences with the Apple /// computer and its software programming. No specific target machine is mentioned in this document since these ideas should be non-target machine specific. These ideas are submitted to stimulate thought about such an emulator and hopefully inspire someone to produce a working Apple /// emulator.

The technical details behind the Apple /// computer, its operating system (SOS), and /// programs (e.g. AppleWriter ///) are based on my extensive collection of /// technical manuals, specification sheets, and many /// technical articles (Dr. John Jeppson's articles are very exhaustive and full of lots of neat /// techoid stuff). I have around 15 Apple manuals, the majority of which were published by Apple, which include user manuals and the technical programming manuals.

For those people seriously interested in implementing an Apple /// emulator program I highly recommend that they have at least the Apple /// Service Reference Manual. This manual, which is almost 500 pages long, is the definitive reference for how the Apple /// computer works. Most of its contents describe theory of operation even though its title suggests service-type information only. The important features of this manual for a /// emulator writer are the /// memory map and the /// memory mapped I/O locations.

I also own an Apple /// computer which still today works very well. I programmed the /// many moons ago and have worked professionally as an Apple Macintosh computer programmer since 1984.

Note: All comments are welcome. If you have anything to add or correct please let me know and I will update the master copy of this document.

## 2.0 EMULATOR GOALS

The /// emulator should provide a complete emulation environment for the faithful execution of Apple /// and /// Plus programs. As far as the emulator user is concerned when they run the emulator program their computer should work just like an Apple /// computer and all /// visual fidelity should be maintained. Emulation of the Apple /// Plus computer may also be supported (this means the /// Plus' interlaced screen). If the /// Plus is supported by the emulator you may want to let the user specify if they want to run a /// or a /// Plus.

I think it would be beyond neat if the emulator could run Apple's running horses demo and the other /// demos.

The /// emulator should support an Apple /// computer with at least 256K of memory and four floppy 140K disks (.D1, .D2, .D3, .D4). Support for 512K of memory may also exist since the ///'s operating system (SOS) supports up to 512K of memory. Memory size, if variable, should always be a multiple of 32K. I believe the lowest memory size supported by the /// (ROM?) is 96K. Support for a ProFile disk may also exist (for this disk there would need to be a disk image with a size of 5M). The first floppy disk (.D1) would correspond to the floppy disk drive that is built into the Apple ///. The other disks correspond to external disks and should exist as image files with specific file names (e.g. "Apple 3 D1", "Apple 3 D2", etc). The ProFile disk image file should also have a specific file name (e.g. "Apple 3 ProFile").

Image file names should have an extension (e.g. ".D3I") since this is needed by PCs.

## 3.0 EMULATOR USER INTERFACE

When the user runs the Apple /// emulator program the user should see on their computer screen a screen (or a window representing the screen on GUI systems) corresponding to the ///'s screen which the user would see if they were in front of a real Apple /// computer. All /// text and graphic modes should be supported by the



/// emulator (this includes the special modes supported by the /// Plus and its interlaced screen architecture).

I recommend that the emulator also support a screen dump facility that writes the current /// screen to either a text file (for text modes) or to a graphic file (for graphic modes) or always just creates a graphic file. The screen dump graphic file should be a standard graphic file for whatever target machine your support (e.g. on the IBM PC running Windows produce .BMP files, on the Apple Macintosh produce PICT files). Since the /// supports custom character sets dumping the screen to a PICT file (or to the target computer's clipboard) may be the best solution.

The emulator screen if implemented in a GUI window may also display a status area at the bottom of the window. This status area would display at least two lines of text and would keep the user informed of what the emulator was doing internally.

#### 4.0 DISK IMAGES

The /// emulator should read disk image files which correspond directly to real /// 140K disks. When the /// emulator starts it should look in its folder and if there exists a /// disk image file the emulator should boot this image. If there are multiple disk image files then the emulator may want to display a list of these images and have the user select an image to boot.

The disk images should be exact copies of real /// disks. To make copies of these disks there should exist an utility program that runs on the /// computer and which outputs disk block data to the /// serial port (I plan to make this utility and call it DTCDumpIt). This utility's output should be a hex/ascii dump that specifies block numbers and has a checksum for each line of data. This utility should ask the user if it should dump a file or a disk.

On the target machine there should exist a similar utility that inputs the disk block data and creates a disk image file. I recommend that the transmitted disk block data consist of a hex dump with block number and checksum information in a human readable fashion. The receiving program (on the target computer) would read this human readable information, verify that the data was sent correctly, and produce binary disk image file images (I plan to create this utility for the Apple Macintosh and call it DTCMake///DiskImage).

There should also exist a disk image file for the ///'s Boot ROM (recommended file name: "Apple 3 Boot ROM"). This image should contain the 4K ROM image. This ROM should be the Revision 1 ROM (not Revision 0) since this was the last ROM produced and SOS 1.3 (the last SOS) requires this ROM.

Users should also be able to format a disk image by specifying the disk drive device name (e.g. .D2). Users should then be able to name the disk image so that they can use it later. Users should be able to assign specific disk images to specific disk drives.

I recommend that all disk image files have a very specific internal format. This format should support the verification of disk image files so that if a disk image file becomes corrupted in some fashion the /// emulator can detect this corruption, not use the image, and alert the user.

Note: Support for existing Apple ][ disk image files may be feasible but I recommend against this since the format of these images could change.

The proposed image format:

The disk image file contains two parts, a header part and a data part. The header part appears first followed by the data part. The header part contains identification and verification information. The data part contains the actual disk blocks for the /// disk. This file contains only text, no binary data appears here in any fashion. The only non-text information that can appear in these files is the Carriage Return (CR) and the Line Feed (LF) characters. The emulator should ignore



LFs if appropriate. All information appears in lines with a maximum length of 255 characters. Character case is immaterial. Blank lines are ignored. The reason for this format is so these image files can be transferred over the internet without the need for any binary-to-text conversion. Also, text-only files can easily be viewed by people using a word processor.

The header part contains:

Line	Comments
Signature	"APPLE /// DISK IMAGE"
Version	"VERSION" version number (e.g. "1")
Image Name	"IMAGE NAME" name of image, anything the user wants, most likely the name of the interpreter on the disk, e.g. "Apple Writer ///"
Creation Date	"CREATED" date image file created, "YYYY-MM-DD"
Created by Name	"CREATED BY" name of person or company who created this image
Comment	"COMMENT" comment for anything user wants
Data Size	"DATA SIZE" size of data part (decimal, e.g. "143360")
Data Checksum	"DATA CHECKSUM" hexadecimal checksum (e.g. "FA7C3188")
Reserved 1	"RESERVED"
Reserved 2	"RESERVED"
Reserved 3	"RESERVED"
Reserved 4	"RESERVED"
Tech Comment	"TECH COMMENT" name of program that this is for
Header Checksum	"HEADER CHECKSUM" hexadecimal checksum (e.g. "B97C31D5")

#### Notes:

The checksum should be calculated as the exclusive-OR of each byte followed by a left rotation of 1 bit. Checksum starts with zero. Checksums should always be 4 bytes in size and be stored in the header as an 8 character string.

The Tech Comment's purpose is to allow people who obtain an image file to be able to contact someone about the file's purpose.

The data part contains lines representing 16 bytes from the original disk. Each line has a specific format which begins with the starting disk address for the line, 16 bytes, the ASCII equivalent of the 16 bytes, and a checksum for the bytes of the line with the format:

```
[00000000] 0123 4567 89ab cdef 0123 4567 89ab cdef [1234567890123456] 12345678
```

The last line of the file must be the word "FINIS".

Sample disk image file:

```
APPLE /// DISK IMAGE
VERSION 1
IMAGE NAME Apple Writer ///
CREATED 1997-10-11
CREATED BY David T. Craig
COMMENT Thanks to Paul Lutus
DATA SIZE 16
DATA CHECKSUM FA7C3188
RESERVED
RESERVED
RESERVED
RESERVED
TECH COMMENT For David Craig's /// Emulator - 71533.606@compuserve.com
HEADER CHECKSUM B97C31D5

[00000000] 0123 4567 89ab cdef 0123 4567 89ab cdef [Apple.///.Emul..] FA7C3188
```

Some Ideas about an Apple /// Computer Emulator -- Version 4  
David T Craig -- 12 Dec 1997 -- 4 / 23



FINIS

## 5.0 6502 CPU EMULATION

The heart of the /// emulator should be the emulation of the 6502 CPU. The heart may be referred to as the "6502 engine." The emulator should support all of the 6502 instructions, the 6502 registers, and the special Apple /// registers (e.g. the bank switch register, the environment register, and the zero-page register). Special register descriptions and usage can be found in the Apple /// SOS Reference Manual.

The 6502 engine must be smart about accessing memory and use the bank switch and environment registers correctly.

If this level of the /// emulation is complete and robust the rest of the /// emulator should work much more easily.

Support for special /// features may also exist at this level of the /// emulator. For example, the /// emulator may not want to emulate all of the ///'s memory-mapped I/O features, but instead intercept access to special areas or routines and call the target machine's operating system to handle these features. See sections ROM EMULATION and MEMORY-MAPPED I/O EMULATION for more details.

## 6.0 ROM EMULATION

The /// emulator should also support as much as possible the ///'s Boot ROM. This means the Boot ROM's routines should work for the most part as-is.

Note: I have a listing of the Boot ROM which could be useful for this emulation discussion.

For the Boot ROM's floppy disk I/O support I recommend that all the gory details here not be supported directly at the memory-mapped I/O level but instead the /// emulator should emulate this I/O. Specifically, the /// emulator should intercept any access to the Boot ROM routines which read or write disk blocks and use the appropriate target machine operating system routines to accomplish this feature.

The /// emulator should also initialize the ROM's character set which the ROM normally loads into a special RAM chip that is not accessible to the ///'s 6502 processor. See section MEMORY BANK SWITCHING EMULATION for more details.

## 7.0 MEMORY-MAPPED I/O EMULATION

All memory-mapped I/O locations that in some way deal with the physical world need to be handled by the /// emulator. These areas include such addresses as the speaker addresses. The Apple /// Service Reference Manual provides detailed information about these addresses.


All accesses to memory by the /// emulator must respect the bank switch and environment register settings so that the emulator does not try to access a memory-mapped address when that address is not mapped into the 6502 address space.

Programs which access low-level I/O locations such as the disk I/O addresses should not be supported. I assume most /// programs will access hardware components using SOS or device drivers.

Note: Chris Smolinski says that emulating the low-level stuff on a Power PC-based Macintosh is not very difficult and works rather fast (he's implemented in his SARA emulator the ///'s floppy disk I/O).

## 8.0 MEMORY BANK SWITCHING EMULATION

The /// emulator must also fully support the ///'s bank switched and enhanced indirect addressing memory architecture. Detailed descriptions and usage of /// memory handling can be found in the Apple /// SOS Reference Manual.

Some Ideas about an  Apple /// Computer Emulator -- Version 4  
David T Craig -- 12 Dec 1997 -- 5/23



The /// emulator should also support the ///'s character set RAM chip. This holds the bitmap descriptions of each of the 128 characters in the /// character. This RAM area, which is not accessible to the ///'s 6502 CPU, holds 1024 bytes. See the Apple /// Standard Device Drivers Manual (Console Character Sets section) for more information.

Note: I believe the storage of the Boot ROM character set is different than the storage of the character set in the SOS.DRIVER file. I believe the ROM character set has bits that are reversed compared to the SOS.DRIVER character set.

The storage of text and graphics in memory should be supported also. This should happen automatically when a /// program writes to the text/graphic memory buffers. The emulator needs to detect such writes and update its screen as appropriate.

## 9.0 SOS SYSTEM CALL EMULATION

The majority of system calls to SOS and its drivers should most likely not be intercepted by the /// emulator. But certain calls may need to be intercepted unless a lower level of the /// emulator intercepts these feature already. System calls to SOS or drivers that may need intercepting by the /// emulator could be:

- o Disk I/O (.D[1-4] and .PROFILE drivers)
- o Keyboard I/O (.CONSOLE driver)
- o Screen I/O (.CONSOLE and .GRAPHIC drivers)
- o Sound generation (.AUDIO driver)
- o Serial port I/O (.RS232 driver)
- o Silentye Printer (.SILENTYPE) [I'm not sure about support for this]
- o Clock I/O (Y2K dates may be a problem)

I recommend that the /// emulator intercept all activity dealing with the above and have the target machine perform the equivalent features. For example, to read or write a disk block the /// emulator should have a routine that accesses the appropriate location in the disk image file.

The /// emulator may also provide the user with some type of setup options so that the user can specify specific properties of some of the above drivers. For example, if the target machine supports several output ports the emulator may let the user specify which port to use (e.g. for the .PRINTER driver the user could assign it to a specific serial or parallel port on the target machine).


Note: The ///'s clock does not support the year 2000 or greater. I think the emulator should support Y2K dates but I'm not sure if SOS's file system date stamps will support this easily.

## 10.0 DEVICE DRIVER EMULATION

This section is for the most part handled by my comments in section SOS SYSTEM CALL EMULATION. I suspect the programming within the /// emulator for this area could be the most work since there are lots of device drivers that make up a simple Apple /// configuration.

One area of device drivers that the /// emulator may not want to emulate is interrupt handling. Since the emulator does not have physical devices connected to it in any direct fashion I don't think interrupts exist as far as the emulator is concerned. Interrupts dealing with disks or the keyboard can be handled at a lower level by having the /// emulator call the appropriate system call in the target machine. These low-level I/O handlers should set up the appropriate driver data areas so that the rest of the ///'s software (SOS and the interpreter) will work correctly. For example, keyboard I/O should be setup in the /// emulator so that when the keyboard input memory-mapped I/O location is accessed the target machine OS really reads the keyboard and sets up the memory-mapped location as appropriate.

## 11.0 KEYBOARD SUPPORT

Some Ideas about an  Apple /// Computer Emulator -- Version 4  
David T Craig -- 12 Dec 1997 -- 6/23





#### 11.1 User interface support

The /// computer's keyboard layout is basically compatible with modern keyboards. The /// keyboard does have two extra keys, Open Apple and Closed Apple which are positioned to the left of the Apple /// keyboard. Also present on the keyboard are four arrow keys. The emulator should support these keys either directly (i.e., the target machine has similar keys) or associate other keys with the ///'s special keys (e.g., the Macintosh computer's two Option keys could be used to simulate the special Open and Closed Apple keys). The emulator's associated keys need not physically be in the same location as the ///'s special keys but having them in the general area will be beneficial.

Note: The /// Plus keyboard contains an extra key, Delete, compared to the /// keyboard.

#### 11.2 Low-level access

The /// emulator should handle low-level access to the keyboard memory-mapped I/O locations as detailed in section DEVICE DRIVER EMULATION.

### 12.0 MONITOR SUPPORT

The emulator should support the Apple's built-in ROM Monitor. Entry to the Monitor should be similar to how this is done on a real /// (at startup if Open Apple and Control keys are pressed). The code in the ROM which tests for Monitor entry should work.

### 13.0 APPLE ][ EMULATION DISK SUPPORT

It would be nice if the /// emulator supported the Apple ][ Emulation Disk. I'm not sure of what would be involved here but suspect that if the ///'s 6502 CPU and the memory-mapped I/O locations are robustly supported that the ][ emulation should work also without any special additional /// emulation features.

Special consideration may need to be given to Apple /// keyboard keys which do not exist in the Apple ][ world. ][ emulation details can be found in the Apple /// Owner's Guide and the Apple /// Service Reference Manual.

Note: I have a disassembled listing of the Apple ][ Emulation Disk ROM source listing which could prove useful in this area.

Further analysis of the ][ emulation disk's boot sequence needs to be done since I'm unknowledgable about this area. Also, I've heard that the ][ emulation accesses an I/O location which disables some /// features.

### 14.0 WHAT LANGUAGE SHOULD THE /// EMULATOR BE WRITTEN IN?

I highly recommend that the /// emulator be written in a high level language such as Pascal or C. This should make the emulator more compatible with different target computers and make development and maintenance of the emulator much easier. I recommend avoiding low-level languages such as assembly.

### 15.0 WHAT TARGET MACHINES SHOULD BE SUPPORTED?

I recommend that the target machine (or machines) for the emulator be machines that are commonly used today by most computer users. This means either the IBM PC or the Apple Macintosh machine family. For the PC world I recommend the /// emulator run under Windows 95 and Windows NT. For the Macintosh world I recommend the emulator run on most Macintosh models which means support the Macintosh 512 and above. Color display should also be supported by the /// emulator (for the Macintosh this means use Color QuickDraw if the machine supports CQD and if CQD is not supported by a Macintosh model use the Classic B/W QD and maybe use patterns as "colors").



Any of these machines should be fast enough to emulate the /// and most likely will be too fast in many areas. I recommend some type of speed control be built into the emulator so that users can control how fast the emulator works. For many /// programs (e.g. AppleWriter /// and VisiCalc ///) emulation speed will be immaterial since these programs typically wait for the user to enter data and then do their thing. But for programs such as games the user will want to control the emulator speed otherwise the game's actions will be super fast and unplayable.

Some people say that the older machines such as pre-68040 Macintoshes will be too slow for a reasonable /// emulator. I would like to see this /// emulator run on a Mac 512 machine and onwards. Running on a Mac 128 machine seems a problem due to this machine's small memory size and should not be supported (if a virtual memory scheme was used by the emulator the Mac 128 could be supported but I think having this extra level of support in the emulator would not be worth it). I disagree and am willing to wager a small sum that I'm right.

#### 16.0 EMULATOR DEBUGGING FACILITIES

The emulator should support a comprehensive built-in debugger. This debugger's purpose should be to let the sophisticated emulator user access any part of the emulator's /// address space. This should include all of the memory that is allocated to the /// as its memory. This memory would encompass the 256K (or 512K) of /// RAM, the /// ROM (4K), the character set RAM (1K), the 6502 registers, and the special /// registers (e.g. bank register).

This debugger will prove invaluable in diagnosing emulator bugs. Not only will the user be able to type commands for the debugger but the emulator will be able to send messages to the debugger.

Logging of all debugger sessions should be stored to a text file for possible analysis. This text file would be created when the emulator starts. The log file should be appended to by the emulator. Only the user can delete the file.

The debugger should exist as a separate window that does not in any way affect the emulator's main window. This window should display only commands that the user enters or replies returned by the debugger. There should not exist a separate window area showing things such as the 6502 registers since all such information should appear in the debugger log file. The window should support at least 80 columns of text and 24 rows.

The emulator user interface should be based on a simple command line control scheme. All commands and command outputs should be text-based. This scheme could be based on the ///'s Monitor's commands or on a little more readable command scheme such as in Apple's MacsBug debugger. There should be full on-line help that discusses the debugger commands in general and each command should also have on-line help available. The debugger should show at the beginning of each line a prompt character to indicate when it is waiting for a command. I recommend the prompt be the ">" character. The debugger should also show a cursor which I recommend to be a black square.

The debugger should support the standard debugging commands such as displaying/setting memory, displaying/setting registers, and disassembling 6502 instructions. This disassembly should support the special SOS BRK call by listing the word "BRK/SOS" instead of just "BRK" and following this with the SOS command number/name and the parameter list address:

```
SOS C0/CREATE 345A
```

The user should be able to enable or disable this feature.

Note: It may be good to also support the Apple ][ ProDOS command calling scheme in case this emulator ever becomes an Apple ][ emulator.

The debugger should support break points, single stepping, and timing buckets. The



timing buckets would be used in conjunction with break points to record how long a sequence of 6502 instructions took to execute. This can be very useful in locating emulator bottlenecks. The debugger supports many break point commands since I have a feeling that this facility will be very powerful and useful during the emulator's development.

The debugger should support the collection of statistics about the emulator. I recommend tracking how many times specific 6502 opcodes are executed (obviously, the debugger would need commands to display and clear this information). I would also track memory accesses on at least a page (256 bytes) basis.

The debugger should be accessible at any time that the emulator is running. I recommend some type of key press combination that the emulator would detect and display the debugger window. Once the debugger window is active it should remain on the screen until the user closes the window.

The emulator should also support a special key press combination at emulator startup time that activates the debugger just before the /// ROM is run. This can give the emulator developer a good way of tracing ROM execution.

The emulator should activate the debugger if any fatal emulation errors are detected and the debugger should show a message detailing the reason for the activation. All of these errors display a dump of the 6502 and SOS control registers. Reasons for debugger activation from the emulator are:

1. A program writes to write-protected memory (e.g. SOS's address space). The displayed message is "EMULATOR EXCEPTION: WRITING TO WRITE-PROTECTED MEMORY".
2. A program executes an undefined 6502 instruction (e.g. 6502 opcode \$02). The displayed message is "EMULATOR EXCEPTION: UNDEFINED 6502 OPCODE".

When the debugger is initialized (which should be when the emulator starts) the debugger should check if a text file named "DDT.TXT" exists. If so, the debugger should read each line from this file and execute it. Obviously, this file should contain debugger instructions. This can be very useful for setting up commonly used break points which if you use many would be tedious to type everytime you wanted to use the emulator.

A memory snapshot facility should also exist. When activated by a debugger command this facility would write to the host computer's disk a binary file containing a copy of all the /// memory areas. This snapshot should also be readable by the debugger so that the user could restart a specific emulation session from the snapshot.

I recommend the following emulator debugger commands which are based on the /// Monitor commands so that these debugger commands will be familiar to Monitor users. These commands for the most part have the general syntax of address-command. See my document "Inside the Apple /// Computer ROM" for a list of the /// Monitor commands. For information about the Apple ][ Monitor commands, which the /// Monitor commands are based upon, see "Apple ][ Reference Manual" (Chapter 3: The System Monitor, dated 1981).

Addresses appearing in debugger commands may be prefaced by "N/" where N is a bank number. For example, to reference address 2000 of bank 4 use 4/2000. If no bank number precedes an address the current bank is used. To reference a ROM address use a bank "number" of "R", for example "R/F000". To reference a character set address use a bank "number" of "C", for example "C/0000". To reference the SOS system bank use "S", e.g. "S/1400".

Commands should be case-insensitive (none of the UNIX case-sensitivity gobbly-gook).

Commands that display more than a screen full of information should either automatically pause when the screen is full, or the user can use the SPACE key.

Note: Commands using ":" may also use ";" which is easier to type since this



character does not need the user of the shift key. Same for "<" and "/".

Most debugger command numeric arguments must be specified in hexadecimal. The exception is the X command which supports hexadecimal, decimal, and binary.

The debugger command parser should be very liberal. This means that users should be able to include extra spaces (or no spaces) and the command should be parsable. For example, if a command needs a list of bytes the user should be able to enter any of the following: "AABBCC", "AA BB CC", " A ABBC C " and the debugger will see these as "AABBCC".

The debugger should also support a command macro facility. This facility allows you to define a macro consisting of other debugger commands. Typing the name of the macro will then type the commands as if you entered them manually.

# **HELP (or ?) cmd name**

Display debugger on-line help for all commands. Help info should be stored in an external text file for easier modification. I recommend that this section of this document be the help file.

Example: HELP

BYE

Return to the emulator.

Example: BYE

# **CARRIAGE RETURN keypress**

Repeat last command.

Example: If the last command was HELP and you press the CARRIAGE RETURN key then HELP will be displayed and executed again.

# **SPACE keypress**

Pause current command's output. Press again to continue.

Example: If a command is executing and you press the SPACE key the command's output will be paused, pressing SPACE again resumes the command's output. Pausing/Resuming are done on an output line basis only.

# **DELETE keypress**

Stop current command's output.

Example: If a command is executing and you press this key then the command will stop executing and you will be returned to the debugger's prompt.

# **RD**

Display 6502 registers and /// system control registers.

Example: RD

A=04 X=01 Y=D8 P=30/00000011 S=F8 PC=034A : E=77/01110111 Z=1A B=03

Some Ideas about an Apple /// Computer Emulator -- Version 4

David T Craig -- 12 Dec 1997 -- 10/23  
S - System Clock rate R - Reset enable R - ROM  
I - I/O space V - Video W - Write protect R - ROM  
C - Screen (Color) S - Stack in use

Show table explaining bits in P and E

N negative  
V overflow  
B break command  
D decimal mode  
I interrupt disable  
Z zero  
C carry



-----  
**byte:SA**

Set 6502 A register to byte.

Example: 45:SA     - -     \_\_\_\_\_

-----  
**byte:SX**

Set 6502 X register to byte.

Example: 7B:SX     - -     \_\_\_\_\_

-----  
**byte:SY**

Set 6502 Y register to byte.

Example: FF:SY     \_\_\_\_\_

-----  
**byte:SP**

Set 6502 P register to byte.

Example: 56:SP     \_\_\_\_\_

-----  
**byte:SS**

Set 6502 S register to byte.

Example: AA:SS     \_\_\_\_\_

-----  
**word:SPC**

Set 6502 PC register to word.

Example: 2000:SPC     \_\_\_\_\_

-----  
**byte:SE**

Set /// E system control register to byte.

Example: 34:SE     \_\_\_\_\_

-----  
**byte:SZ**

Set /// Z system control register to byte.

Example: 19:SZ     \_\_\_\_\_

-----  
**byte:SB**

Set /// B system control register to byte.

Example: 06:SB     \_\_\_\_\_



### ----- addr1.addr2

Dump memory data to screen from address 1 to address 2 and display ASCII character at the right of the screen.

Example (assumes current bank is bank 4): 300.30F

4/0300- B900 080A 0A0A 9900 08C8 D0F4 A62B A909 [F..d.uy%^&90@..G]

### ----- ZPAGE

Dump the contents of the current interpreter's Zero Page (256 bytes). Also supported are commands for the Stack Page and the Extend Page:

SPAGE - stack page  
EPAGE - extend page

To dump the pages for SOS (and drivers) use the following commands:

SZPAGE - zero page  
SSPAGE - stack page  
SEPAGE - extend page

Example: ZPAGE

Zero Page (interpreter)

00 1 2 3 4 5 6 7 ...  
1400- 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF  
1420- 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF  
...  
14E0- 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF

### ----- addr:bytes

Store starting at the address the bytes.

Example: 2000:AA BB CC DD EE FF  
2000:AABBCCDDEEFF

### ----- addr:'text'

Store text starting at address (high bit clear).

Example: 2000:'Hello World'  
2000:'David's Dog' -- (this stores) David's Dog

### ----- addr:"text"

Store text starting at address (high bit set).

Example: 2000:"Hello World"  
2000:"David's Dog"  
2000:"I said 'Hi!'" -- (this stores) David's Dog  
I said "Hi!"

### ----- addr3<addr1.addr2M

Move data in address range to address 3.

Example: 2000<3000.3100M

Some Ideas about an Apple /// Computer Emulator -- Version 4  
David T Craig -- 12 Dec 1997 -- 12 / 23



-----  
**addr3<addr1.addr2V**

Verify data in address range equals data starting at address 3.

Example: 2000<3000.3100V     ~~~~~

Displays either "OK" if the verification succeeds, or "MISMATCH" if the verification fails.

-----  
**bytes<addr1.addr2S**

Search memory in address range for the bytes.

Example: AA<3000.3100S     -- searches for byte AA  
 AABCC<3000.3100S     -- searches for bytes AA BB CC

If a search finds a match then the starting address of the match is displayed, otherwise "PATTERN NOT FOUND" is displayed.

*PATTERN FOUND AT addr*

-----  
**'text'<addr1.addr2S**

Search memory in address range for text (high bit clear).

Example: 'D'<3000.3100S     ~~~~  
 'David'<3000.3100S     ~~~~

-----  
**"text"<addr1.addr2S**

Search memory in address range for text (high bit set).

Example: "D"<3000.3100S     ~~~~  
 "David"<3000.3100S     ~~~~

-----  
**disk.block<addr1.addr2W**

Write address range to disk # disk starting at disk block. If disk # is not present then uses disk .D1. Disk should equal 1, 2, 3, or 4. The address range always ends on a block boundary no matter what you type.

Example: 1.117<2000.21FFW     -- write 512 bytes to disk 1 block \$117

Note: Disk /// disks contain 280 blocks (\$118) so the block range is 0-117 (hexadecimal).

-----  
**disk.block<addr1.addr2R**

Read from disk # disk starting at block to the address range. If disk # is not present then uses disk .D1. See the W command for more info.

Example: 1.117<2000.21FFR     ~~~~ read 512 bytes from disk 1 block \$117

-----  
**disk.block-block:DISK**

Read block range from disk # disk to a special debugger 4K buffer which is not used by the emulator. If the typed block range is greater than 4K then only the first 4K will be read. You can then examine this buffer's contents either with a hex/ascii



dump or with a disassembly (command L). This command is useful when you want to examine a disk's contents. For disassembly purposes, you can specify the logical starting address for the buffer. See the DISKBUFFER command.

To disassemble the special disk buffer (see the L command) use bank X (stands for "extra") as part of the disassembly address parameter (e.g. "X/100"). Same for dumping memory or whatever commands you want to use with this special buffer.

Example: 1.0-7:DISK -- read 8 blocks (0 to 7) from disk 1

#### ----- addr:DISKBUFFER

Set disk buffer starting logical address. Default address is 2000. See the DISK command.

Example: A000:DISKBUFFER -- ~~~~~

Range is 0000-FFFF

#### ----- addr1.addr2L

Disassemble instructions in address range. If only addr1 appears then disassemble 20 instructions. Disassembly includes the opcode cycle count.

Example: 300L -- assumes bank 4 is current

4/0300-	A9 C1	'X.'	(2)	LDA #\$C1	;
4/0302-	20 ED FD	'...'	(5)	JSR \$FDED	;
4/0305-	18	'.'	(2)	CLC	;
4/0306-	69 0A	'T.'	(4)	ADC #\$01	;
4/0308-	C9 DB	'..'	(3)	CMP #\$DB	;
4/030A-	D0 F6	'..'	(3)	BNE \$0302	;
4/030C-	60	'U'	(4)	RTS	;
1	2	3	4	5	6 (see Note)

Note: Column 1 = bank register/address  
 Column 2 = memory bytes  
 Column 3 = ASCII for the memory bytes  
 Column 4 = opcode cycle count  
 Column 5 = disassembled instructions  
 Column 6 = remark character ";" (optional, see DISASMREM)

L by itself disassembles the next 20 instructions.

#### ----- DISASMREM

Display ";" after each disassembly line that is produced by the L command. Default is to not display the remark. Useful if you plan to add comments to a disassembly. See also DISASMREMOFF.

Example: DISASMREM

#### ----- DISASMREMOFF

Turn off DISASMREM. See also DISASMREM.

Example: DISASMREMOFF

#### ----- addrG





Call subroutine at the address.

Example: A000G     -- ~~~~~

#### addrJ

Jump to the address.

Example: A000J     -- ~~~~~

#### wordX

Convert word (or up to 4 hex digits) to hexadecimal, decimal, and binary (X stands for "translate"). Prefix character for byte determines its base: no prefix = hex, . = dec, t = binary.

Example: AX     ->     A(16)     10(10)     0000 0000 0000 1010(2)  
           .10X     ->     A(16)     10(10)     0000 0000 0000 1010(2)  
           t1010     ->     A(16)     10(10)     0000 0000 0000 1010(2)  
           FFFFX     ->     FFFF(16)     65535(10)     1111 1111 1111 1111(2)

*Put in table for easier viewing*

#### addr1.addr2:CS

Calculate and display a checksum for address range. Checksum is a 4 byte quantity which is calculated the same as the disk image file checksums.

Example: 300.500:CS     -- ~~~~~  
           CHECKSUM=AF897CEE

#### addrT

Trace instructions starting at the address. Each traced instruction displays register contents. Press the SPACE to pause the trace, press DELETE to stop the trace. The displayed registers contain values after the previously listed command executes.

Example: A000T     -- assuming bank 4 is current

4/A000-     A9 C1     'X.'     (2)     LDA #\$C1  
 A=C1     X=01     Y=D8     P=30/00000011     S=F8     PC=A002 : E=77/01110111     Z=1A     B=04  
 4/A002-     20 ED FD     '...'     (5)     JSR \$FDED  
 A=C1     X=01     Y=D8     P=30/00000011     S=F6     PC=FD00 : E=77/01110111     Z=1A     B=04  
       *lisp*                        *bit names*                        *bit names*

Note: Press the DELETE key to stop the trace, SPACE to pause/resume.

#### addrSS

Single step trace starting at the address. After each step pause and wait for user to press SPACE to continue or DELETE to stop the single step.

Example: A000T <sup>SS</sup> -- assuming bank 4 is current

4/A000-     A9 C1     'X.'     (2)     LDA #\$C1  
 A=C1     X=01     Y=D8     P=30/00000011     S=F8     PC=A002 : E=77/01110111     Z=1A     B=04  
       *lisp*                        *names*                        *names*

Note: Press SS by itself to single step the next instruction, or press CARRIAGE RETURN to repeat the SS.

**addr:BP**

Set a break point at address. When address is accessed the debugger is entered and displays the registers. Up to 100 break points should be supported.

Example: A000:BP

**addr:BPC**

Clear break point at address.

Example: A000:BPC

**SOS:BP**

Set a break point when a SOS call is made. This means when the BRK opcode is executed. Same as M00:BP.

Example: SOS:BP

**Mopcode:BP**

Set a break point when opcode is executed.

Example: M60:BP -- set break point when the RTS instruction (60) is executed.

**ROM:BP**

Set a break point when a call is made to the ROM.

Example: ROM:BP

**addr1.addr2:BPW**

Set a break point when any address within address range is written to. BPW = Break Point Write.

Example: 300.123AR:BPW

**addr1.addr2:BPR**

Set a break point when any address within address range is read from. BPR = Break Point Read.

Example: 300.123A:BPR

**addr.byte:BPE**

Set a break point when the address contents equal the byte value. BPE = Break Point Equals.

Example: 300.AA:BPE

**addr.byte1-byte2:BPE**

Set a break point when the address contents equal a byte value in the byte range. BPE



*new cmd  
BPNE BP not equals  
same syntax as BPE*

= Break Point Equals.

Example: 300.AA-BB:BPE

-----  
**addr.byte1 byte2 ... :BPEA** *e*

Set a break point when the address contents equal byte 1 value, or equals byte 2 value, etc. Supports up to 16 byte values. BPEA = Break Point Equals Any.

Example: 300.AABBCCDD:BPEA  
300.AA BB CC DD:BPEA

-----  
**addr1.addr2.byte1 byte2 ... :BPEA** *e*

Set a break point when the address range contains any bytes equalling the byte values. BPEA = Break Point Equals Any.

Example: 300.400.AABBCCDD:BPEA

-----  
**addr1.addr2.byte1-byte2:BPEA** *e*

Set a break point when the address range contents equal the byte range. BPEA = Break Point Equals Any.

Example: 300.400.AA-BB:BPEA

-----  
**BPD**

Display break point table.

Example: BPD

#	Address Range	BP	Setting
1	4/2000-4/21FF	BPEA	AA-BB

-----  
**BPC**

Clear break point table.

Example: BPC

-----  
**addr1.addr2:TB**

Set timing bucket for address range. When address 1 is accessed timing starts. When address 2 is accessed timing stops. Up to 100 timing buckets should be supported.

Example: A000.A1FF:TB

-----  
**TBD**

Display timing bucket table. Shows all set timing buckets and the time in 1/60th of a second and in seconds spent in each bucket.

Example: TBD

#	Address Range	Time (1/60s)	Time (secs)
---	---------------	--------------	-------------



1	4/A000-4/A1FF	34	0.567
2	4/A300-4/A310	5	0.083
		39	0.650

#### addr:TBC

Clear timing bucket starting at address.

Example: A000:TBC

#### TBC

Clear timing bucket table.

Example: TBC

#### error:SOSE

List SOS general error message for the error number. If no error number is present then list all general errors. Error info should be stored in an external text file for easier modification. See the SOS Reference Manual for a list of these errors.

Example: 01:SOSE

BADSCNUM - Invalid SOS call number

#### error:SOSFE

Display SOS fatal error message for the error number. If no error number is present then list all fatal errors. See the SOS Reference Manual for a list of these errors.

Example: 01:SOSFE

BADBRK - Invalid BRK

#### command:SOS

Display SOS command name and SOS command area (e.g. file system) for the command number. If no command number present then list all SOS command numbers and their names. Command info should be stored in an external text file for easier modification. See the SOS Reference Manual for a list of these commands.

Example: C0:SOS

CREATE (File System)

#### SOSON

Turn on disassembly of SOS calls which displays SOS followed by the command number and parameter address. The emulator defaults to this.

Example: SOSON

#### SOSOFF



Turns off SOSON.

Example: SOSOFF

-----  
**disk:CAT**

Display catalog of SOS disk stored in disk # disk. Includes recursive list of all subdirectories. Should show same file info as Apple's System Utilities program.

Note: Other commands that may be supported include CATPASCAL for Apple ][ Pascal disks and CATDOS for Apple ][ DOS disks. This may come in handy if you want to see what these disks contain if you have them as disk image files.

Example: 1:CAT

-----  
**disk.file\_name:INFO**

Displays information about the specified file in the disk. Information includes standard SOS file information but also block list of all index blocks (if any) associated with the file and block list of all data blocks for the file.

Example: 1.APPLE3.TEXT:INFO

-----  
**disk.block:DUMP**

Display contents of specified disk block in the standard hex/ascii dump format.

Example: 1.0:DUMP

-----  
**disk:DRIVERS**

Display list of contents of the SOS.DRIVER file stored on the disk. List includes driver names, driver information, and other items that are in the driver file (e.g. character sets).

Example: 1:DRIVERS

-----  
**disk:CHECKIMAGE**

Check validity of disk image in disk # disk. Computes header and data part checksums and compares against the image file's listed checksums.

Example: 1:CHECKIMAGE

-----  
**DIT**

Display Driver Information Table (DIT), a data structure maintained by this debugger. Contains list of all loaded drivers, their names, sizes, and entry point addresses.

Example: DIT

-----  
**MIT**

Display Memory Information Table (MIT), a data structure maintained by this debugger. See section EMULATOR MEMORY STRUCTURE for what this structure contains.

Example: MIT



# OPCODES

Display a histogram of opcode execution counts. Includes the actual number of the counts. Sorted by frequency. Opcodes not executed are listed below the histogram.

Example: OPCODES

```
LDA  2,188,973 *****
STA  12,123  *****
CMP  467     *****
-----
      2,201,563
```

Unexecuted opcodes: TXS NOP

# OPCODESCLR

Reset opcode histogram table.

Example: OPCODESCLR

# page1.page2:MEMORYR

Display memory write access table. This table lists on a 256 byte page basis counts for each time the page was read. If page1.page2 specified then lists only those pages. If a single page is specified then display only that page's access count.

Example: 0.5:MEMORYR

# page1.page2:MEMORYW

Display memory read access table. This table lists on a 256 byte page basis counts for each time the page was written. See MEMORYW for page options.

Example: 0.5:MEMORYW

# MEMORYCLR

Reset both memory access tables.

Example: MEMORYCLR

# value:SCROLL

Set debugger display scrolling rate interline delay. Value is in 1/10th of a second. Default is no delay (value = 0). Useful if you want to for example dump lots of memory and don't want to mess with the SPACE key to read what is displayed. Set the scrolling delay to a comfortable value, sit back, and enjoy the show.

Example: 10:SCROLL -- sets scrolling delay to 1 second

# filename:LOG

Close log file, create a new one with filename, and output all debugger displays to this new file. Useful if you're running the emulator from a write-protected disk and you want to re-direct the output to a writable disk file.

Some Ideas about an Apple /// Computer Emulator -- Version 4

David T Craig -- 12 Dec 1997 -- 20/23



Example: MyDiary:LOG

#### ----- SNAPSHOTW

Write the contents of all of the emulator's memory to binary file on the host computer's hard disk. This snapshot could prove useful in diagnosing an emulator problem. The binary file should be named "Snapshot\_YYYYMMDD\_HHMMSS.BIN".

Example: SNAPSHOTW

#### ----- SNAPSHOTRfile-name

Read a snapshot file into the emulator's memory.

Example: SNAPSHOTR Snapshot\_19971225\_123456.BIN

#### ----- MACRO name commands

Define a macro name and commands for this macro. You can use any name containing alphanumeric characters or periods with a maximum length of 31 characters. Up to 25 macros may be defined. All commands are verified and if any syntax errors occur you will be told and the macro will not be defined. Macro commands cannot include other macro commands.

Example: MACRO my.dump 300.400 A000.A1FF A000L

#### ----- MACROL

List all defined macros.

Example: MACROL

```
# Name / Contents
-----
1 my.dump
  300.400 A000.A1FF A000L
```

#### ----- !macro-name

Execute a macro with the name "macro-name". Each command within the macro is displayed followed by the commands' display.

Example: !my.dump

```
300.400
...
A000.A1FF
...
A000L
...
```

*font display current font bitmap char font?ROM ROM font bitmap*

#### ----- VERSION

Display debugger version information. Includes version number and creation date/time.

=====



## 17.0 EMULATOR MEMORY STRUCTURE

I recommend that the emulator's internal memory structure for the Apple /// memory resources be structured as follows:

o Memory block containing the size of memory and references to each /// memory bank (the references can be whatever is appropriate -- on the Mac these could be Mac memory pointers or handles):

- number of switchable banks (1..15)
- reference to bank S (32K: 0000-1FFF, A000-FFFF) \*
- reference to bank 0/\$0 - switchable (32k: 2000-9FFF)
- reference to bank 1/\$1 - switchable (32k: 2000-9FFF)
- ...
- reference to bank 14/\$E - switchable (32k: 2000-9FFF)
- reference to Boot ROM ROM address space (4k: F000-FFFF)
- reference to Boot ROM RAM address space (4k: F000-FFFF)
- reference to I/O RAM address space (4k: C000-CFFF)

\* The system (S) bank is always on-line and is never bank switched. SOS and part of the interpreter reside here.

o Memory block containing the 6502 registers:

- Accumulator (A) 8 bits
- X index (X) 8 bits
- Y index (Y) 8 bits
- Status Register (P) 8 bits
- Stack Pointer (S) 8 bits
- Program Counter (PC) 16 bits

o Memory block containing the special /// System Control Registers:

- E: Environment Register (FFDF) 8 bits
- Z: Zero Page Register (FFD0) 8 bits
- B: Bank Register (FFEF) 8 bits

## 18.0 WHAT'S NEXT?

Persons seriously interested in creating an Apple /// emulator program should try to obtain as much /// technical information as possible. The author has lots of info which he can copy at minimal charge (10 cents per page plus postage). These persons should also have access to a working Apple /// computer with a fair number of /// programs.

Other areas of compatibility should also be investigated that this document does not address. This includes support for other input devices such as the mouse which does have a 3rd party driver available.

## 19.0 REFERENCES

- Apple /// Owner's Guide, Apple Computer, 1981
- Apple /// Plus Owner's Guide, Apple Computer, 1982
- Apple /// System Data Sheet, Apple Computer, July 1983
- Apple /// Plus System Data Sheet, Apple Computer, October 1983
- Apple /// Standard Device Drivers Manual, Apple Computer, 1981

Some Ideas about an Apple /// Computer Emulator -- Version 4  
David T Craig -- 12 Dec 1997 -- 22/23

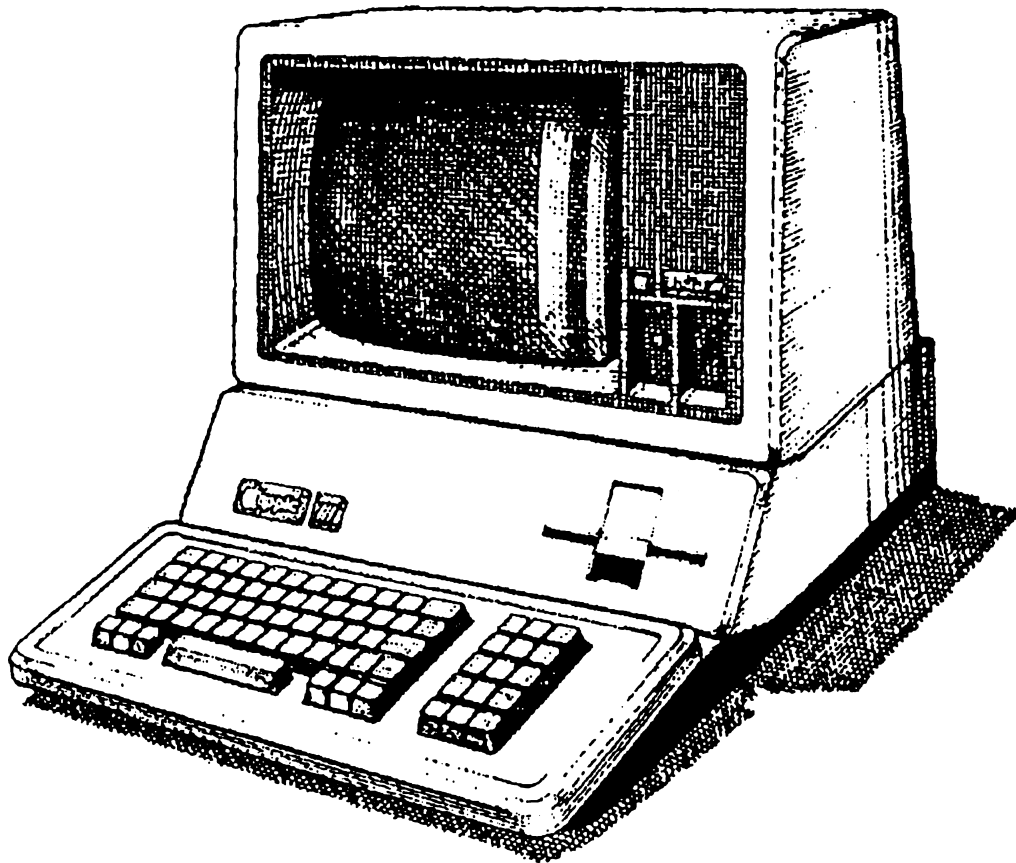


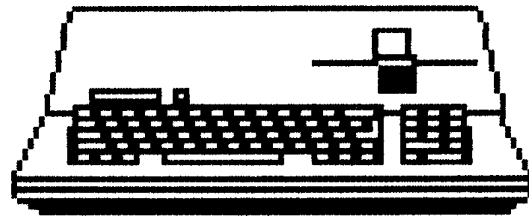


Apple /// SOS Reference Manual, Apple Computer, 1982  
Apple /// SOS Device Driver Writer's Guide, Apple Computer, 1982  
Apple /// Service Reference Manual (Level 2), Apple Computer, 1983  
/// Bits: John Jeppson's Guided Tour of Highway ///, Softalk magazine, May 1983  
Bank Switch Razzle-Dazzle, Softalk magazine, August 1982  
The Apple Nobody Knows, Apple Orchard magazine, Fall 1981  
Apple /// Entry Points, Andy Wells, Call-APPLE, October 1981  
Inside the Apple /// Computer ROM, David Craig, November 1997  
###



# Apple III Computer Information





Apple ///  
Apple ///+

Apple /// SOS Technical Information

# SOS 1.3

## Floppy Bootstrap Loader Source Code Listing

**This listing shows the code which is found at the beginning of a SOS boot disk. When the Apple /// computer starts the computer's ROM loads this code from the floppy disk and executes the code. This code loads the Apple ///'s operating system, SOS.**

Source Code Listing  
for

**Apple ///**

# **SOS Floppy Bootstrap Loader**

David T. Craig  
736 Edgewater  
Wichita, Kansas 67230

10/31/89 9:45

HD:Apple ///:SOS Floppy Bootstrap Loader

Page 1

DAVID L. CRAIG

```

0000| ;*****
0000| ; APPLE /// BOOTSTRAP LOADER FOR FLOPPY DISK
0000| ; - Disassembled 10-March-1988 by Scott Stinson
0000| ;*****
0000|
0000| .ABSOLUTE
0000| .PROC BOOTSTRAPLOADER
0000| .ORG 0A000
0000|
0000| ;-----
0000| ; EQUATES
0000| ;-----
0000|
0000| ;-----
0000| ; ZERO PAGE LOCATIONS
0000| ;-----
0000|
0000| IBDRVN .EQU 82 ; DRIVE NUMBER
0000| IBTRK .EQU 83 ; TRACK NUMBER
0000| IBSECT .EQU 84 ; SECTOR NUMBER
0000| IBBUFP .EQU 85 ; BUFFER POINTER
0000| IBCMD .EQU 87 ; COMMAND NUMBER
0000| IBBUPTMP .EQU 0E3 ; BUFFER POINTER TEMPORARY
0000| FILECNT .EQU 0E5 ; FILE COUNT
0000| INDXBLCNT .EQU 0E7 ; INDEX BLOCK COUNT
0000| SOSJMPADR .EQU 0E8 ; SOS JUMP ADDRESS
0000|
0000| ;-----
0000| ; HARDWARE I/O ADDRESSES
0000| ;-----
0000|
0000| SCREENLOC .EQU 0628 ; SCREEN LOCATION
0000| KBDSTROBE .EQU 0C010 ; KEYBOARD STROBE
0000| IOBEEP .EQU 0C040 ; I/O BEEP
0000|
0000| ;-----
0000| ; GENERAL EQUATES
0000| ;-----
0000|
0000| RETINT .EQU 40 ; RETURN FROM INTERRUPT
0000| IDXBLK1 .EQU 0C00 ; INDEX BLOCK 1
0000| IDXBLK2 .EQU 0D00 ; INDEX BLOCK 2
0000| LOADADR .EQU 1E00 ; LOADING ADDRESS
0000| OFFSET .EQU 1E08 ; OFFSET
0000| FIRSTPAGE .EQU 2000 ; FIRST PAGE
0000| MAINBUFP .EQU 0A200 ; MAIN BUFFER
0000| REGRWTS .EQU 0F000 ; READ/WRITE SECTOR ROUTINE
0000| SECTABL .EQU 0F4A0 ; SECTOR TABLE
0000| NMIVECTOR .EQU 0FFCA ; NON-MASKABLE INTERRUPT VECTOR
0000| EREG .EQU 0FFDF ; ENVIRONMENT REGISTER
0000| BREG .EQU 0FFEF ; BANK REGISTER
0000|
0000| ;-----
0000| ; ENTRY POINT
0000| ;-----
0000|
0000| ENTRY SEI ; SET INTERRUPT DISABLE
0001| CLD ; CLEAR DECIMAL FLAG
0002| LDA #77 ; LOAD ACCUMULATOR WITH $77
0004| STA EREG ; STORE IN ENVIRONMENT REGISTER
0007| ; SET 2 MHZ, I/O SPACE ENABLED, SCREEN ENABLED,
0007| ; RESET ENABLED, WRITE PROTECT NOT ENABLED,
0007| ; PRIMARY STACK, AND ROM SELECTED
0007| LDA #0FF ; LOAD ACCUMULATOR WITH $FF
0009| TXS ; TRANSFER X-REGISTER TO STACK POINTER
000A| BIT KBDSTROBE ; CLEAR KEYBOARD
000D| LDA #RETINT ; LOAD ACCUMULATOR WITH RETURN FROM INTERRUPT
000F| STA NMIVECTOR ; STORE IN NON-MASKABLE INTERRUPT VECTOR
0012| LDA #07 ; LOAD ACCUMULATOR WITH $07
0014| STA BREG ; STORE IN BANK REGISTER
0017| LDA #00 ; LOAD ACCUMULATOR WITH $00
0019| DEC BREG ; DECREMENT BANK REGISTER
001C| STA FIRSTPAGE ; STORE IN FIRST PAGE OF BANK
001F| LDA FIRSTPAGE ; LOAD X-REGISTER WITH FIRST PAGE BYTE
0022| BNE $010 ; BRANCH IF BYTE IS NOT EQUAL TO $00
0024|
0024| ;-----
0024| ; This section reads in the SOS directory.
0024| ;-----
0024|
0024| READSOSDIR LDA #00 ; LOAD ACCUMULATOR WITH $00-BLOCK HIGH BYTE
0026| STA IBBUFP ; STORE IN BUFFER POINTER LOW BYTE
0028| LDY #0A2 ; LOAD X-REGISTER WITH $A2
002A| STX IBBUFP+1 ; STORE IN BUFFER POINTER HIGH BYTE
002C| LDY #02 ; LOAD X-REGISTER WITH $02-BLOCK LOW BYTE
002E| LDY IBBUFP ; LOAD Y-REGISTER WITH BUFFER POINTER LOW BYTE
0030| STY IBBUPTMP ; STORE IN BUFFER POINTER TEMPORARY LOW BYTE
0032| LDY IBBUFP+1 ; LOAD Y-REGISTER WITH BUFFER POINTER HIGH BYTE
0034| STY IBBUPTMP+1 ; STORE IN BUFFER POINTER TEMPORARY HIGH BYTE
0036| JSR READBLK ; JUMP TO READ A BLOCK FROM FLOPPY DISK DRIVE

```

ROM [

10/31/89 9:45

HD:Apple ///:SOS Floppy Bootstrap Loader

Page 2

```

A039| A0 02          LDY    #02          ; LOAD Y-REGISTER WITH $02
A03B| B1 E3          LDA    @IBBUPTMP,Y  ; LOAD ACCUMULATOR WITH NEXT BLOCK TO READ LOW
A03D|                ; BYTE
A03D| AA            TAX          ; TRANSFER ACCUMULATOR TO X-REGISTER
A03E| C8            INY          ; INCREMENT Y-REGISTER
A03F| B1 E3          LDA    @IBBUPTMP,Y  ; LOAD ACCUMULATOR WITH NEXT BLOCK TO READ HIGH
A041|                ; BYTE
A041| D0EB          BNE    RDSOSDIRLP    ; BRANCH IF NEXT BLOCK TO READ HIGH BYTE IS NOT
A043|                ; EQUAL TO ZERO
A043| E0 00          CPX    #00          ; CHECK TO SEE IF NEXT BLOCK TO READ LOW BYTE IS
A045|                ; ZERO
A045| D0E7          BNE    RDSOSDIRLP    ; BRANCH IF NEXT BLOCK TO READ LOW BYTE IS NOT
A047|                ; EQUAL TO ZERO
A047|
A047| ; -----
A047| ; This section searches the SOS directory for the SOS.KERNEL file.
A047| ; -----
A047|
A047| AD 25A2          SRCHSOSKER LDA    MAINBUFF+25 ; LOAD ACCUMULATOR WITH FILE COUNT LOW BYTE
A04A| 85 E5          STA    FILECNT      ; STORE IN FILE COUNT LOW BYTE
A04C| AD 26A2          LDA    MAINBUFF+26 ; LOAD ACCUMULATOR WITH FILE COUNT HIGH BYTE
A04F| 85 E6          STA    FILECNT+1    ; STORE IN FILE COUNT HIGH BYTE
A051| 05 E5          ORA    FILECNT      ; OR ACCUMULATOR WITH FILE COUNT LOW BYTE
A053| D003          BNE    $010         ; BRANCH IF FILE COUNT IS NOT EQUAL TO ZERO
A055| 4C 56A1          JMP    WRNTFNDERR ; JUMP TO WRITE NOT FOUND ERROR MESSAGE TO
A058|                ; SCREEN
A058| A5 E5          $010 LDA    FILECNT      ; LOAD ACCUMULATOR WITH FILE COUNT LOW BYTE
A05A| D002          BNE    $020         ; BRANCH IF NOT EQUAL TO $00
A05C| C6 E6          DEC    FILECNT+1    ; DECREMENT FILE COUNT HIGH BYTE
A05E| C6 E5          $020 DEC    FILECNT      ; DECREMENT FILE COUNT LOW BYTE
A060| A9 2B          LDA    #2B          ; LOAD ACCUMULATOR WITH $28
A062| 85 85          STA    IBBUFP      ; STORE IN BUFFER POINTER LOW BYTE
A064| A9 A2          LDA    #0A2         ; LOAD ACCUMULATOR WITH $A2
A066| 85 86          STA    IBBUFP+1    ; STORE IN BUFFER POINTER HIGH BYTE
A068| AE 24A2          LDX    MAINBUFF+24 ; LOAD X-REGISTER WITH ENTRIES PER BLOCK
A06B| CA            DEX          ; DECREMENT X-REGISTER
A06C| A0 00          SRCHLP  LDY    #00          ; LOAD Y-REGISTER WITH $00
A06E| B1 85          LDA    @IBBUFP,Y    ; LOAD ACCUMULATOR WITH STORAGE TYPE AND NAME
A070|                ; LENGTH BYTE
A070| F01A          BEQ    $020         ; BRANCH IF EQUAL TO ZERO
A072| 29 0F          AND    #0F          ; MASK OFF BITS 4,5,6,7
A074| CD 92A1          CMP    FLNMLEN    ; COMPARE WITH FILE NAME LENGTH
A077| D013          BNE    $020         ; BRANCH IF NOT EQUAL TO ZERO
A079| A8            TAY          ; TRANSFER NAME LENGTH TO Y-REGISTER
A07A| B1 85          $010 LDA    @IBBUFP,Y  ; LOAD ACCUMULATOR WITH FILE NAME BYTE
A07C| D9 92A1          CMP    FLNME-1,Y    ; COMPARE WITH FILE NAME BYTE
A07F| D00B          BNE    $020         ; BRANCH IF NOT EQUAL
A081| 88            DEY          ; DECREMENT NAME LENGTH
A082| D0F6          BNE    $010         ; BRANCH IF NAME LENGTH NOT EQUAL TO ZERO
A084| B1 85          LDA    @IBBUFP,Y    ; LOAD ACCUMULATOR WITH STORAGE TYPE AND NAME
A086|                ; LENGTH BYTE
A086| 29 F0          AND    #0F0         ; MASK OFF BITS 0,1,2,3
A088| C9 20          CMP    #20          ; COMPARE WITH $20 FOR SAPLING FILE
A08A| F032          BEQ    READIDXBLK    ; BRANCH IF EQUAL TO READ INDEX BLOCK
A08C| 08          $020 PHP          ; PUSH PROCESSOR STATUS ON STACK
A08D| CA            DEX          ; DECREMENT ENTRIES PER BLOCK
A08E| F010          BEQ    $030         ; BRANCH IF ENTRIES PER BLOCK IS EQUAL TO ZERO
A090| 18            CLC          ; CLEAR CARRY
A091| A5 85          LDA    IBBUFP      ; LOAD ACCUMULATOR WITH BUFFER POINTER LOW BYTE
A093| 6D 23A2          ADC    MAINBUFF+23 ; ADD ENTRY LENGTH LOW BYTE
A096| 85 85          STA    IBBUFP      ; STORE IN BUFFER POINTER LOW BYTE
A098| A5 86          LDA    IBBUFP+1    ; LOAD ACCUMULATOR WITH BUFFER POINTER HIGH BYTE
A09A| 69 00          ADC    #00          ; ADD $00
A09C| 85 86          STA    IBBUFP+1    ; STORE IN BUFFER POINTER HIGH BYTE
A09E| D009          BNE    $040         ; BRANCH ALWAYS
A0A0| A9 04          $030 LDA    #04          ; LOAD ACCUMULATOR WITH $04
A0A2| 85 85          STA    IBBUFP      ; STORE IN BUFFER POINTER LOW BYTE
A0A4| E6 86          INC    IBBUFP+1    ; INCREMENT BUFFER POINTER HIGH BYTE
A0A6| AE 24A2          LDX    MAINBUFF+24 ; LOAD X-REGISTER WITH ENTRIES PER BLOCK
A0A9| 28          $040 PLP          ; PULL PROCESSOR STATUS FROM STACK
A0AA| F0C0          BEQ    SRCHLP      ; BRANCH IF NOT EQUAL TO ZERO
A0AC| 38            SEC          ; SET CARRY
A0AD| A5 E5          LDA    FILECNT      ; LOAD ACCUMULATOR WITH FILE COUNT LOW BYTE
A0AF| E9 01          SBC    #01          ; SUBTRACT $01
A0B1| 85 E5          STA    FILECNT      ; STORE IN FILE COUNT LOW BYTE
A0B3| A5 E6          LDA    FILECNT+1    ; LOAD ACCUMULATOR WITH FILE COUNT HIGH BYTE
A0B5| E9 00          SBC    #00          ; SUBTRACT $00
A0B7| 85 E6          STA    FILECNT+1    ; STORE IN FILE COUNT HIGH BYTE
A0B9| B0B1          BCS    SRCHLP      ; BRANCH IF MORE FILE ENTRIES
A0BB| 4C 56A1          JMP    WRNTFNDERR ; JUMP TO WRITE NOT FOUND ERROR MESSAGE TO
A0BE|                ; SCREEN
A0BE|
A0BE| ; -----
A0BE| ; This section reads in the index block of the SOS.KERNEL file.
A0BE| ; -----
A0BE|
A0BE| A0 11          READIDXBLK LDY    #11          ; LOAD Y-REGISTER WITH $11
A0C0| B1 85          LDA    @IBBUFP,Y    ; LOAD KEY POINTER LOW BYTE
A0C2| AA            TAX          ; TRANSFER ACCUMULATOR TO X-REGISTER-BLOCK LOW
A0C3|                ; BYTE

```

10/31/89 9:45

HD:Apple ///:SOS Floppy Bootstrap Loader

Page 3

```

A0C3| C8          INY          ; INCREMENT Y-REGISTER
A0C4| B1 85      LDA          @IBBUFP,Y ; LOAD KEY POINTER HIGH BYTE
A0C6| A0 00      LDY          #00      ; LOAD Y-REGISTER WITH $00
A0C8| 84 85      STY          IBBUFP   ; STORE IN BUFFER POINTER LOW BYTE
A0CA| A0 0C      LDY          #0C      ; LOAD Y-REGISTER WITH $0C
A0CC| 84 86      STY          IBBUFP+1 ; STORE IN BUFFER POINTER HIGH BYTE
A0CE| 20 1DA1    JSR          READBLK  ; JUMP TO READ A BLOCK FROM FLOPPY DISK DRIVE
A0D1|
A0D1| ;-----
A0D1| ; This section reads in the first block of the SOS.KERNEL file.
A0D1| ;-----
A0D1| AE 000C      RDISOSKER LDX      IDXBLK1 ; LOAD X-REGISTER WITH INDEX BLOCK LOW BYTE
A0D4| AD 000D      LDA          IDXBLK2 ; LOAD ACCUMULATOR WITH INDEX BLOCK HIGH BYTE
A0D7| A0 00      LDY          #00      ; LOAD Y-REGISTER WITH $00
A0D9| 84 85      STY          IBBUFP   ; STORE IN BUFFER POINTER LOW BYTE
A0DB| A0 1E      LDY          #1E      ; LOAD Y-REGISTER WITH $1E
A0DD| 84 86      STY          IBBUFP+1 ; STORE IN BUFFER POINTER HIGH BYTE
A0DF| 20 1DA1    JSR          READBLK  ; JUMP TO READ A BLOCK FROM FLOPPY DISK DRIVE
A0E2|
A0E2| ;-----
A0E2| ; This section does a verification of the SOS.KERNEL file to make
A0E2| ; sure it is the proper SOS.KERNEL file. It checks for "SOS KRNL" in
A0E2| ; the first 8 bytes of the file.
A0E2| ;-----
A0E2| A0 08          FLVRFY      LDY          #08      ; LOAD Y-REGISTER WITH $08
A0E4| B9 FF1D      FLVRFYLP   LDA          LOADADR-1,Y ; LOAD ACCUMULATOR WITH BYTE FROM SOS.KERNEL
A0E7| D9 9CA1      CMP          FLVERIFY-1,Y ; COMPARE WITH VERIFICATION BYTE
A0EA| F003          BEQ          $010      ; BRANCH IF EQUAL
A0EC| 4C 6AA1      JMP          WRINKERERR ; JUMP TO WRITE INVALID KERNEL ERROR MESSAGE TO
A0EF|                                     ; SCREEN
A0EF| 88          $010      DEY          ; DECREMENT Y-REGISTER
A0F0| D0F2          BNE          FLVRFYLP   ; BRANCH IF NOT EQUAL TO ZERO TO CHECK REST OF 8
A0F2|                                     ; SOS.KERNEL BYTES
A0F2| ;-----
A0F2| ; This section reads in the SOS.KERNEL file.
A0F2| ;-----
A0F2| A9 01          RDSOSKER   LDA          #01      ; LOAD ACCUMULATOR WITH $01
A0F4| 85 E7          STA          INDXBLKCNT ; STORE IN INDEX BLOCK COUNT
A0F6| A4 E7          RDSOSKELP LDY          INDXBLKCNT ; LOAD Y-REGISTER WITH INDEX BLOCK COUNT
A0F8| BE 000C      LDX          IDXBLK1,Y ; LOAD X-REGISTER WITH BLOCK LOW BYTE
A0FB| B9 000D      LDA          IDXBLK2,Y ; LOAD ACCUMULATOR WITH BLOCK HIGH BYTE
A0FE| D004          BNE          $010      ; BRANCH IF BLOCK HIGH BYTE IS NOT EQUAL TO ZERO
A100| E0 00          CPX          #00      ; CHECK TO SEE IF BLOCK LOW BYTE IS NOT EQUAL TO
A102|                                     ; ZERO
A102| F007          BEQ          JUMPSOSKER ; BRANCH IF BLOCK LOW BYTE IS NOT EQUAL TO ZERO
A104| 20 1DA1      $010      JSR          READBLK  ; JUMP TO READ A BLOCK FROM FLOPPY DISK DRIVE
A107| E6 E7          INC          INDXBLKCNT ; INCREMENT INDEX BLOCK COUNT
A109| D0EB          BNE          RDSOSKELP   ; BRANCH IF INDEX BLOCK COUNT IS NOT EQUAL TO
A10B|                                     ; ZERO TO READ MORE OF THE SOS.KERNEL
A10B| ;-----
A10B| ; This section jumps to the SOS.KERNEL loader.
A10B| ;-----
A10B| 18          JUMPSOSKER CLC          ; CLEAR CARRY
A10C| A9 0E          LDA          #0E      ; LOAD ACCUMULATOR WITH $0E
A10E| 6D 081E      ADC          OFFSET    ; ADD OFFSET LOW BYTE
A111| 85 E8          STA          SOSJMPADR ; STORE IN SOS JUMP ADDRESS LOW BYTE
A113| A9 1E          LDA          #1E      ; LOAD ACCUMULATOR WITH $1E
A115| 6D 091E      ADC          OFFSET+1  ; ADD OFFSET HIGH BYTE
A118| 85 E9          STA          SOSJMPADR+1 ; STORE IN SOS JUMP ADDRESS HIGH BYTE
A11A| 6C E800      JMP          @SOSJMPADR ; JUMP TO SOS.KERNEL LOADER
A11D|
A11D| ;-----
A11D| ; This section reads a block of data from the floppy disk drive.
A11D| ; On entry the x-register contains the block low byte and the
A11D| ; accumulator contains the block high byte.
A11D| ;-----
A11D| 86 83          READBLK   STX          IBTRK  ; STORE BLOCK LOW BYTE IN TRACK NUMBER
A11F| 4A          LSR          A          ; DIVIDE BLOCK BY 8 TO GET TRACK NUMBER
A120| 66 83          ROR          IBTRK
A122| 4A          LSR          A
A123| 66 83          ROR          IBTRK
A125| 4A          LSR          A
A126| 66 83          ROR          IBTRK
A128| 8A          TXA          ; TRANSFER X-REGISTER WHICH CONTAINS THE BLOCK
A129|                                     ; LOW BYTE TO ACCUMULATOR
A129| 29 07          AND          #07      ; MASK OFF BITS 3,4,5,6,7
A12B| AA          TAX          ; TRANSFER ACCUMULATOR TO X-REGISTER
A12C| BD A0F4      LDA          SECTABL,X ; LOAD ACCUMULATOR WITH PROPER SECTOR TO READ
A12F| 85 84          STA          IBSECT    ; STORE IN SECTOR NUMBER
A131| A9 01          LDA          #01      ; LOAD ACCUMULATOR WITH $01
A133| 85 87          STA          IBCMD    ; STORE IN COMMAND NUMBER
A135| A9 00          LDA          #00      ; LOAD ACCUMULATOR WITH $00
A137| 85 82          STA          IBDRVN    ; STORE IN DRIVE NUMBER

```

10/31/89 9:45

HD:Apple ///:SOS Floppy Bootstrap Loader

Page 4

```

A139| 20 00F0      JSR      REGRWTS      ; JUMP TO READ A SECTOR FROM FLOPPY DISK
A13C| 9005        BCC      $010      ; BRANCH IF NO DISK ERRORS OCCURED
A13E| A2 FF      LDX      #0FF      ; LOAD ACCUMULATOR WITH $FF
A140| 9A          TXS          ; TRANSFER X-REGISTER TO STACK POINTER
A141| B03B        BCS      WRDISKERR  ; BRANCH TO WRITE DISK ERROR MESSAGE TO SCREEN
A143| E6 86      INC      IBBUFF+1  ; INCREMENT BUFFER POINTER HIGH BYTE
A145| E6 84      INC      IBSECT    ; INCREMENT SECTOR NUMBER
A147| E6 84      INC      IBSECT    ; INCREMENT SECTOR NUMBER
A149| 20 00F0      JSR      REGRWTS  ; JUMP TO READ A SECTOR FROM FLOPPY DISK
A14C| 9005        BCC      $020      ; BRANCH IF NO DISK ERRORS OCCURED
A14E| A2 FF      LDX      #0FF      ; LOAD ACCUMULATOR WITH $FF
A150| 9A          TXS          ; TRANSFER X-REGISTER TO STACK POINTER
A151| B02B        BCS      WRDISKERR  ; BRANCH TO WRITE DISK ERROR MESSAGE TO SCREEN
A153| E6 86      INC      IBBUFF+1  ; INCREMENT BUFFER POINTER HIGH BYTE
A155| 60          RTS          ; RETURN TO CALLER
A156|
A156| -----
A156| ; This section writes the not found error message to the screen.
A156| -----
A156|
A156| A2 1B      WRNNTFNDERR LDX      #1B      ; LOAD X-REGISTER WITH $1B
A158| A0 21      LDY      #21      ; LOAD Y-REGISTER WITH $21
A15A| BD A4A1    $010      LDA      NTFNDERR-1,X ; LOAD ACCUMULATOR WITH NOT FOUND ERROR MESSAGE
A15D|           ; BYTE
A15D| 99 2806      STA      SCREENLOC,Y ; WRITE IT TO THE SCREEN
A160| 88      DEY          ; DECREMENT Y-REGISTER
A161| CA      DEX          ; DECREMENT X-REGISTER
A162| D0F6      BNE      $010      ; BRANCH IF MORE CHARACTERS TO WRITE ON SCREEN
A164| AD 40C0    LDA      IOBEEP     ; BEEP SPEAKER
A167| 4C 67A1    $020      JMP      $020     ; HANG FOREVER !!
A16A|
A16A| -----
A16A| ; This section writes the invalid kernel error message to the screen.
A16A| -----
A16A|
A16A| A2 13      WRINKERERR LDX      #13      ; LOAD X-REGISTER WITH $13
A16C| A0 1D      LDY      #1D      ; LOAD Y-REGISTER WITH $1D
A16E| BD BFA1    $010      LDA      INVKEERR-1,X ; LOAD ACCUMULATOR WITH INVALID KERNEL ERROR
A171|           ; MESSAGE BYTE
A171| 99 2806      STA      SCREENLOC,Y ; WRITE IT TO THE SCREEN
A174| 88      DEY          ; DECREMENT Y-REGISTER
A175| CA      DEX          ; DECREMENT X-REGISTER
A176| D0F6      BNE      $010      ; BRANCH IF MORE CHARACTERS TO WRITE ON SCREEN
A178| AD 40C0    LDA      IOBEEP     ; BEEP SPEAKER
A17B| 4C 7BA1    $020      JMP      $020     ; HANG FOREVER !!
A17E|
A17E| -----
A17E| ; This section writes the disk error message to the screen.
A17E| -----
A17E|
A17E| A2 0A      WRDISKERR LDX      #0A      ; LOAD X-REGISTER WITH $0A
A180| A0 18      LDY      #18      ; LOAD Y-REGISTER WITH $18
A182| BD D2A1    $010      LDA      DISKERR-1,X ; LOAD ACCUMULATOR WITH DISK ERROR MESSAGE BYTE
A185| 99 2806      STA      SCREENLOC,Y ; WRITE IT TO THE SCREEN
A188| 88      DEY          ; DECREMENT Y-REGISTER
A189| CA      DEX          ; DECREMENT X-REGISTER
A18A| D0F6      BNE      $010      ; BRANCH IF MORE CHARACTERS TO WRITE ON SCREEN
A18C| AD 40C0    LDA      IOBEEP     ; BEEP SPEAKER
A18F| 4C 8FA1    $020      JMP      $020     ; HANG FOREVER !!
A192|
A192| -----
A192| ; STORAGE FOR THE ERROR MESSAGE AND FILE VERIFICATION ROUTINES
A192| -----
A192|
A192| 0A      FLNMELEN .BYTE 0A
A193| 53 4F 53 2E 4B 45 52 FLNME .ASCII "SOS.KERNEL"
A19A| 4E 45 4C
A19D| 53 4F 53 20 4B 52 4E FLVERIFY .ASCII "SOS KRNL"
A1A4| 4C
A1A5| 46 49 4C 45 20 27 53 NTFNDERR .ASCII "FILE 'SOS.KERNEL' NOT FOUND"
A1AC| 4F 53 2E 4B 45 52 4E
A1B3| 45 4C 27 20 4E 4F 54
A1BA| 20 46 4F 55 4E 44
A1C0| 49 4E 56 41 4C 49 44 INVKEERR .ASCII "INVALID KERNEL FILE"
A1C7| 20 4B 45 52 4E 45 4C
A1CE| 20 46 49 4C 45
A1D3| 44 49 53 4B 20 45 52 DISKERR .ASCII "DISK ERROR"
A1DA| 52 4F 52
A1DD|
A1DD| .END

```

## SYMBOL TABLE DUMP

AB - Absolute	LB - Label	UD - Undefined	MC - Macro
RF - Ref	DF - Def	PR - Proc	FC - Func
PB - Public	PV - Private	CS - Consts	

```

BOOTSTRA PR ---- | BREG      AB FFEF | DISKERR  LB A1D3 | ENTRY    LB A000 | EREG      AB FFDF |

```



10/31/89 9:45

HD:Apple ///:SOS Floppy Bootstrap Loader

Page 5

```

FILECNT AB 00E5 | FIRSTPAG AB 2000 | FLNME LB A193 | FLNMELEN LB A192 | FLVERIFY LB A19D |
FLVRFY LB A0E2 | FLVRFYLP LB A0E4 | IBBUFF AB 0085 | IBBUFFTM AB 00E3 | IBCMD AB 0087 |
IBDRVN AB 0082 | IBSECT AB 0084 | IBTRK AB 0083 | IDXBLK1 AB 0C00 | IDXBLK2 AB 0D00 |
INDXBLKC AB 00E7 | INVKEERR LB A1C0 | IOBEEP AB C040 | JUMPSOSK LB A10B | KBDSTROB AB C010 |
LOADADR AB 1E00 | MAINBUFF AB A200 | NMIVECTO AB FFCA | NTFNDERR LB A1A5 | OFFSET AB 1E08 |
RDISOSKE LB A0D1 | RDSOSDIR LB A02E | RDSOSKEL LB A0F6 | RDSOSKER LB A0F2 | READBLK LB A11D |
READIDXB LB A0BE | READSOSD LB A024 | REGRWTS AB F000 | RETINT AB 0040 | SCREENLO AB 0628 |
SECTABL AB F4A0 | SOSJMPAD AB 00E8 | SRCHLP LB A06C | SRCHSOSK LB A047 | WRDISKER LB A17E |
WRINKERE LB A16A | WRNTFNDE LB A156 |

```

Assembly complete: 363 lines  
 0 Errors flagged on this Assembly

---

#### 6502 OPCODE STATIC FREQUENCIES

---

```

ADC : 4 | ****
AND : 3 | ***
BCC : 2 | **
BCS : 3 | ***
BEQ : 6 | *****
BIT : 1 m *
BNE : 15 | *****
CLC : 2 | **
CLD : 1 m *
CMP : 4 | ****
CPX : 2 | **
DEC : 3 | ***
DEX : 5 | *****
DEY : 5 | *****
INC : 6 | *****
INY : 2 | **
JMP : 7 | *****
JSR : 6 | *****
LDA : 37 M *****
LDX : 12 | *****
LDY : 14 | *****
LSR : 3 | ***
ORA : 1 m *
PHP : 1 m *
PLP : 1 m *
ROR : 3 | ***
RTS : 1 m *
SBC : 2 | **
SEC : 1 m *
SEI : 1 m *
STA : 23 | *****
STX : 2 | **
STY : 6 | *****
TAX : 3 | ***
TAY : 1 m *
TXA : 1 m *
TXS : 3 | ***

```

Minimum frequency = 1  
 Maximum frequency = 37

Average frequency = 5

Unused opcodes:

ASL BMI BPL BRK BVC BVS CLI CLV CPY EOR INX NOP PHA PLA ROL RTI  
 SED TSX TYA

Program opcode usage: 66 %

---

(1.00) That's all, Folks ...

---

*seems like an  
early version*

Apple /// Computer Information

# APPLE /// SOS BOOTSTRAP LOADER HEXADECIMAL DUMP

Source

DISK1.dofile as found with Chris Smolinski's Macintosh SARA emulator application

Printed by David T. Craig • December 1997

This hex dump, which was produced by the Apple Macintosh MPW DumpFile tool, lists the Apple /// SOS Bootstrap Loader. This 512 byte loader exists at block 0 of SOS disks and is loaded by the Apple /// ROM into memory addresses \$A000-\$A1FF. This code's purpose is to begin the loading of SOS from the floppy disk into the ///'s memory.

```

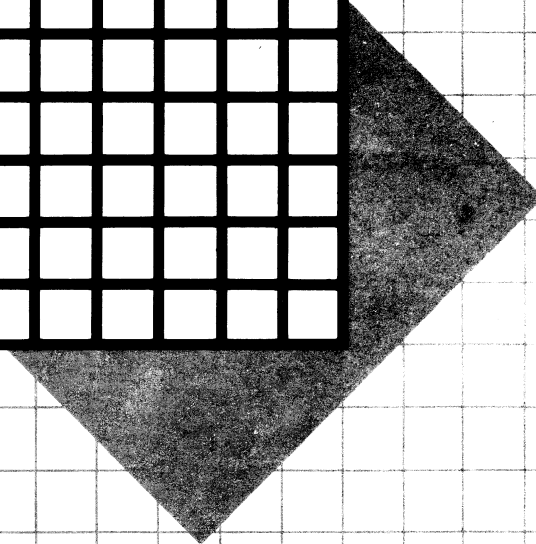
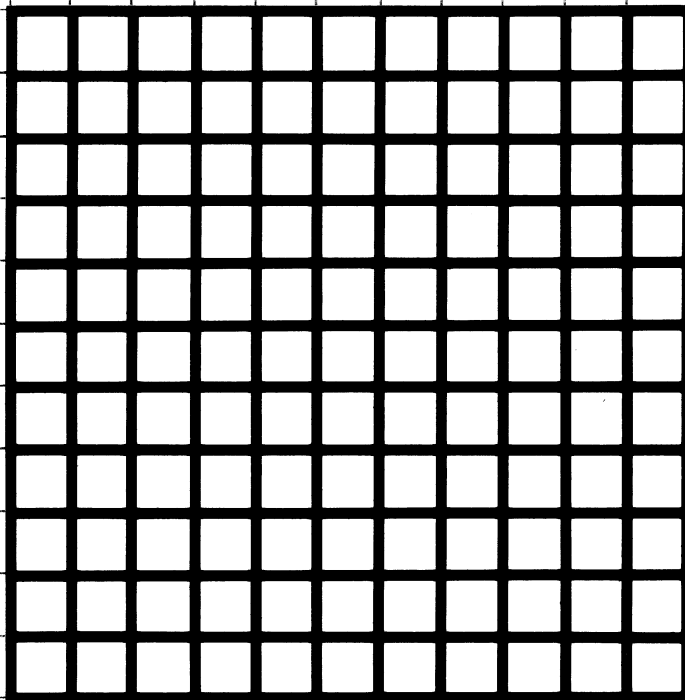
0: 4C 6E A0 53 4F 53 20 42 4F 4F 54 20 20 31 2E 31 Ln+SOS.BOOT..1.1
10: 20 0A 53 4F 53 2E 4B 45 52 4E 45 4C 20 20 20 20 ..SOS.KERNEL....
20: 20 53 4F 53 20 4B 52 4E 4C 49 2F 4F 20 45 52 52 .SOS.KRNLI/O.ERR
30: 4F 52 08 00 46 49 4C 45 20 27 53 4F 53 2E 4B 45 OR..FILE.'SOS.KE
40: 52 4E 45 4C 27 20 4E 4F 54 20 46 4F 55 4E 44 25 RNEL'.NOT.FOUND%
50: 00 49 4E 56 41 4C 49 44 20 4B 45 52 4E 45 4C 20 .INVALID.KERNEL.
60: 46 49 4C 45 3A 00 00 0C 00 1E 0E 1E 04 A4 78 D8 FILE:.....$xÿ
70: A9 77 8D DF FF A2 FB 9A 2C 10 C0 A9 40 8D CA FF @wçfl~ç'ö,.,ç@æç ~
80: A9 07 8D EF FF A2 00 CE EF FF 8E 00 20 AD 00 20 @.çÖ~ç.ÇÖ~é..≠..
90: D0 F5 A9 01 85 E0 A9 00 85 E1 A9 00 85 A9 A2 -1@.Ö+@.Ö.ÖÖ@ç
A0: 85 86 20 BE A1 E6 E0 A9 00 85 E6 E6 86 E6 86 E6 ÖÜ.æ°Ê+@.ÖÊÜÊÜÊ
B0: E6 20 BE A1 A0 02 B1 85 85 E0 C8 B1 85 85 E1 D0 Ê.æ°+±ÖÖ+±ÖÖ.-
C0: EA A5 E0 D0 E6 AD 6C A0 85 E2 AD 6D A0 85 E3 18 Í.+Ê≠1+Ö,≠m+Ö,,.
D0: A5 E3 69 02 85 E5 38 A5 E2 ED 23 A4 85 E4 A5 E5 .„i.ÖÂ8.,Ì#SÖ%.Â
E0: E9 00 85 E5 A0 00 B1 E2 29 0F CD 11 A0 D0 21 A8 È.ÖÂ+.±,).Ö.+!@
F0: B1 E2 D9 11 A0 D0 19 88 D0 F6 A0 00 B1 E2 29 F0 ±,ÿ.+-.à-^+.±,)
100: 53 4F 53 20 4B 52 4E 4C 62 00 01 00 0E 2E 44 31 SOS.KRNlb.....D1
110: 2F 53 4F 53 2E 49 4E 54 45 52 50 AA A5 A0 F9 A0 /SOS.INTERP™.††
120: A0 A5 A0 A0 A5 A0 A0 C5 A0 A0 98 A0 F0 A1 A0 CC †.††.††≈††ò†
130: A0 A0 C5 A0 A0 A0 A0 EE A0 A0 C4 0E 2E 44 31 ††≈†††††Ó††f...D1
140: 2F 53 4F 53 2E 44 52 49 56 45 52 FF 9A A0 FF 9A /SOS.DRIVER~ò†~ö
150: A0 A0 A0 A0 D0 A0 A0 C1 A0 A0 8A A0 A0 F9 A0 C1 ††††-††;††ä††††;
160: E9 A0 9E A1 A0 F5 A0 A0 A5 A0 A0 88 00 00 88 0C È†û°†††.††à..à.
170: A9 00 AA 9D 00 1A 9D 00 16 9D 00 1B 9D 00 18 9D @.™ù..ù..ù..ù..ù
180: 00 14 9D 00 01 CA D0 EB A9 30 8D DF FF A2 FB 9A ..ù..-Î@0çfl~ç'ö
190: A9 1A 8D D0 FF 20 D4 1F AD DF FF 29 10 09 28 8D @.ç-~.~.≠fl~)..(ç
1A0: DF FF A2 FF 9A A9 1A 8D D0 FF AD 01 19 8D EF FF fl~ç~ö@.ç-~≠..çÖ~
1B0: 6C 02 00 AA AD EF FF 48 8E EF FF A5 27 05 26 F0 1..™≠Ö~HéÖ~.~.®
1C0: 33 A5 26 D0 02 C6 27 C6 26 18 A5 23 65 27 85 23 3&-..Δ'Δ&..#e'Ö#
1D0: A5 25 65 27 85 25 E6 27 A4 26 F0 07 B1 22 91 24 *%e'Ö%Ê'S&®.±"è$
1E0: 88 D0 F9 B1 22 91 24 88 C6 23 C6 25 C6 27 D0 EC à-~±"è$àΔ#ΔΔ'-Î
1F0: E6 23 E6 25 68 8D EF FF 60 18 A5 24 65 10 85 10 Ê#Ê%hçÖ~`.~$e.Ö.

```

###

**Apple III**

System Data Sheet



EX LIBRIS: David T. Craig  
736 Edgewater  
[# \_\_\_\_\_] Wichita, Kansas 67230 (USA)

# The Apple III

## The Most Powerful Personal Computer In Its Class

Too much information? Not enough time? The Apple III was created to meet the information-handling needs of decision makers at all levels, in every size and kind of company. And the Apple III can grow with you, so as your responsibilities increase, your ability to handle them stays one step ahead.

You can use the power of your Apple III to create financial forecasts, budgets, and reports; for accounting, resource management, and project scheduling; in electronic communications, software development, and computer-assisted training. Over 400 business programs are available today for the Apple III — plus the extensive library of CP/M® business software (with the Apple SoftCard™ III). And most Apple II Plus programs will run in the Apple III's "emulation" mode.

The Apple III: the personal computer for business.



### Powerful features for professional needs.

The Apple III is ready to go as soon as you unpack it, connect a monitor, and provide power. No interface cards are required, and you don't have to open the computer. The Apple III already has a built-in disk drive, video outputs for color and monochromatic displays, and a numeric keypad.

Other built-in features include:

**Large User Memory.** The Apple III's 256K of internal memory means you can work with sophisticated programs and large financial and text documents, quickly and efficiently.

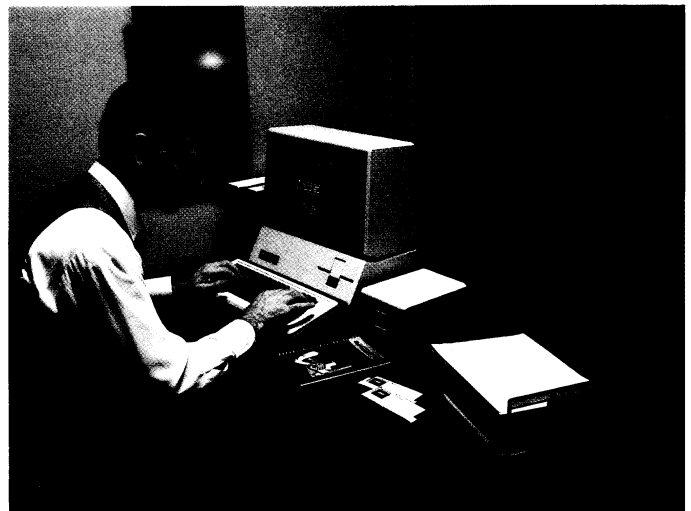
**Color Graphics.** The 16-color graphics capability of the Apple III allows you to grasp the meaning of charts and graphs quickly. If you're not using a color monitor, your information is displayed in 16 shades, so the facts still stand out clearly.

**High-Resolution Video.** The Apple III displays 107,520 points of information on the screen (560 horizontal x 192 vertical) in text and monochromatic graphics modes. While text is normally presented in an 80-column by 24-line monochromatic format, it can be switched to 40-column monochromatic or color-on-color.

**Accessory Connectors.** The most common accessories plug right into the Apple III. Connectors and interfacing hardware are already built in for the Apple Daisy Wheel Printer (or other serial printer), the Apple Silentype Printer, external floppy disk drives, color and monochromatic video displays (NTSC, RGB, and composite), a modem, and hand controls. The Apple III also has four inside expansion slots for additional accessories.

### Apple III Sophisticated Operating System: it does it all for you.

Today . . . you can bring financial models into reports, insert names into form letters automatically, and turn numbers into charts, because the Apple III's Sophisticated Operating System (SOS) treats all your files identically. And, since applications programs written for the Apple III are all based on this common SOS formatting, you can combine them on a ProFile™ mass storage system and move freely from one to another. The uniformity of SOS also provides an ideal environment for software development.



Tomorrow . . . you can expand your Apple III elegantly. Because SOS controls all communications with accessories, you don't have to figure out how to make the computer work with a new printer, disk drive, or modem. SOS does this for you by using special files known as "device drivers." Apple III programs come with the most commonly-used device drivers, and you can make programs compatible with new equipment by copying a driver file for the new device onto a program disk. Your software can just as easily be revised to take advantage of SOS upgrades, and of hardware enhancements to the computer itself.

### **Installation's easy. Learning is, too.**

Because the Apple III already has a built-in disk drive and video connector, the computer is ready to work as soon as you connect a monitor and provide power. Then, Apple makes it just as easy to learn how to use it. A comprehensive Owner's Guide gets you started, and a System Demonstration disk introduces you to the computer's text editing and graphics capabilities. Reference manuals and SOS utilities disks are included for more advanced needs, and additional tutorials on the computer and its programs are also available.



### **Durable. Dependable. Reliable.**

The Apple III is dependable, inside and out. Outside, it has a rugged die-cast aluminum chassis. Inside, electronics based on advanced microprocessor circuitry assure reliable operation. The system also meets UL and CSA standards.

Every time the computer is powered up, it performs a brief self-diagnostic routine. Should problems arise, help is close at hand, because of Apple's extensive dealer/service network. Average turnaround time on Apple III servicing is less than one day.



### **Standard Features**

- 256K internal memory (RAM)
- Built-in disk drive
- Custom microprocessor circuitry
- High-resolution color graphics (16 colors)
- 80-column, 24-line text display, upper and lower case
- Contoured typewriter-style keyboard; 61 keys; all 128 ASCII codes; auto-repeat on all keys
- Numeric keypad (13 keys)
- Special-purpose keys: Up-Arrow, Down-Arrow, Left-Arrow, Right-Arrow; programmable Open-Apple and Solid-Apple; TAB; SHIFT; ALPHA LOCK; CONTROL; RETURN; ENTER; ESCAPE
- Quick-connect plugs for disk drives, video and audio devices, serial printers, modems, and hand controls
- Four expansion slots for accessory interface cards
- Apple II Plus emulation mode
- High-quality sound generation
- Lockable case
- Self-testing diagnostics on powerup

### **Optional Accessories**

- Monitor III or color monitor
- Apple Daisy Wheel Printer
- Apple Dot Matrix Printer
- Apple Silentype Printer
- Disk III floppy-disk drives
- ProFile hard-disk systems
- Apple SoftCard III System (for CP/M capability)
- Parallel Card III
- Serial Card III
- Programming languages (Business BASIC, Pascal, COBOL)
- Cursor III joysticks

## Technical Specifications

### ■ Video Display:

Text and graphics may be displayed simultaneously. Graphics modes:

- 280 x 192, 16 colors (with some limitations);
- 280 x 192, monochromatic;
- 140 x 192, 16 colors;
- 560 x 192, monochromatic;
- All Apple II modes (in emulation)

Graphics commands allow either of two screen buffers to be displayed.

Text modes:

- 80-column, 24-line monochromatic;
- 40-column, 24-line, 16-color foreground and background;
- 40-column, 24-line monochromatic.

All text modes have a software-definable, 128-character set (upper and lower-case), with normal or inverse display.

### ■ Central Processing Unit (CPU):

The custom-designed microprocessor circuitry of the Apple III utilizes the 6502B as one of its major components. Other circuitry provides extended addressing capability, relocatable stack, zero page, and memory mapping.

Type:

6502B.

Clock Speed:

1.4 MHz average; 1.8 MHz maximum.

Operations Per Second (8-bit):

Up to 750,000.

Data Bus:

Two 8-bit formats, combined for extended addressing.

Address Bus:

19 bits.

Address Range:

262,144 bytes (256K).

Registers:

Accumulator (A); Index Registers (X,Y); Stack Pointer (S); Program Counter (PC); Environmental Register (E); Bank (B); Zero Page (Z); Processor Status (P).

### ■ Memory:

256K dynamic RAM;

4K ROM (initialization and self-test diagnostics).

### ■ SOS (Sophisticated Operating System):

Handles all system I/O;

Can be configured to handle standard or custom I/O devices and peripherals by adding or deleting "device drivers";

All languages and application programs access data through the SOS file system.

### ■ Inputs and Outputs:

Keyboard:

- 61 keys on main keyboard;
- 13 keys on numeric keypad;
- Full 128-character, ASCII encoded;
- All keys have automatic repeat;
- Four directional-arrow keys with two-speed repeat;
- Two user-definable Apple keys;
- Seven other special keys: SHIFT, CONTROL, ALPHA LOCK, TAB, ESCAPE, RETURN, ENTER.

Storage Devices:

- One 5.25-inch floppy disk drive built in, 140K (143, 360) bytes per diskette;
- Three additional drives can be connected by daisy-chain cable (Total: 560K bytes on-line storage);
- Up to four ProFile hard-disk drives (5 megabytes each) may be added with plug-in interface cards.

Video Output:

- RCA phono connector for NTSC monochromatic composite video;
- DB-15 connector for:
  - NTSC color composite video;
  - NTSC monochromatic composite video;
  - RGB color video;
  - Composite sync signal;
  - Power supply voltages.
- Color signals appear as 16-level grey scale on monochromatic displays.

Audio Output:

- Built-in two-inch speaker; miniature phono jack on back panel;
- Driven by 6-bit D/A converter or fixed-frequency "beep" generator.

Serial (Printer/Modem) Port:

- RS-232C compatible, DB-25 female connector;
- Software-selectable baud rate and duplex mode.

One port may be used for the Silentype printer.

One port may be used for the Silentype printer.

Expansion:

- Four 50-pin expansion slots (fully buffered, with interrupt and DMA priority structure).

Joystick/Silentype Ports:

- Two DB-9 connectors.

### ■ Languages Available:

Apple Business BASIC, Apple III Pascal, Apple III COBOL.

### ■ Emulation Mode:

Provides hardware emulation of 48K byte Apple II Plus. Allows most Apple II programs, with the exception of Pascal and FORTRAN, to run without modification.

### ■ Electrical Specifications:

The Apple III's power cord should be plugged into a three-wire 110-120 volt outlet.

### ■ Physical Specifications:

- Height: 4.8 inches (12.20 cm)
- Depth: 18.2 inches (46.22 cm)
- Width: 17.5 inches (44.45 cm)
- Weight: 26 lbs. (11.8 kg)

The Apple III meets the following agency regulations:

- UL 114 — Office Appliances and Business Equipment.

CSA 22.2, No. 154 — Data Processing Equipment.

## The Apple III Personal Computer System Package

### U.S. Order Number A3S0256

With your order for an Apple III

System you will receive:

- 256K Apple III;
- Power cord;
- Monitor cable;
- System Demonstration disk;
- System Utilities disk;
- System Utilities Data disk (contains device driver files, character sets, and keyboard layouts);
- Apple II Plus Emulation disk;
- Owner's Guide;
- Standard Device Drivers Manual;
- Warranty and service information.

Specifications or products may change without notice.

Apple, the Apple logo, ProFile, and Silentype are trademarks of Apple Computer, Inc. SoftCard is a trademark of MicroSoft Corporation.

CP/M is a trademark of Digital Research, Inc.



20525 Mariani Avenue  
Cupertino, California 95014  
(408) 996-1010  
TLX 171-576