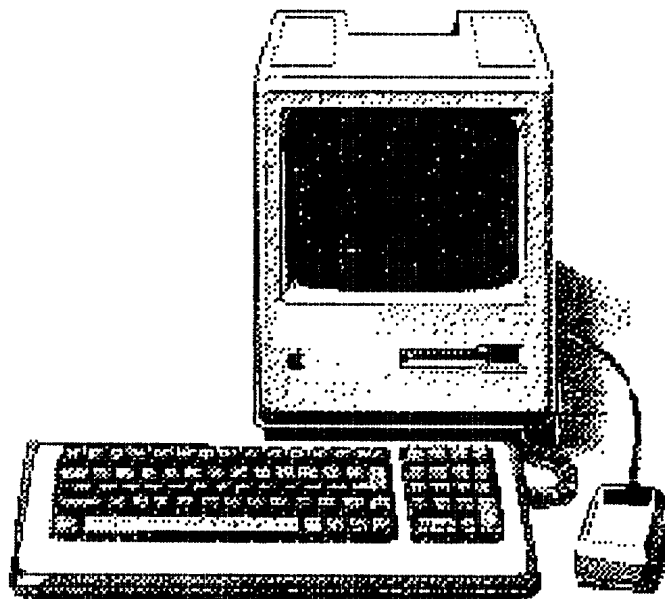


**🍏 Apple Macintosh Technical Information**

# Macintosh Hardware Description

## Macintosh 128K Macintosh 512K



---

MACINTOSH USER EDUCATION

---

THE MACINTOSH HARDWARE

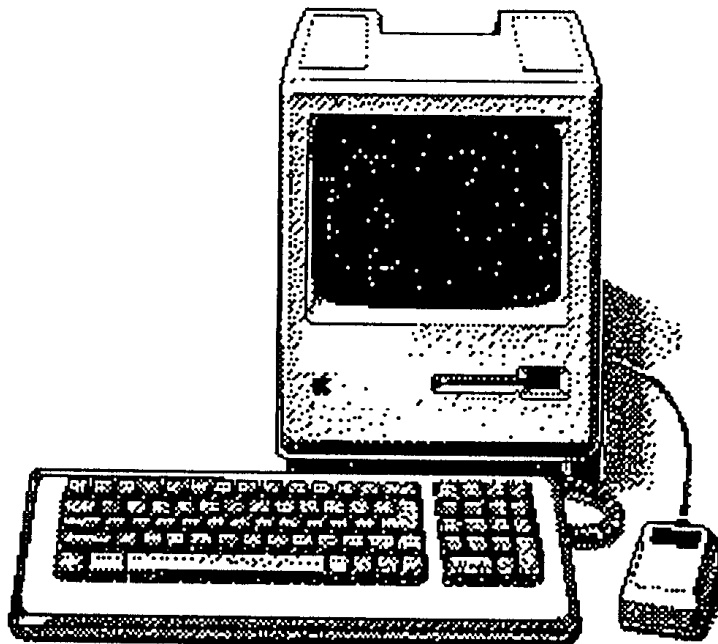
/HARDWARE/HDWR

---

Modification History: First Draft Chris Espinosa & Nick Turner 2/4/85  
Second Draft Brian Howard 2/13/85

---

## 128K and 512K models



2 Macintosh Hardware

---

TABLE OF CONTENTS

---

3 About This Chapter  
n Overview of the Hardware  
n The Video Interface  
n The Sound Generator  
n Diagram  
n The SCC  
n Diagram  
n The Mouse  
n Diagram  
n The Keyboard and Keypad  
n Keyboard Communication Protocol  
n Keypad Communication Protocol  
n The Disk Interface  
n Controlling the Disk-State Control Lines  
n Reading the Disk Registers  
n Writing to the Disk Registers  
n Explanations of the Disk Registers  
n The Real-Time Clock  
n Accessing the Clock Chip  
n The One-Second Interrupt  
n The VIA  
n VIA Register A  
n VIA Register B  
n The VIA Peripheral Control Register  
n The VIA Timers  
n VIA Interrupts  
n Other VIA Registers  
n System Startup  
n Summary

Copyright (c) 1985 Apple Computer, Inc. All rights reserved.  
Distribution of this draft in limited quantities does not constitute  
publication.

---

**ABOUT THIS CHAPTER**

---

This chapter provides a basic description of the hardware of the Macintosh 128K and 512K computers. It gives you information that you'll need to connect other devices to the Macintosh and to write device drivers or other low-level programs. It will help you figure out which technical documents you'll need to design peripherals; in some cases, you'll have to obtain detailed specifications from the manufacturers of the various interface chips.

This chapter is oriented toward assembly-language programmers. It assumes you're familiar with the basic operation of microprocessor-based devices. Knowledge of the Macintosh Operating System will also be helpful.

(warning)

Only the Macintosh 128K and 512K are covered in this chapter. In particular, note that the memory addresses and screen size are different on the Macintosh XL (and may be different in future versions of the Macintosh). To maintain software compatibility across the Macintosh line, and to allow for future changes to the hardware, you're **strongly advised** to use the Toolbox and Operating System routines wherever possible.

To learn how your program can determine which hardware environment it's operating in, see the description of the `Environs` procedure in the Operating System Utilities chapter.

---

**OVERVIEW OF THE HARDWARE**

---

The Macintosh computer contains a Motorola MC68000 microprocessor clocked at 7.8336 megahertz, random access memory (RAM), read-only memory (ROM), and several chips that enable it to communicate with external devices. There are five I/O devices: the video display; the sound generator; a Synertek SY6522 Versatile Interface Adapter (VIA) for the mouse and keyboard; a Zilog Z8530 Serial Communications Controller (SCC) for serial communication; and an Apple custom chip, called the IWM ("Integrated Woz Machine") for disk control.

The Macintosh uses memory-mapped I/O, which means that each device in the system is accessed by reading or writing to specific locations in the address space of the computer. Each device contains logic that recognizes when it's being accessed and responds in the appropriate manner.

The MC68000 can directly access 16 megabytes (Mb) of address space. In the Macintosh, this is divided into four equal sections. The first four Mb are for RAM, the second four Mb are for ROM, the third are for the SCC, and the last four are for the IWM and the VIA. Since each of the devices within the blocks has far fewer than four Mb of

## 4 Macintosh Hardware

individually addressable locations or registers, the addresses within each block "wrap around" and are repeated several times within the block.

RAM is the "working memory" of the system. Its base address is address  $\emptyset$ . The first 256 bytes of RAM (addresses  $\emptyset$  through  $\$FF$ ) are used by the MC68000 as exception vectors; these are the addresses of the routines that gain control whenever an exception such as an interrupt or a trap occurs. (The summary at the end of this chapter includes a list of all the exception vectors.) RAM also contains the system and application heaps, the stack, and other information used by applications. In addition, the following hardware devices share the use of RAM with the MC68000:

- the video display, which reads the information for the display from one of two screen buffers
- the sound generator, which reads its information from one of two sound buffers
- the disk speed controller, which shares its data space with the sound buffers

The MC68000 accesses to RAM are interleaved (alternated) with the video display's accesses during the active portion of a screen scan line (video scanning is described in the next section). The sound generator and disk speed controller are given the first access after each scan line. At all other times, the MC68000 has uninterrupted access to RAM, increasing the average RAM access rate to about 6 megahertz (MHz).

ROM is the system's permanent read-only memory. Its base address,  $\$400000$ , is available as the constant romStart and is also stored in the global variable ROMBase. ROM contains the routines for the Toolbox and Operating System, and the various system traps. Since the ROM is used exclusively by the MC68000, it's always accessed at the full processor rate of 7.83 MHz.

The address space reserved for the device I/O contains blocks devoted to each of the devices within the computer. This region begins at address  $\$800000$  and continues to the highest address at  $\$FFFFFF$ .

(note)

Since the VIA is involved in some way in almost every operation of the Macintosh, the following sections frequently refer to the VIA and VIA-related constants. The VIA itself is described later, and all the constants are listed in the summary at the end of this chapter.

---

THE VIDEO INTERFACE

---

The video display is created by a moving electron beam that scans across the screen, turning on and off as it scans in order to create black and white pixels. Each pixel is a square, approximately 1/74 inch on a side.

To create a screen image, the electron beam starts at the top left corner of the screen (see Figure 1). The beam scans horizontally across the screen from left to right, creating the top line of graphics. When it reaches the last pixel on the right end of the top line it turns off, and continues past the last pixel to the physical right edge of the screen. Then it flicks invisibly back to the left edge and moves down one scan line. After tracing across the black border, it begins displaying the data in the second scan line. The time between the display of the rightmost pixel on one line and the leftmost pixel on the next is called the horizontal blanking interval. When the electron beam reaches the last pixel of the last (342nd) line on the screen, it traces out to the right edge and then flicks up to the top left corner, where it traces the left border and then begins once again to display the top line. The time between the last pixel on the bottom line and the first one on the top line is called the vertical blanking interval. At the beginning of the vertical blanking interval, the VIA generates a vertical blanking interrupt.

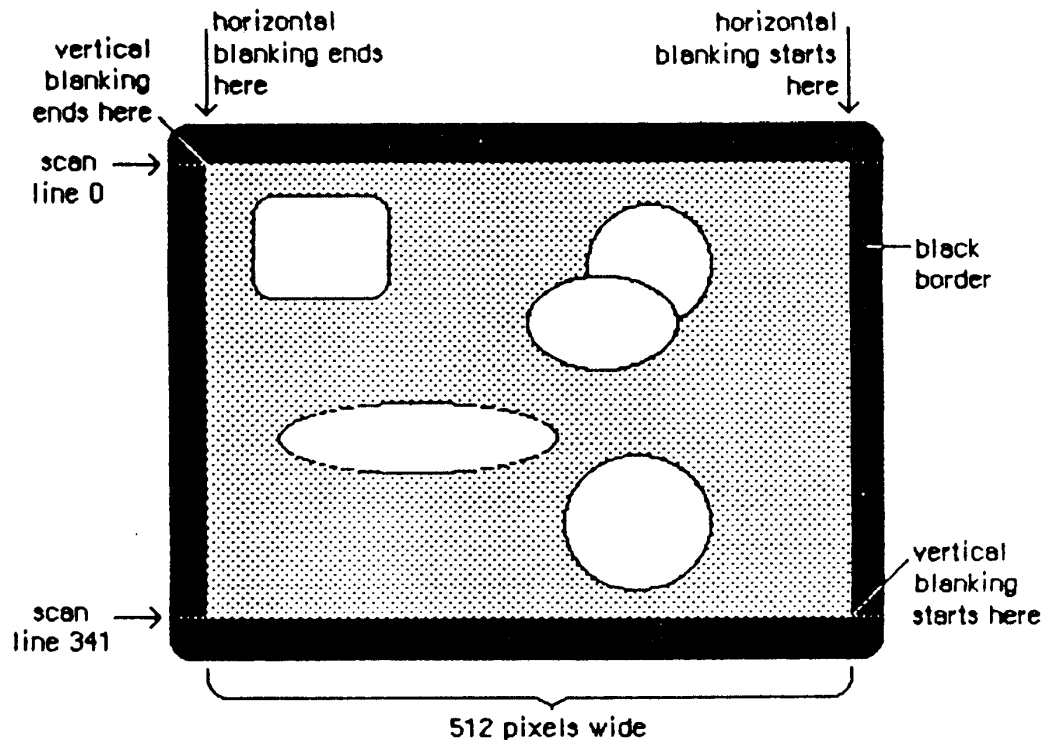


Figure 1. Video Scanning Pattern

The pixel clock rate (the frequency at which pixels are displayed) is 15.6672 MHz, or about .064 microseconds (usec) per pixel. For each scan line, 512 pixels are drawn on the screen, requiring 32.68 usec. The horizontal blanking interval takes the time of an additional 192 pixels, or 12.25 usec. Thus, each full scan line takes 44.93 usec, which means the horizontal scan rate is 22.25 kilohertz.

A full screen display consists of 342 horizontal scan lines, occupying 15367.65 usec, or about 15.37 milliseconds (msec). The vertical blanking interval takes the time of an additional 28 scan lines-- 1258.17 usec, or about 1.26 msec. This means the full screen is redisplayed once every 16625.8 usec. That's about 16.6 msec per frame, which means the vertical scan rate (the full screen display frequency) is 60.15 hertz.

The video generator uses 21,888 bytes of RAM to compose a bit-mapped video image 512 pixels wide by 342 pixels tall. Each bit in this range controls a single pixel in the image: A 0 bit is white, and a 1 bit is black.

There are two screen buffers (areas of memory from which the video circuitry can read information to create a screen display): the main buffer and the alternate buffer. The starting addresses of the screen buffers depend on how much memory you have in your Macintosh. In a Macintosh 128K, the main screen buffer starts at \$1A700 and the alternate buffer starts at \$12700; for a 512K Macintosh, add \$60000 to these numbers.

(warning)

To be sure you don't use the wrong area of memory and to maintain compatibility with future Macintosh systems, you should get the video base address and bit map dimensions from screenBits (see the QuickDraw chapter).

Each scan line of the screen displays the contents of 32 consecutive words of memory, each word controlling 16 horizontally adjacent pixels. In each word, the high-order bit (bit 15) controls the leftmost pixel and the low-order bit (bit 0) controls the rightmost pixel. The first word in each scan line follows the last word on the line above it. The starting address of the screen is thus in the top left corner, and the addresses progress from there to the right and down, to the last byte in the extreme bottom right corner.

Normally, the video display doesn't flicker when you read from or write to it, because the video memory accesses are interleaved with the processor accesses. But if you're creating an animated image by repeatedly drawing the graphics in quick succession, it may appear to flicker if the electron beam displays it when your program hasn't finished updating it, showing some of the new image and some of the old in the same frame.

One way to prevent flickering when you're updating the screen continuously is to use the vertical and horizontal blanking signals to synchronize your updates to the scanning of video memory. Small changes to your screen can be completed entirely during the interval between frames (the first 1.26 msec following a vertical blanking interrupt), when nothing is being displayed on the screen. When making larger changes, the trick is to keep your changes happening always ahead of the spot being displayed by the electron beam, as it scans byte by byte through the video memory. Changes you make in the memory already passed over by the scan spot won't appear until the next frame. If you start changing your image when the vertical blanking interrupt occurs, you have 1.26 msec of unrestricted access to the image. After that, you can change progressively less and less of your image as it's scanned onto the screen, starting from the top (the lowest video memory address). From vertical blanking interrupt, you have only 1.26 msec in which to change the first (lowest address) screen location, but you have almost 16.6 msec to change the last (highest address) screen location.

Another way to create smooth, flicker-free graphics, especially useful with changes that may take more 16.6 msec, is to use the two screen buffers as alternate displays. If you draw into the one that's currently not being displayed, and then switch the buffers during the



8 Macintosh Hardware

next vertical blanking, your graphics will change all at once, producing a clean animation. (See the Vertical Retrace Manager chapter to find out how to specify tasks to be performed during vertical blanking.)

If you want to use the alternate screen buffer, you'll have to specify this to the Segment Loader (see the Segment Loader chapter for details). To switch to the alternate screen buffer, clear the following bit of VIA data register A (vBase+vBufA):

```
vPage2    .EQU    6    ;0 = alternate screen buffer
```

For example:

```
BCLR    #vPage2,vBase+vBufA
```

To switch back to the main buffer, set the same bit.

(warning)

Whenever you change a bit in a VIA data register, be sure to leave the other bits in the register unchanged.

(warning)

The alternate screen buffer may not be supported in future versions of the Macintosh.

---

THE SOUND GENERATOR

---

The Macintosh sound circuitry uses a series of values taken from an area of RAM to create a changing waveform in the output signal. This signal drives a small speaker inside the Macintosh and is connected to the external sound jack on the back of the computer. If a plug is inserted into the external sound jack, the internal speaker is disabled. The external sound line can drive a load of 600 or more ohms, such as the input of almost any audio amplifier, but not a directly connected external speaker.

The sound generator may be turned on or off by writing 1 (off) or 0 (on) to the following bit of VIA data register B (vBase+vBufB):

```
vSndEnb    .EQU    7    ;0 = sound enabled, 1 = disabled
```

For example:

```
BSET    #vSndEnb,vBase+vBufB    ;turn off sound
```

By storing a range of values in the sound buffer, you can create the corresponding waveform in the sound channel. The sound generator uses a form of pulse-width encoding to create sounds. The sound circuitry reads one word in the sound buffer during each horizontal blanking interval (including the "virtual" intervals during vertical blanking) and uses the high-order byte of the word to generate a pulse of

electricity whose duration (width) is proportional to the value in the byte. Another circuit converts this pulse into a voltage that's attenuated (reduced) by a three-bit value from the VIA. This reduction corresponds to the current setting of the volume level. To set the volume directly, store a three-bit number in the low-order bits of VIA data register A (vBase+vBufA). You can use the following constant to isolate the bits involved:

```
vSound .EQU 7 ;sound volume bits
```

Here's an example of how to set the sound level:

```
MOVE.B vBase+vBufA,D0 ;get current value of register A
ANDI.B #255-vSound,D0 ;clear the sound bits
ORI.B #3,D0 ;set medium sound level
MOVE.B D0,vBase+vBufA ;put the data back
```

After attenuation, the sound signal is passed to the audio output line.

The sound circuitry scans the sound buffer at a fixed rate of 370 words per video frame, repeating the full cycle 60.15 times per second. To create sounds with frequencies other than multiples of the basic scan rate, you must store phase-shifted patterns into the sound buffer between each scan. You can use the vertical and horizontal blanking signals (available in the VIA) to synchronize your sound buffer updates to the buffer scan. You may find that it's much easier to use the routines in the Sound Driver to do these functions.

(warning)

The low-order byte of each word in the sound buffer is used to control the speed of the motor in the disk drive. Don't store any information there, or you'll interfere with the disk I/O.

There are two sound buffers, just as there are two screen buffers. The address of the main sound buffer is stored in the global variable SoundBase and is also available as the constant soundLow. The main sound buffer is at \$1FD00 in a 128K Macintosh, and the alternate buffer is at \$1A100; for a 512K Macintosh, add \$60000 to these values. Each sound buffer contains 370 words of data. As when you want to use the alternate screen buffer, you'll have to specify to the Segment Loader that you want the alternate buffer (see the Segment Loader chapter for details). To select the alternate sound buffer for output, clear the following bit of VIA data register A (vBase+vBufA):

```
vSndPg2 .EQU 3 ;0 = alternate sound buffer
```

To return to the main buffer, set the same bit.

(warning)

Be sure to switch back to the main sound buffer before doing a disk access, or the disk won't work properly.

(warning)

The alternate sound buffer may not be supported in future versions of the Macintosh.

There's another way to generate a simple, square-wave tone of any frequency, using almost no processor intervention. To do this, first load a constant value into all 370 sound buffer locations (use \$00's for minimum volume, \$FF's for maximum volume). Next, load a value into the VIA's timer 1 latches, and set the high-order two bits of the VIA's auxiliary control register (vBase+vACR) for "square wave output" from timer 1. The timer will then count down from the latched value at 1.2766 usec/count, over and over, inverting the vSndEnb bit of VIA register B (vBase+vBufB) after each count down. This takes the constant voltage being generated from the sound buffer and turns it on and off, creating a square-wave sound whose period is

$$2 * 1.2766 \text{ usec} * \text{timer 1's latched value}$$

(note)

You may want to disable timer 1 interrupts during this process (bit 6 in the VIA's interrupt enable register, which is at vBase+vIER).

To stop the square-wave sound, reset the high-order two bits of the auxiliary control register.

(note)

See the SY6522 technical specifications for details of the VIA registers. See also "Sound Driver Hardware" in the Sound Driver chapter.

#### Diagram

---

Figure 2 shows a block diagram for the sound port.

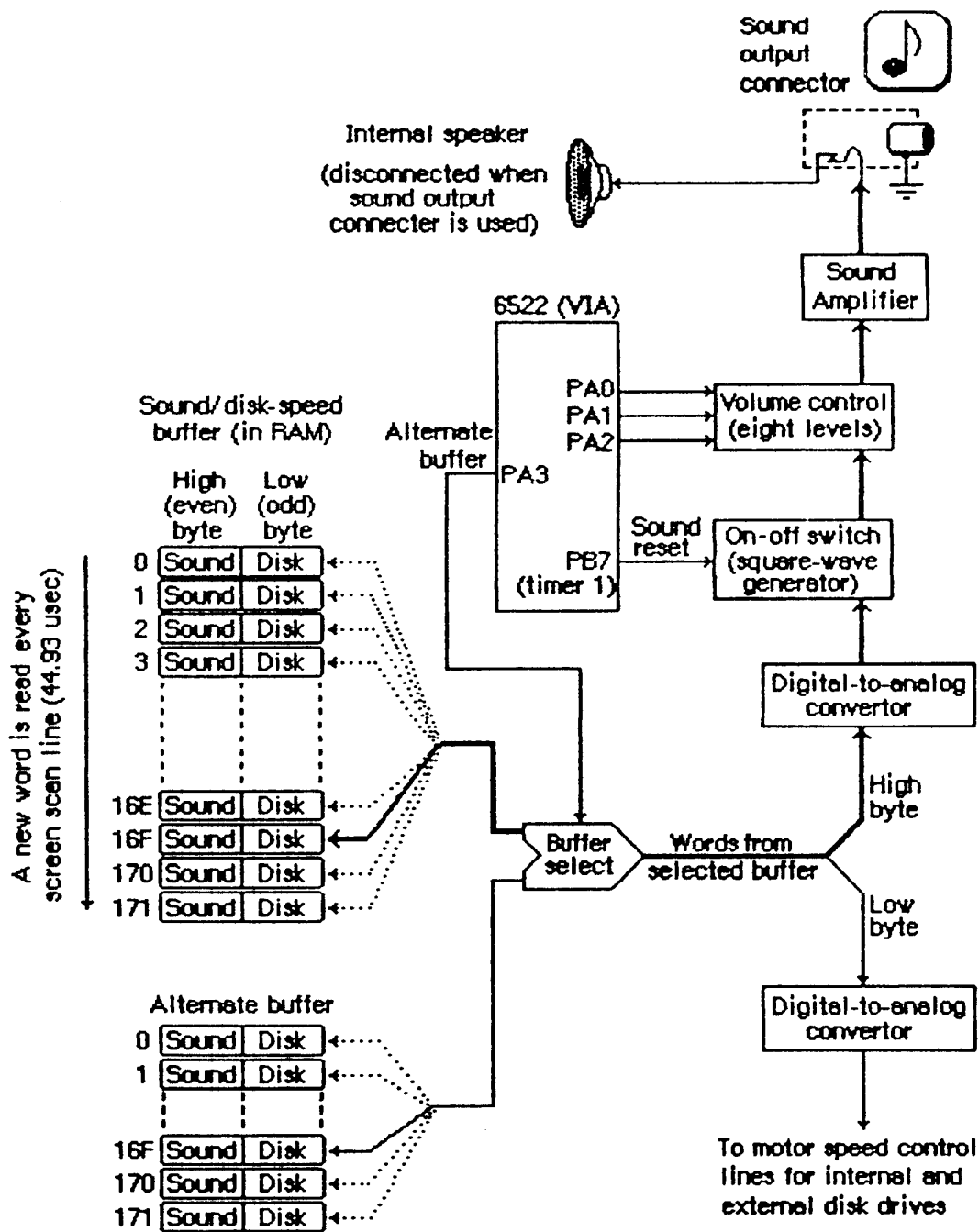


Figure 2. Diagram of Sound Port

---

THE SCC

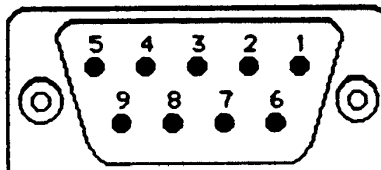
---

The two serial ports are controlled by a Zilog Z8530 Serial Communications Controller (SCC). The port known as SCC port A is the one with the modem icon on the back of the Macintosh. SCC port B is the one with the printer icon.

Macintosh serial ports conform to the EIA standard RS422, which differs from the more common RS232C standard. While RS232C modulates a signal with respect to a common ground ("single-ended" transmission), RS422 modulates two signals against each other ("differential" transmission). The RS232C receiver senses whether the received signal is sufficiently negative with respect to ground to be a logic "1", whereas the RS422 receiver simply senses which line is more negative than the other. This makes RS422 more immune to noise and interference, and more versatile over longer distances. If you ground the positive side of each RS422 receiver and leave unconnected the positive side of each transmitter, you've converted to EIA standard RS423, which can be used to communicate with most RS232C devices over distances up to fifty feet or so.

The serial inputs and outputs of the SCC are connected to the ports through differential line drivers (26LS30) and receivers (26LS32). The line drivers can be tri-stated between transmissions, to allow other devices to transmit over those lines. A driver is activated by the lowering the SCC's Ready To Send (RTS) output for that port. Port A and port B are identical except that port A (the modem port) has a higher interrupt priority, making it more suitable for high-speed communication.

Figure 3 shows the DB-9 pinout for the SCC output jacks.



- 1 Ground
- 2 +5 volts
- 3 Ground
- 4 Transmit data +
- 5 Transmit data -
- 6 +12 volts
- 7 Handshake/external clock
- 8 Receive data +
- 9 Receive data -

Figure 3. Pinout for SCC Output Jacks

(warning)

Do not draw more than 1000 milliamps at +12 volts, and 2000 milliamps at +5 volts from all connectors combined.

Each port's input-only handshake line (pin 7) is connected to the SCC's Clear To Send (CTS) input for that port, and is designed to accept an external device's Data Terminal Ready (DTR) handshake signal. This line is also connected to the SCC's external synchronous clock (TRxC) input for that port, so that an external device can perform high-speed synchronous data exchange. Note that you can't use the line for receiving DTR if you're using it to receive a high-speed data clock.

The handshake line is sensed by the Macintosh using the positive (noninverting) input of one of the standard RS422 receivers (26LS32 chip), with the negative input grounded. The positive input was chosen because this configuration is more immune to noise when no active device is connected to pin 7.

(note)

Because this is a differential receiver, any handshake or clock signal driving it must be "bi-polar", alternating between a positive voltage and a negative voltage, with respect to the internally grounded negative input. If a device tries to use ground (0 volts) as one of its handshake logic levels, the Macintosh will receive that level as an indeterminate state, with unpredictable results.

The SCC itself (at its PCLK pin) is clocked at 3.672 megahertz. The internal synchronous clock (RTxC) pins for both ports are also connected to this 3.672 MHz clock. This is the clock that, after dividing by 16, is normally fed to the SCC's internal baud-rate generator.

The SCC chip generates level-1 processor interrupts during I/O over the serial lines. For more information about SCC interrupts, see the Device Manager chapter.

The locations of the SCC control and data lines are given in the following table as offsets from the constant `sccWBase` for writes, or `sccRBase` for reads. These base addresses are also available in the global variables `SCCW` and `SCCR`. The SCC is on the upper byte of the data bus, so you must use only even-addressed byte reads (a byte read of an odd SCC read address tries to reset the entire SCC). When writing, however, you must use only odd-addressed byte writes (the MC68000 puts your data on both bytes of the bus, so it works correctly). A word access to any SCC address will shift the phase of the computer's high-frequency timing by 128 nanoseconds (system software adjusts it correctly during the system startup process).

## 14 Macintosh Hardware

<u>Location</u>	<u>Contents</u>
sccWBase+aData	Write data register A
sccRBase+aData	Read data register A
sccWBase+bData	Write data register B
sccRBase+bData	Read data register B
sccWBase+aCtl	Write control register A
sccRBase+aCtl	Read control register A
sccWBase+bCtl	Write control register B
sccRBase+bCtl	Read control register B

(warning)

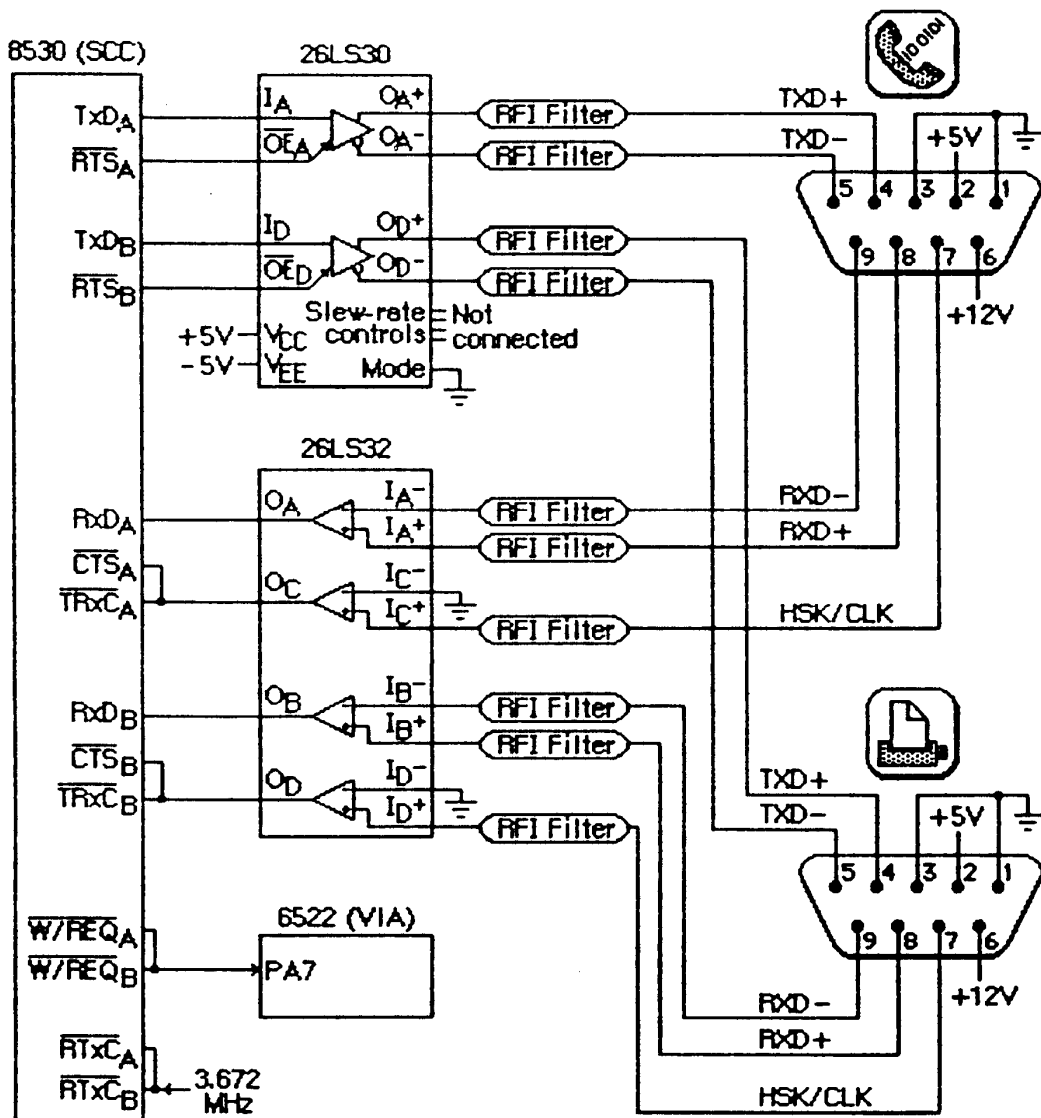
Don't access the SCC chip more often than once every 2.2 usec. The SCC requires that much time to let its internal lines stabilize.


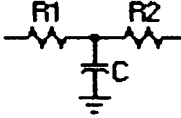
Refer to the technical specifications of the Zilog Z8530 for the detailed bit maps and control methods (baud rates, protocols, and so on) of the SCC.

Diagram

---

Figure 4 shows a circuit diagram for the serial ports.



Note:  = 

$R1 + R2 = 40 \text{ to } 60 \text{ ohms}$   
 $C = 150 \text{ to } 300 \text{ pF}$

Figure 4. Diagram of Serial Ports



---

**THE MOUSE**

---

The DB-9 connector labeled with the mouse icon connects to the Apple mouse (Apple II, Apple III, Lisa, and Macintosh mice are electrically identical). The mouse generates four square-wave signals that describe the amount and direction of the mouse's travel. Interrupt-driven routines in the Macintosh ROM convert this information into the corresponding motion of the pointer on the screen. By turning an option called mouse scaling on or off in the Control Panel desk accessory, the user can change the amount of screen pointer motion that corresponds to a given mouse motion, depending on how fast the mouse is moved; for more information about mouse scaling, see the discussion of parameter RAM in the Operating System Utilities chapter.

(note)

The mouse is a relative-motion device; that is, it doesn't report where it is, only how far and in which direction it's moving. So if you want to connect graphics tablets, touch screens, light pens, or other absolute-position devices to the mouse port, you must either convert their coordinates into motion information or install your own device-handling routines.

The mouse operates by sending square-wave trains of information to the Macintosh that change as the velocity and direction of motion change. The rubber-coated steel ball in the mouse contacts two capstans, each connected to an interrupter wheel: Motion along the mouse's X axis rotates one of the wheels and motion along the Y axis rotates the other wheel.

The Macintosh uses a scheme known as quadrature to detect which direction the mouse is moving along each axis. There's a row of slots on an interrupter wheel, and two beams of infrared light shine through the slots, each one aimed at a phototransistor detector. The detectors are offset just enough so that, as the wheel turns, they produce two square-wave signals (called the interrupt signal and the quadrature signal) 90 degrees out of phase. The quadrature signal precedes the interrupt signal by 90 degrees when the wheel turns one way, and trails it when the wheel turns the other way.

The interrupt signals, X1 and Y1, are connected to the SCC's DCDA and DCDB inputs, respectively, while the quadrature signals, X2 and Y2, go to inputs of the VIA's data register B. When the Macintosh is interrupted (from the SCC) by the rising edge of a mouse interrupt signal, it checks the VIA for the state of the quadrature signal for that axis: If it's low, the mouse is moving to the left (or down), and if it's high, the mouse is moving to the right (or up). When the SCC interrupts on the falling edge, a high quadrature level indicates motion to the left (or down) and a low quadrature level indicates motion to the right (or up):

<u>SCC</u> Mouse interrupt X1 (or Y1)	<u>VIA</u> Mouse quadrature X2 (or Y2)	<u>Mouse</u> Motion direction in X (or Y) axis
Positive edge	Low High	Left (or down) Right (or up)
Negative edge	Low High	Right (or up) Left (or down)

Figure 5 shows the interrupt (Y1) and quadrature (Y2) signals when the mouse is moved downwards.

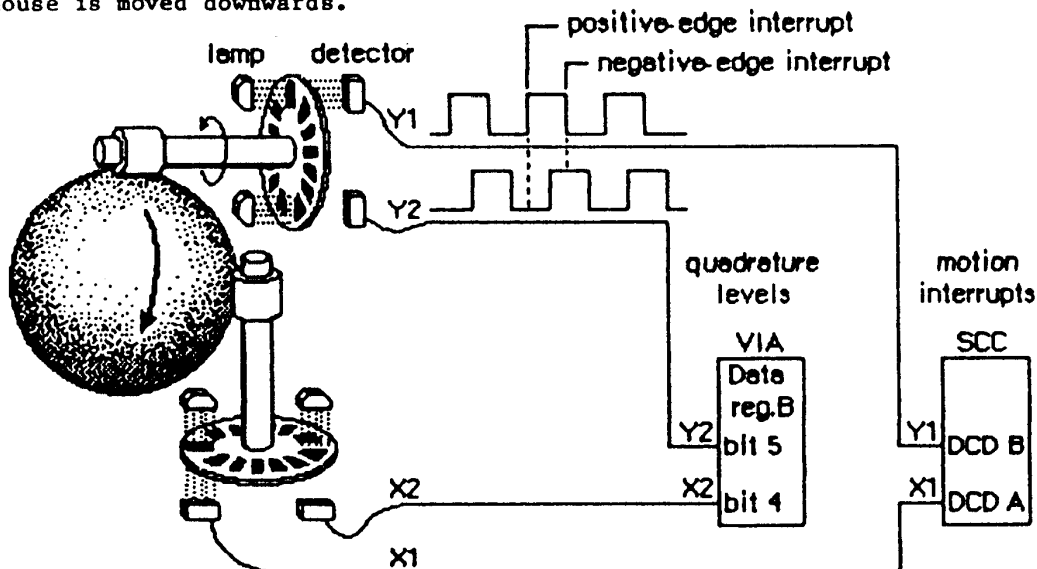


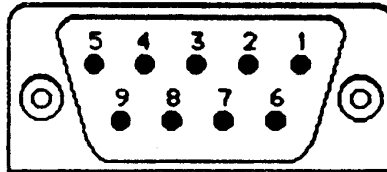
Figure 5. Mouse Mechanism

The switch on the mouse is a pushbutton that grounds pin 7 on the mouse connector when pressed. The state of the button is checked by software during each vertical blanking interrupt. The small delay between each check is sufficient to debounce the button. You can look directly at the mouse button's state by examining the following bit of VIA data register B ( $vBase+vBufB$ ):

```
vSW .EQU 3 ;0 = mouse button is down
```

If the bit is clear, the mouse button is down. However, it's recommended that you let the Operating System handle this for you through the event mechanism.

Figure 6 shows the DB-9 pinout for the mouse jack at the back of the Macintosh.



- 1 Ground
- 2 +5 volts
- 3 Ground
- 4 Mouse X2 (VIA quadrature signal)
- 5 Mouse X1 (SCC interrupt signal)
- 6 (not connected)
- 7 Mouse switch
- 8 Mouse Y2 (VIA quadrature signal)
- 9 Mouse Y1 (SCC interrupt signal)

Figure 6. Pinout for Mouse Jack

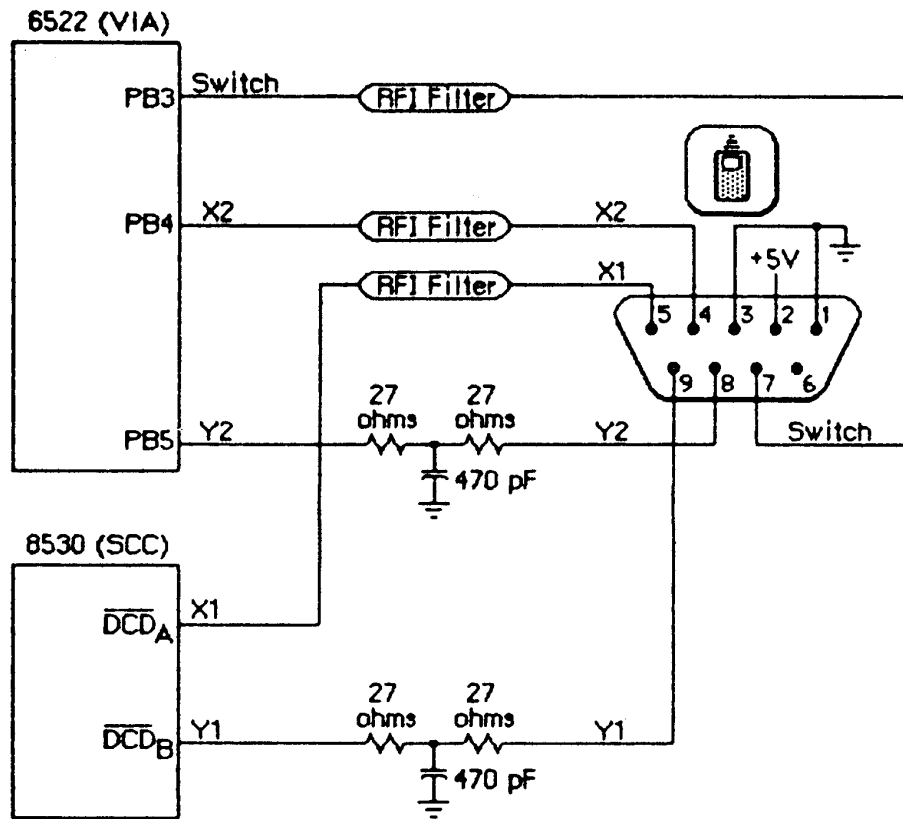
(warning)


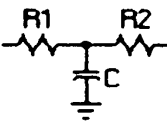
Do not draw more than 200 milliamps at +5 volts from all connectors combined.

Diagram

---

Figure 7 shows a circuit diagram for the mouse port.



Note:  = 

$R1 + R2 = 40 \text{ to } 60 \text{ ohms}$   
 $C = 150 \text{ to } 300 \text{ pF}$

Figure 7. Diagram of Mouse Port

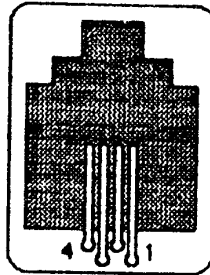
**THE KEYBOARD AND KEYPAD**

The Macintosh keyboard and numeric keypad each contain an Intel 8021 microprocessor that scans the keys. The 8021 contains ROM and RAM, and is programmed to conform to the interface protocol described below.

The keyboard plugs into the Macintosh through a four-wire RJ-11 telephone-style jack. If a numeric keypad is installed in the system,

## 20 Macintosh Hardware

the keyboard plugs into it and it in turn plugs into the Macintosh. Figure 8 shows the pinout for the keyboard jack on the Macintosh, on the keyboard itself, and on the numeric keypad.



- 1 Ground
- 2 Clock
- 3 Data
- 4 +5 volts

Figure 8. Pinout for Keyboard Jack

(warning)

Do not draw more than 200 milliamps at +5 volts from all connectors combined.

#### Keyboard Communication Protocol

The keyboard data line is bidirectional and is driven by whatever device is sending data. The keyboard clock line is driven by the keyboard only. All data transfers are synchronous with the keyboard clock. Each transmission consists of eight bits, with the highest-order bits first.

When sending data to the Macintosh, the keyboard clock transmits eight 330-usec cycles (160 usec low, 170 usec high) on the normally high clock line. It places the data bit on the data line 40 usec before the falling edge of the clock line and maintains it for 330 usec. The data bit is clocked into the Macintosh's VIA shift register on the rising edge of the keyboard clock cycle.

When the Macintosh sends data to the keyboard, the keyboard clock transmits eight 400-usec cycles (180 usec low, 220 usec high) on the clock line. On the falling edge of the keyboard clock cycle, the Macintosh places the data bit on the data line and holds it there for 400 usec. The keyboard reads the data bit 80 usec after the rising edge of the keyboard clock cycle.

Only the Macintosh can initiate communication over the keyboard lines. On power-up of either the Macintosh or the keyboard, the Macintosh is in charge, and the external device is passive. The Macintosh signals that it's ready to begin communication by pulling the keyboard data line low. Upon detecting this, the keyboard starts clocking and the

Macintosh sends a command. The last bit of the command leaves the keyboard data line low; the Macintosh then indicates it's ready to receive the keyboard's response by setting the data line high.

The first command the Macintosh sends out is the Model Number command. The keyboard's response to this command is to reset itself and send back its model number to the Macintosh. If no response is received for 1/2 second, the Macintosh tries the Model Number command again. Once the Macintosh has successfully received a model number from the keyboard, normal operation can begin. The Macintosh sends the Inquiry command; the keyboard sends back a Key Transition response if a key has been pressed or released. If no key transition has occurred after 1/4 second, the keyboard sends back a Null response to let the Macintosh know it's still there. The Macintosh then sends the Inquiry command again. In normal operation, the Macintosh sends out an Inquiry command every 1/4 second. If it receives no response within 1/2 second, it assumes the keyboard is missing or needs resetting, so it begins again with the Model Number command.

There are two other commands the Macintosh can send: the Instant command, which gets an instant keyboard status without the 1/4-second timeout, and the Test command, to perform a keyboard self-test. Here's a list of the commands that can be sent from the Macintosh to the keyboard:

<u>Command name</u>	<u>Value</u>	<u>Keyboard response</u>
Inquiry	\$10	Key Transition code or Null (\$7B)
Instant	\$14	Key Transition code or Null (\$7B)
Model Number	\$16	Bit 0: 1 Bits 1-3: keyboard model number, 1-8 Bits 4-6: next device number, 1-8 Bit 7: 1 if another device connected
Test	\$36	ACK (\$7D) or NAK (\$77)

The Key Transition responses are sent out by the keyboard as a single byte: Bit 7 high means a key-up transition, and bit 7 low means a key-down. Bit 0 is always high. The Key Transition responses for key-down transitions on the keyboard are shown (in hexadecimal) in Figure 9. Note that these response codes are different from the key codes returned by the keyboard driver software. The keyboard driver strips off bit 7 of the response and shifts the result one bit to the right, removing bit 0. For example, response code \$33 becomes \$19, and \$2B becomes \$15.

`	1	2	3	4	5	6	7	8	9	0	-	=	Backspace
65	25	27	29	2B	2F	2D	35	39	33	3B	37	31	67
Tab	Q	W	E	R	T	Y	U	I	O	P	[	]	\
61	19	1B	1D	1F	23	13	41	45	3F	47	43	3D	55
Caps Lock	A	S	D	F	G	H	J	K	L	;	'	Return	
73	01	03	05	07	0B	09	4D	51	4B	53	4F	49	
Shift	Z	X	C	V	B	N	M	,	.	/		Shift	
71	0D	0F	11	13	17	5B	5D	57	5F	59		71	
Option	⌘											Enter	Option
75	6F											69	75
							space						
							63						

U.S. keyboard

§	1	2	3	4	5	6	7	8	9	0	-	=	←
65	25	27	29	2B	2F	2D	35	39	33	3B	37	31	67
→	Q	W	E	R	T	Y	U	I	O	P	[	]	↻
61	19	1B	1D	1F	23	21	41	45	3F	47	43	3D	
◊	A	S	D	F	G	H	J	K	L	;	'	`	55
73	01	03	05	07	0B	09	4D	51	4B	53	4F	49	
◊	\	Z	X	C	V	B	N	M	,	.	/	◊	71
71	0D	0F	11	13	17	5B	5D	57	5F	59	15		
⌘	⌘											⌘	⌘
75	6F						space					63	75

International keyboard (Great Britain key caps shown)

Clear	-	⌘	⌘
0F	1D	0D	05
7	8	9	⌘
33	37	39	1B
4	5	6	⌘
2D	2F	31	11
1	2	3	Enter
27	29	2B	
0	.		
25	03	19	

Keypad (U.S. key caps shown)

Figure 9. Key-Down Transitions

### Keypad Communication Protocol

When a numeric keypad is used, it must be inserted between the keyboard and the Macintosh; that is, the keypad cable plugs into the jack on the front of the Macintosh, and the keyboard cable plugs into a jack on the numeric keypad. In this configuration, the timings and protocol for the clock and data lines work a little differently: The keypad acts like a keyboard when communicating with the Macintosh, and acts like a Macintosh when communicating over the separate clock and data lines going to the keyboard. All commands from the Macintosh are now received by the keypad instead of the keyboard, and only the keypad can communicate directly with the keyboard.

When the Macintosh sends out an Inquiry command, one of two things may happen, depending on the state of the keypad. If no key transitions have occurred on the keypad since the last Inquiry, the keypad sends an Inquiry command to the keyboard and, later, retransmits the keyboard's response back to the Macintosh. But if a key transition has occurred on the keypad, the keypad responds to an Inquiry by sending back the Keypad response (\$79) to the Macintosh. In that case, the Macintosh immediately sends an Instant command, and this time the keypad sends back its own Key Transition response. As with the keyboard, bit 7 high means key-up and bit 7 low means key-down.

The Key Transition responses for key-down transitions on the keypad are shown in Figure 9 above. Again, note that these response codes are different from the key codes returned by the keyboard driver software. The keyboard driver strips off bit 7 of the response and shifts the result one bit to the right, removing bit 0.

### THE DISK INTERFACE

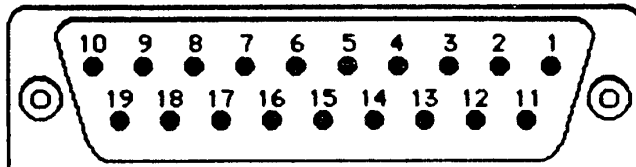
The Macintosh disk interface uses a design similar to that used on the Apple II and Apple III computers, employing the Apple custom IWM chip. Another custom chip called the Analog Signal Generator (ASG) reads the disk speed buffer in RAM and generates voltages that control the disk speed. Together with the VIA, the IWM and the ASG generate all the signals necessary to read, write, format, and eject the 3 1/2-inch disks used by the Macintosh.

The IWM controls four of the disk state-control lines (called CA0, CA1, CA2, and LSTRB), chooses which drive (internal or external) to enable, and processes the disk's read-data and write-data signals. The VIA provides another disk state-control line called SEL.

A buffer in RAM (actually the low-order bytes of words in the sound buffer) is read by the ASG to generate a pulse-width modulated signal that's used to control the speed of the disk motor. The Macintosh Operating System uses this speed control to allow it to store more sectors of information in the tracks closer to the edge of the disk by running the disk motor at slower speeds.



Figure 10 shows the DB-19 pinout for the external disk jack at the back of the Macintosh.



1	Ground	11	CA0
2	Ground	12	CA1
3	Ground	13	CA2
4	Ground	14	LSTRB
5	-12 volts	15	Write request
6	+5 volts	16	SEL
7	+12 volts	17	External drive enable
8	+12 volts	18	Read data
9	(not connected)	19	Write data
10	Motor speed control		

Figure 10. Pinout for Disk Jack

(warning)

This connector was designed for a Macintosh 3 1/2-inch disk drive, which represents a load of 500 milliamps at +12 volts, 500 milliamps at +5 volts, and 0 milliamps at -12 volts. If any other device uses this connector, it must not exceed these loads by more than 100 milliamps at +12 volts, 200 milliamps at +5 volts, and 10 milliamps at -12 volts, including loads from all other connectors combined.

Controlling the Disk State-Control Lines

The IWM contains registers that can be used by the software to control the state-control lines leading out to the disk. By reading or writing certain memory locations, you can turn these state-control lines on or off. Other locations set various IWM internal states. The locations are given in the following table as offsets from the constant dBase, the base address of the IWM; this base address is also available in a global variable named IWM. The IWM is on the lower byte of the data bus, so use odd-addressed byte accesses only.

<u>IWM line</u>	<u>Location to turn line on</u>	<u>Location to turn line off</u>
Disk state-control lines:		
CA0	dBase+ph0H	dBase+ph0L
CA1	dBase+ph1H	dBase+ph1L
CA2	dBase+ph2H	dBase+ph2L
LSTRB	dBase+ph3H	dBase+ph3L
Disk enable line:		
ENABLE	dBase+motorOn	dBase+motorOff
IWM internal states:		
SELECT	dBase+extDrive	dBase+intDrive
Q6	dBase+q6H	dBase+q6L
Q7	dBase+q7H	dBase+q7L

To turn one of the lines on or off, do any kind of memory byte access (read or write) to the respective location.

The CA0, CA1, and CA2 lines are used along with the SEL line from the VIA to select from among the registers and data signals in the disk drive. The LSTRB line is used when writing control information to the disk registers (as described below), and the ENABLE line enables the selected disk drive. SELECT is an IWM internal line that chooses which disk drive can be enabled: On selects the external drive, and off selects the internal drive. The Q6 and Q7 lines are used to set up the internal state of the IWM for reading disk register information, as well as for reading or writing actual disk-storage data.

You can read information from several registers in the disk drive to find out whether the disk is locked, whether a disk is in the drive, whether the head is at track 0, how many heads the drive has, and whether there's a drive connected at all. In turn, you can write to some of these registers to step the head, turn the motor on or off, and eject the disk.

#### Reading from the Disk Registers

Before you can read from any of the disk registers, you must set up the state of the IWM so that it can pass the data through to the MC68000's memory space where you'll be able to read it. To do that, you must first turn off Q7 by reading or writing dBase+q7L. Then turn on Q6 by accessing dBase+q6H. After that, the IWM will be able to pass data from the disk's RD/SENSE line through to you.

Once you've set up the IWM for disk register access, you must next select which register you want to read. To read one of the disk registers, first enable the drive you want to use (by accessing dBase+intDrive or dBase+extDrive and then dBase+motorOn) and make sure LSTRB is low. Then set CA0, CA1, CA2, and SEL to address the register you want. Once this is done, you can read the disk register data bit in the high-order bit of dBase+q7L. After you've read the data, you

may read another disk register by again setting the proper values in CA0, CA1, CA2, and SEL, and then reading dBase+q7L.

(warning)

When you're finished reading data from the disk registers, it's important to leave the IWM in a state that the Disk Driver will recognize. To be sure it's in a valid logic state, always turn Q6 back off (by accessing dBase+q6L) after you've finished reading the disk registers.

The following table shows how you must set the disk state-control lines to read from the various disk registers and data signals:

State-control lines				Register	Information in register
CA2	CA1	CA0	SEL	addressed	
0	0	0	0	DIRTN	Head step direction
0	0	0	1	CSTIN	Disk in place
0	0	1	0	STEP	Disk head stepping
0	0	1	1	WRTPRT	Disk locked
0	1	0	0	MOTORON	Disk motor running
0	1	0	1	TKO	Head at track 0
0	1	1	1	TACH	Tachometer
1	0	0	0	RDDATA0	Read data, lower head
1	0	0	1	RDDATA1	Read data, upper head
1	1	0	0	SIDES	Single- or double-sided drive
1	1	1	1	DRVIN	Drive installed

Writing to the Disk Registers

To write to a disk register, first be sure that LSTRB is off, then turn on CA0 and CA1. Next, set SEL to 0. Set CA0 and CA1 to the proper values from the table below, then set CA2 to the value you want to write to the disk register. Hold LSTRB high for at least one usec but not more than one msec (unless you're ejecting a disk) and bring it low again. Be sure that you don't change CA0-CA2 or SEL while LSTRB is high, and that CA0 and CA1 are set high before changing SEL.

The following table shows how you must set the disk state-control lines to write to the various disk registers:

Control lines			Register	Register function
CA1	CA0	SEL	addressed	
0	0	0	DIRTN	Set stepping direction
0	1	0	STEP	Step disk head one track
1	0	0	MOTORON	Turn on/off disk motor
1	1	0	EJECT	Eject the disk

---

Explanations of the Disk Registers

---

The information written to or read from the various disk registers can be interpreted as follows:

- The DIRTN signal sets the direction of subsequent head stepping: 0 causes steps to go toward the inside track (track 79), 1 causes them to go toward the outside track (track 0).
- CSTIN is 0 only when a disk is in the drive.
- Setting STEP to 0 steps the head one full track in the direction last set by DIRTN. When the step is complete (about 12 msec), the disk drive sets STEP back to 1, and then you can step again.
- WRTPRT is 0 whenever the disk is locked. Do not write to a disk unless WRTPRT is 1.
- MOTORON controls the state of the disk motor: 0 turns on the motor, and 1 turns it off. The motor will run only if the drive is enabled and a disk is in place; otherwise, writing to this line will have no effect.
- TKO goes to 0 only if the head is at track 0. This is valid beginning 12 msec after the step that puts it at track 0.
- Writing 1 to EJECT ejects the disk from the drive. To eject a disk, you must hold LSTRB high for at least 1/2 second.
- The current disk speed is available as a pulse train on TACH. The TACH line produces 60 pulses for each rotation of the drive motor. The disk motor speed is controlled by the ASG as it reads the disk speed RAM buffer.
- RDDATA0 and RDDATA1 carry the instantaneous data from the disk head.
- SIDES is always 0 on single-sided drives and 1 on double-sided drives.
- DRVIN is always 0 if the selected disk drive is physically connected to the Macintosh, otherwise it floats to 1.

---

THE REAL-TIME CLOCK

---

The Macintosh real-time clock is a custom chip whose interface lines are available through the VIA. The clock contains a four-byte counter that's incremented once each second, as well as a line that can be used by the VIA to generate an interrupt once each second. It also contains 20 bytes of RAM that are powered by a battery when the Macintosh is turned off. These RAM bytes, called parameter RAM, contain important

data that needs to be preserved even when the system power is not available. The Operating System maintains a copy of parameter RAM that you can access in low memory. To find out how to use the values in parameter RAM, see the Operating System Utilities chapter.

#### Accessing The Clock Chip

---

The clock is accessed through the following bits of VIA data register B (vBase+vBufB):

rTCDat	.EQU	0	;real-time clock serial data line
rTCClk	.EQU	1	;real-time clock data-clock line
rTCEnb	.EQU	2	;real-time clock serial enable

These three bits constitute a simple serial interface. The rTCDat bit is a bidirectional serial data line used to send command and data bytes back and forth. The rTCClk bit is a data-clock line, always driven by the processor (you set it high or low yourself) that regulates the transmission of the data and command bits. The rTCEnb bit is the serial enable line, which signals the real-time clock that the processor is about to send it serial commands and data.

To access the clock chip, you must first enable its serial function. To do this, set the serial enable line (rTCEnb) to 0. Keep the serial enable line low during the entire transaction; if you set it to 1, you'll abort the transfer.

(warning)

Be sure you don't alter any of bits 3-7 of VIA data register B during clock serial access.

A command can be either a write request or a read request. After the eight bits of a write request, the clock will expect the next eight bits across the serial data line to be your data for storage into one of the internal registers of the clock. After receiving the eight bits of a read request, the clock will respond by putting eight bits of its data on the serial data line. Commands and data are transferred serially in eight-bit groups over the serial data line, with the high-order bit first and the low-order bit last.

To send a command to the clock, first set the rTCDat bit of VIA data direction register B (vBase+vDirB) so that the real-time clock's serial data line will be used for output to the clock. Next, set the rTCClk bit of vBase+vBufB to 0, then set the rTCDat bit to the value of the first (high-order) bit of your data byte. Then raise (set to 1) the data-clock bit (rTCClk). Then lower the data-clock, set the serial data line to the next bit, and raise the data-clock line again. After the last bit of your command has been sent in this way, you can either continue by sending your data byte in the same way (if your command was a write request) or switch to receiving a data byte from the clock (if your command was a read request).

To receive a byte of data from the clock, you must first send a command that's a read request. After you've clocked out the last bit of the command, clear the rTCData bit of the data direction register so that the real-time clock's serial data line can be used for input from the clock; then lower the data-clock bit (rTCClk) and read the first (high-order) bit of the clock's data byte on the serial data line. Then raise the data-clock, lower it again, and read the next bit of data. Continue this until all eight bits are read, then raise the serial enable line (rTCEnb), disabling the data transfer.

The following table lists the commands you can send to the clock. A 1 in the high-order bit makes your command a read request; a 0 in the high-order bit makes your command a write request. (In this table, "z" is the bit that determines read or write status, and bits marked "a" are bits whose values depend on what parameter RAM byte you want to address.)

<u>Command byte</u>	<u>Register addressed by the command</u>
z0000001	Seconds register 0 (lowest-order byte)
z0000101	Seconds register 1
z0001001	Seconds register 2
z0001101	Seconds register 3 (highest-order byte)
00110001	Test register (write only)
00110101	Write-protect register (write only)
z010aa01	RAM address 100aa (\$10-\$13)
zlaaaa01	RAM address 0aaaa (\$00-\$0F)

Note that the last two bits of a command byte must always be 01.

If the high-order bit (bit 7) of the write-protect register is set, this prevents writing into any other register on the clock chip (including parameter RAM). Clearing the bit allows you to change any values in any registers on the chip. Don't try to read from this register; it's a write-only register.

The two highest-order bits (bits 7 and 6) of the test register are used as device control bits during testing, and should always be set to 0 during normal operation. Setting them to anything else will interfere with normal clock counting. Like the write-protect register, this is a write-only register; don't try to read from it.

All clock data must be sent as full eight-bit bytes, even if only one or two bits are of interest. The rest of the bits may not matter, but you must send them to the clock or the write will be aborted when you raise the serial enable line.

It's important to use the proper sequence if you're writing to the clock's seconds registers. If you write to a given seconds register, there's a chance that the clock may increment the data in the next higher-order register during the write, causing unpredictable results. To avoid this possibility, always write to the registers in low-to-high order. Similarly, the clock data may increment during a read of all four time bytes, which could cause invalid data to be read. To avoid this, always read the time twice (or until you get the same value

twice).

(warning)

When you've finished reading from the clock registers, always end by doing a final write such as setting the write-protect bit. Failure to do this may leave the clock in a state that will run down the battery more quickly than necessary.

### The One-Second Interrupt

---

The clock also generates a VIA interrupt once each second (if this interrupt is enabled). The enable status for this interrupt can be read from or written to bit 0 of the VIA's interrupt enable register (vBase+vIER). When reading the enable register, a 1 bit indicates the interrupt is enabled, and 0 means it's disabled. Writing \$01 to the enable register disables the clock's one-second interrupt (without affecting any other interrupts), while writing \$81 enables it again. See the Device Manager chapter for more information about writing your own interrupt handlers.

(warning)

Be sure when you write to bit 0 of the VIA's interrupt enable register that you don't change any of the other bits.

### THE VIA

---

The Synertek SY6522 Versatile Interface Adapter (VIA) controls the keyboard, internal real-time clock, parts of the disk, sound, and mouse interfaces, and various internal Macintosh signals. Its base address is available as the constant vBase and is also stored in a global variable named VIA. The VIA is on the upper byte of the data bus, so use even-addressed byte accesses only.

There are two parallel data registers within the VIA, called A and B, each with a data direction register. There are also several event timers, a clocked shift register, and an interrupt flag register with an interrupt enable register.

Normally you won't have to touch the direction registers, since the Operating System sets them up for you at system startup. A 1 bit in a data direction register means the corresponding bit of the respective data register will be used for output, while a 0 bit means it will be used for input.

(note)

For more information on the registers and control structure of the VIA, consult the technical specifications for the SY6522 chip.

VIA Register A

VIA data register A is at vBase+vBufA. The corresponding data direction register is at vBase+vDirA.

<u>Bit(s)</u>	<u>Name</u>	<u>Description</u>
7	vSCCWReq	SCC wait/request
6	vPage2	Alternate screen buffer
5	vHeadSel	Disk SEL line
4	vOverlay	ROM low-memory overlay
3	vSndPg2	Alternate sound buffer
0-2	vSound (mask)	Sound volume

The vSCCWReq bit can signal that the SCC has received a character (used to maintain serial communications during disk accesses, when the CPU's interrupts from the SCC are disabled). The vPage2 bit controls which screen buffer is being displayed, and the vHeadSel bit is the SEL control line used by the disk interface. The vOverlay bit (used only during system startup) can be used to place another image of ROM at the bottom of memory, where RAM usually is (RAM moves to \$600000). The sound buffer is selected by the vSndPg2 bit. Finally, the vSound bits control the sound volume.

VIA Register B

VIA data register B is at vBase+vBufB. The corresponding data direction register is at vBase+vDirB.

<u>Bit</u>	<u>Name</u>	<u>Description</u>
7	vSndEnb	Sound enable/disable
6	vH4	Horizontal blanking
5	vY2	Mouse Y2
4	vX2	Mouse X2
3	vSW	Mouse switch
2	rTCEnb	Real-time clock serial enable
1	rTCclk	Real-time clock data-clock line
0	rTCData	Real-time clock serial data

The vSndEnb bit turns the sound generator on or off, and the vH4 bit is set when the video beam is in its horizontal blanking period. The vY2 and vX2 bits read the quadrature signals from the Y (vertical) and X (horizontal) directions, respectively, of the mouse's motion lines. The vSW bit reads the mouse switch. The rTCEnb, rTCclk, and rTCData bits control and read the real-time clock.



The VIA Peripheral Control Register

The VIA's peripheral control register, at vBase+vPCR, allows you to set some very low-level parameters (such as positive-edge or negative-edge triggering) dealing with the keyboard data and clock interrupts, the one-second real-time clock interrupt line, and the vertical blanking interrupt.

<u>Bit(s)</u>	<u>Description</u>
5-7	Keyboard data interrupt control
4	Keyboard clock interrupt control
1-3	One-second interrupt control
0	Vertical blanking interrupt control

The VIA Timers

The timers controlled by the VIA are called timer 1 and timer 2. Timer 1 is used to time various events having to do with the Macintosh sound generator. Timer 2 is used by the Disk Driver to time disk I/O events. If either timer isn't being used by the Operating System, you're free to use it for your own purposes. When a timer counts down to 0, an interrupt will be generated if the proper interrupt enable has been set. See the Device Manager chapter for information about writing your own interrupt handlers.

To start one of the timers, store the appropriate values in the high- and low-order bytes of the timer counter (or the timer 1 latches, for multiple use of the value). The counters and latches are at the following locations:

<u>Location</u>	<u>Contents</u>
vBase+vT1C	Timer 1 counter (low-order byte)
vBase+vT1CH	Timer 1 counter (high-order byte)
vBase+vT1L	Timer 1 latch (low-order byte)
vBase+vT1LH	Timer 1 latch (high-order byte)
vBase+vT2C	Timer 2 counter (low-order byte)
vBase+vT2CH	Timer 2 counter (high-order byte)

(note)

When setting a timer, it's not enough to simply store a full word to the high-order address, because the high- and low-order bytes of the counters are not adjacent. You must explicitly do two stores, one for the high-order byte and one for the low-order byte.

VIA Interrupts

The VIA (through its IRQ line) can cause a level-0 processor interrupt whenever one of the following occurs: Timer 1 or timer 2 times out; the keyboard is clocking a bit in through its serial port; the shift register for the keyboard serial interface has finished shifting in or out; the vertical blanking interval is beginning; or the one-second clock has ticked. For more information on how to use these interrupts, see the Device Manager chapter.

The interrupt flag register at vBase+vIFR contains flag bits that are set whenever the interrupt corresponding to that bit has occurred. The Operating System uses these flags to determine which device has caused an interrupt. Bit 7 of the interrupt flag register is not really a flag: It remains set (and the IRQ line to the processor is held low) as long as any enabled VIA interrupt is occurring.

Bit	Interrupting device
7	IRQ (all enabled VIA interrupts)
6	Timer 1
5	Timer 2
4	Keyboard clock
3	Keyboard data bit
2	Keyboard data ready
1	Vertical blanking interrupt
0	One-second interrupt

The interrupt enable register, at vBase+vIER, lets you enable or disable any of these interrupts. If an interrupt is disabled, its bit in the interrupt flag register will continue to be set whenever that interrupt occurs, but it won't affect the IRQ flag, nor will it interrupt the processor.

The bits in the interrupt enable register are arranged just like those in the interrupt flag register, except for bit 7. When you write to the interrupt enable register, bit 7 is "enable/disable": If bit 7 is a 1, each 1 in bits 0-6 enables the corresponding interrupt; if bit 7 is a 0, each 1 in bits 0-6 disables that interrupt. In either case, 0's in bits 0-6 do not change the status of those interrupts. Bit 7 is always read as a 1.

Other VIA Registers

The shift register, at vBase+vSR, contains the eight bits of data that have been shifted in or that will be shifted out over the keyboard data line.

The auxiliary control register, at vBase+vACR, is described in the SY6522 documentation. It controls various parameters having to do with the timers and the shift register.

---

SYSTEM STARTUP

---

When power is first supplied to the Macintosh, a carefully orchestrated sequence of events takes place.

First, the processor is held in a wait state while a series of circuits gets the system ready for operation. The VIA and IWM are initialized, and the mapping of ROM and RAM are altered temporarily by setting the overlay bit in VIA data register A. This places the ROM starting at the normal ROM location \$400000, and a duplicate image of the same ROM starting at address 0 (where RAM normally is), while RAM is placed starting at \$600000. Under this mapping, the Macintosh software executes out of the normal ROM locations above \$400000, but the MC68000 can obtain some critical low-memory vectors from the ROM image it finds at address 0.

Next, a memory test and several other system tests take place. After the system is fully tested and initialized, the software clears the VIA's overlay bit, mapping the system RAM back where it belongs, starting at address 0. Then the disk startup process begins.

First the internal disk is checked: If there's a disk inserted, the system attempts to read it. If no disk is in the internal drive and there's an external drive with an inserted disk, the system will try to read that one. Otherwise, the question-mark disk icon is displayed until a disk is inserted. If the disk startup fails for some reason, the "sad Macintosh" icon is displayed and the Macintosh goes into an endless loop until it's turned off again.

Once a readable disk has been inserted, the first two sectors (containing the system startup blocks) are read in and the normal disk load begins.

---

SUMMARY

---

(warning)

This information applies only to the Macintosh 128K and 512K, not to the Macintosh XL.

---

Constants

---

; VIA base addresses

```
vBase      .EQU    $EFE1FE    ;main base for VIA chip (in variable VIA)
vBufB     .EQU    vBase      ;register B base
vBufA     .EQU    $EFFFFE    ;register A base
vBufM     .EQU    vBufB     ;register containing mouse signals
vVIFR     .EQU    $EFFFBE    ;interrupt flag register
vVIER     .EQU    $EFFFDFE   ;interrupt enable register
```

; Offsets from vBase

```
vBufB     .EQU    512*0      ;register B (zero offset)
vDirB     .EQU    512*2      ;register B direction register
vDirA     .EQU    512*3      ;register A direction register
vT1C      .EQU    512*4      ;timer 1 counter (low-order byte)
vT1CH     .EQU    512*5      ;timer 1 counter (high-order byte)
vT1L      .EQU    512*6      ;timer 1 latch (low-order byte)
vT1LH     .EQU    512*7      ;timer 1 latch (high-order byte)
vT2C      .EQU    512*8      ;timer 2 counter (low-order byte)
vT2CH     .EQU    512*9      ;timer 2 counter (high-order byte)
vSR       .EQU    512*10     ;shift register (keyboard)
vACR      .EQU    512*11     ;auxiliary control register
vPCR      .EQU    512*12     ;peripheral control register
vIFR      .EQU    512*13     ;interrupt flag register
vIER      .EQU    512*14     ;interrupt enable register
vBufA     .EQU    512*15     ;register A
```

; VIA register A constants

```
vAOut     .EQU    $7F        ;direction register A: 1 bits = outputs
vAInit    .EQU    $7B        ;initial value for vBufA (medium volume)
vSound    .EQU    7          ;sound volume bits
```

; VIA register A bit numbers

```
vSndPg2   .EQU    3          ;0 = alternate sound buffer
vOverlay  .EQU    4          ;1 = ROM overlay (system startup only)
vHeadSel  .EQU    5          ;disk SEL control line
vPage2    .EQU    6          ;0 = alternate screen buffer
vSCCWReq  .EQU    7          ;SCC wait/request line
```

2/11/85

/HARDWARE/HDWR.S

## 36 Macintosh Hardware

## ; VIA register B constants

```
vBOut      .EQU    $87      ;direction register B: 1 bits = outputs
vBInit     .EQU    $07      ;initial value for vBufB
```

## ; VIA register B bit numbers

```
rTCData    .EQU    0        ;real-time clock serial data line
rTCclk     .EQU    1        ;real-time clock data-clock line
rTCEnb     .EQU    2        ;real-time clock serial enable
vSW        .EQU    3        ;0 = mouse button is down
vX2        .EQU    4        ;mouse X quadrature level
vY2        .EQU    5        ;mouse Y quadrature level
vH4        .EQU    6        ;1 = horizontal blanking
vSndEnb    .EQU    7        ;0 = sound enabled, 1 = disabled
```

## ; SCC base addresses

```
sccRBase   .EQU    $9FFFF8   ;SCC base read address (in variable SCCRd)
sccWBase   .EQU    $BFFFF9   ;SCC base write address (in variable SCCWr)
```

## ; Offsets from SCC base addresses

```
aData      .EQU    6        ;channel A data in or out
aCtl       .EQU    2        ;channel A control
bData      .EQU    4        ;channel B data in or out
bCtl       .EQU    0        ;channel B control
```

## ; Bit numbers for control register RR0

```
rxBF       .EQU    0        ;1 = SCC receive buffer full
txBE       .EQU    2        ;1 = SCC send buffer empty
```

## ; IWM base address

```
dBase      .EQU    $DFE1FF   ;IWM base address (in variable IWM)
```

## ; Offsets from dBase

```
ph0L      .EQU    512*0     ;CA0 off (0)
ph0H      .EQU    512*1     ;CA0 on (1)
ph1L      .EQU    512*2     ;CA1 off (0)
ph1H      .EQU    512*3     ;CA1 on (1)
ph2L      .EQU    512*4     ;CA2 off (0)
ph2H      .EQU    512*5     ;CA2 on (1)
ph3L      .EQU    512*6     ;LSTRB off (low)
ph3H      .EQU    512*7     ;LSTRB on (high)
mtrOff    .EQU    512*8     ;disk enable off
mtrOn     .EQU    512*9     ;disk enable on
intDrive  .EQU    512*10    ;select internal drive
extDrive  .EQU    512*11    ;select external drive
q6L       .EQU    512*12    ;Q6 off
q6H       .EQU    512*13    ;Q6 on
q7L       .EQU    512*14    ;Q7 off
```

```

q7H      .EQU    512*15      ;Q7 on

; Screen and sound addresses for 512K Macintosh (will also work for
; 128K, since addresses wrap)

screenLow .EQU    $7A700      ;top left corner of main screen buffer
soundLow  .EQU    $7FD00      ;main sound buffer (in variable SoundBase)
pwmBuffer .EQU    $7FD01      ;main disk speed buffer
ovlyRAM   .EQU    $600000     ;RAM start address when overlay is set
ovlyScreen .EQU    $67A700    ;screen start with overlay set
romStart  .EQU    $400000     ;ROM start address (in variable ROMBase)

```

#### Variables

---

```

ROMBase    Base address of ROM
SoundBase  Address of main sound buffer
SCCRd      SCC read base address
SCCWwr     SCC write base address
IWM        IWM base address
VIA        VIA base address

```

#### Exception Vectors

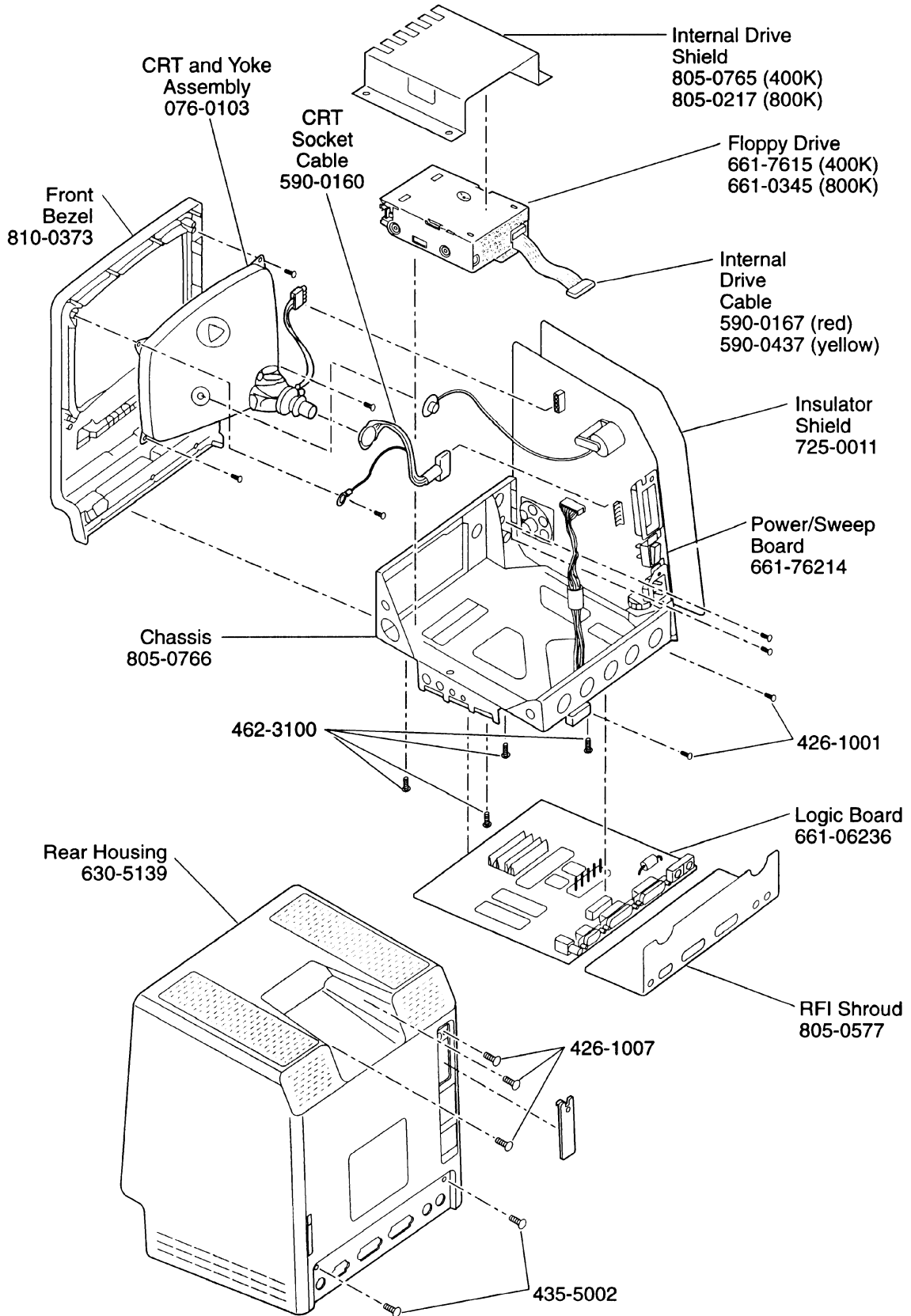
---

Location	Purpose
\$00	Reset: initial stack pointer (not a vector)
\$04	Reset: initial vector
\$08	Bus error
\$0C	Address error
\$10	Illegal instruction
\$14	Divide by zero
\$18	CHK instruction
\$1C	TRAPV instruction
\$20	Privilege violation
\$24	Trace interrupt
\$28	Line 1010 emulator
\$2C	Line 1111 emulator
\$30-\$3B	Unassigned (reserved)
\$3C	Uninitialized interrupt
\$40-\$5F	Unassigned (reserved)
\$60	Spurious interrupt
\$64	VIA interrupt
\$68	SCC interrupt
\$6C	VIA+SCC vector (temporary)
\$70	Interrupt switch
\$74	Interrupt switch + VIA
\$78	Interrupt switch + SCC
\$7C	Interrupt switch + VIA + SCC
\$80-\$BF	TRAP instructions
\$C0-\$FF	Unassigned (reserved)

2/11/85

/HARDWARE/HDWR.S

# Exploded View Macintosh 128K/512K



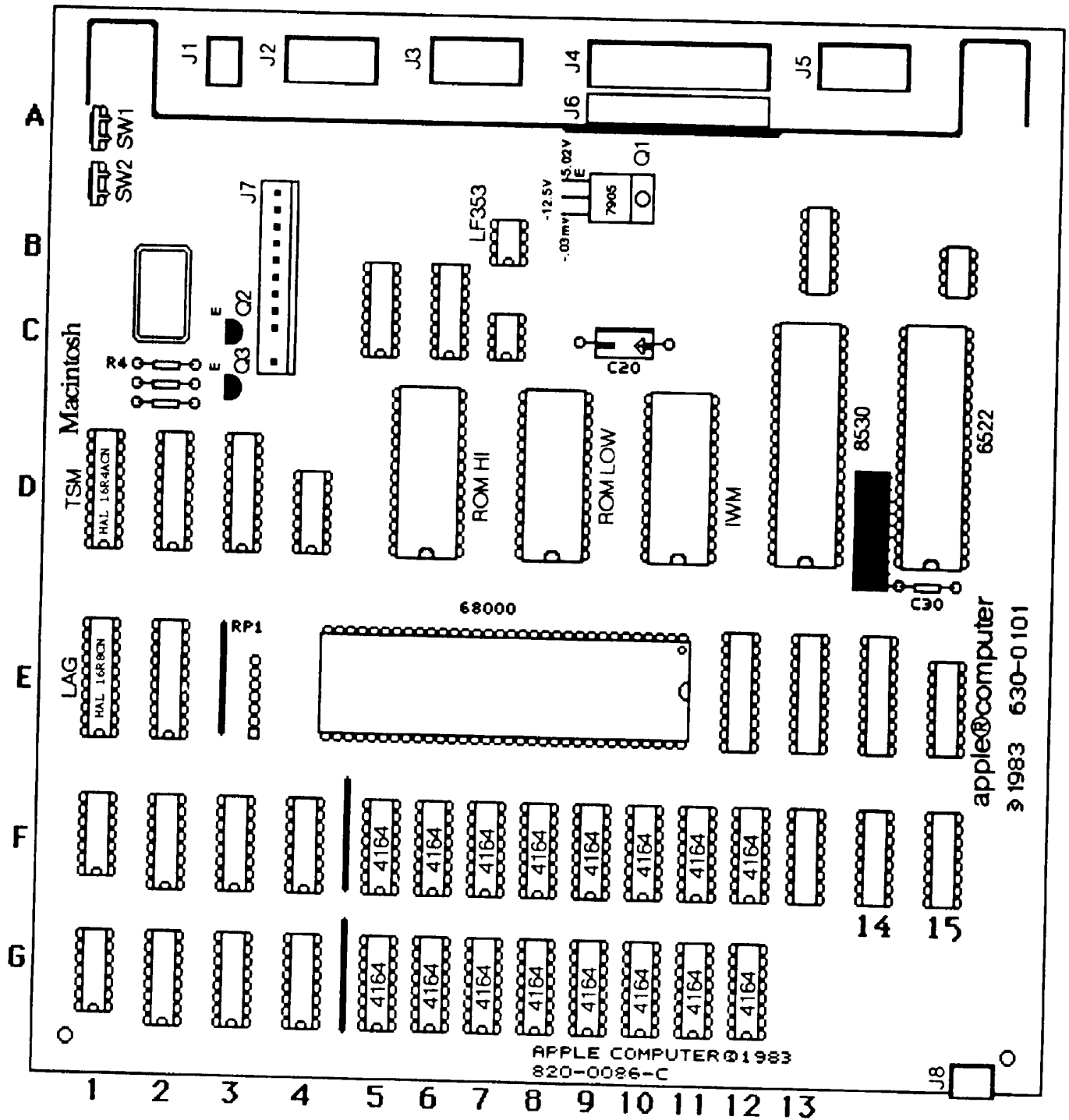


Figure 10-2 Early models of the 128K logic board bear part number 820-0086-C.  
 Courtesy of Sophisticated Circuits, Inc.



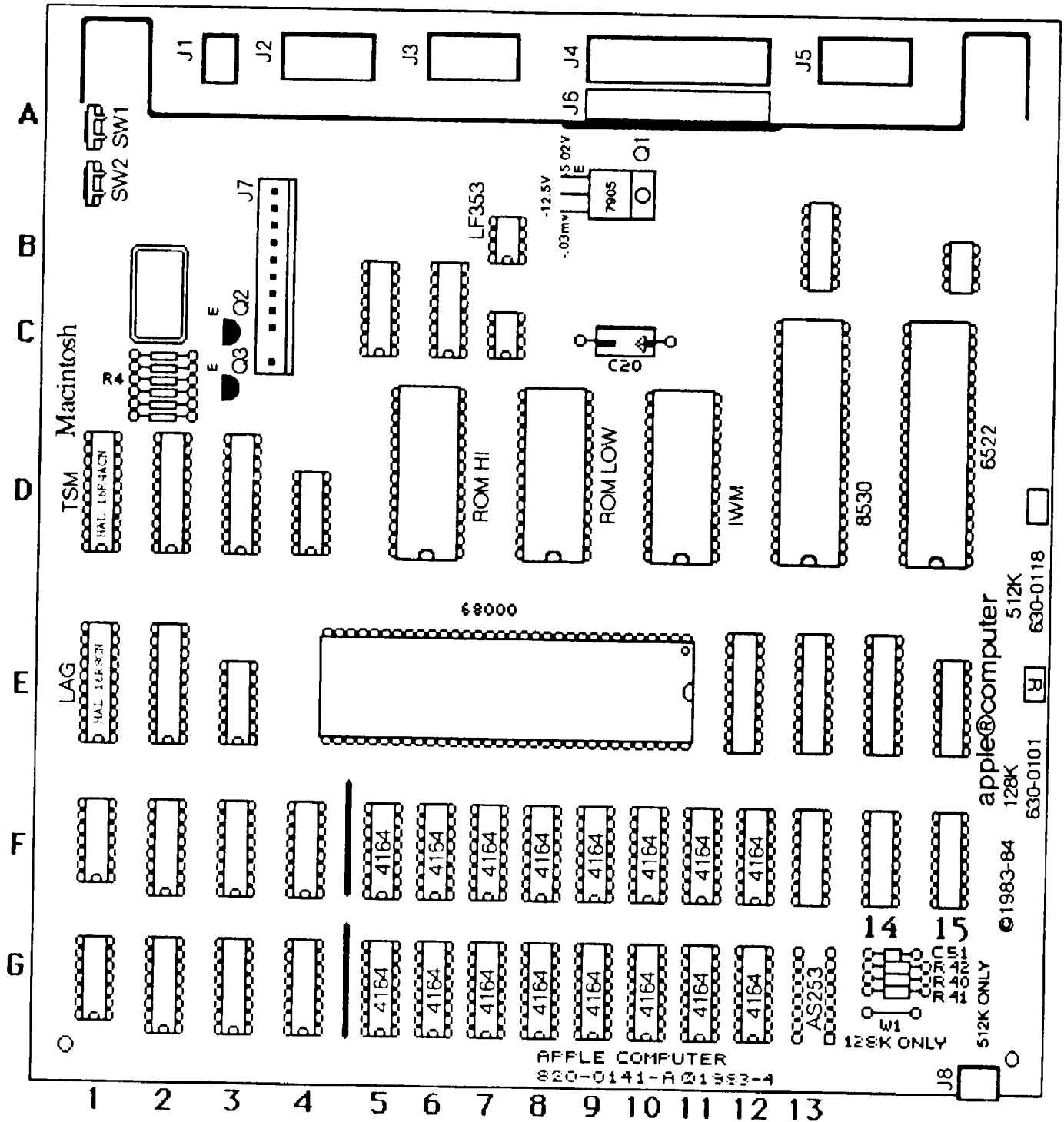


Figure 10-3 Later models of the 128K logic board bear part number 820-0141-A. Courtesy of Sophisticated Circuits, Inc.

# BYTE<sup>12490</sup>®

FEBRUARY 1984 Vol. 9, No. 2  
\$3.50 in USA  
\$3.95 in Canada/£2.10 in U.K.  
A McGraw-Hill Publication  
0360-5280

**BENCHMARKS**

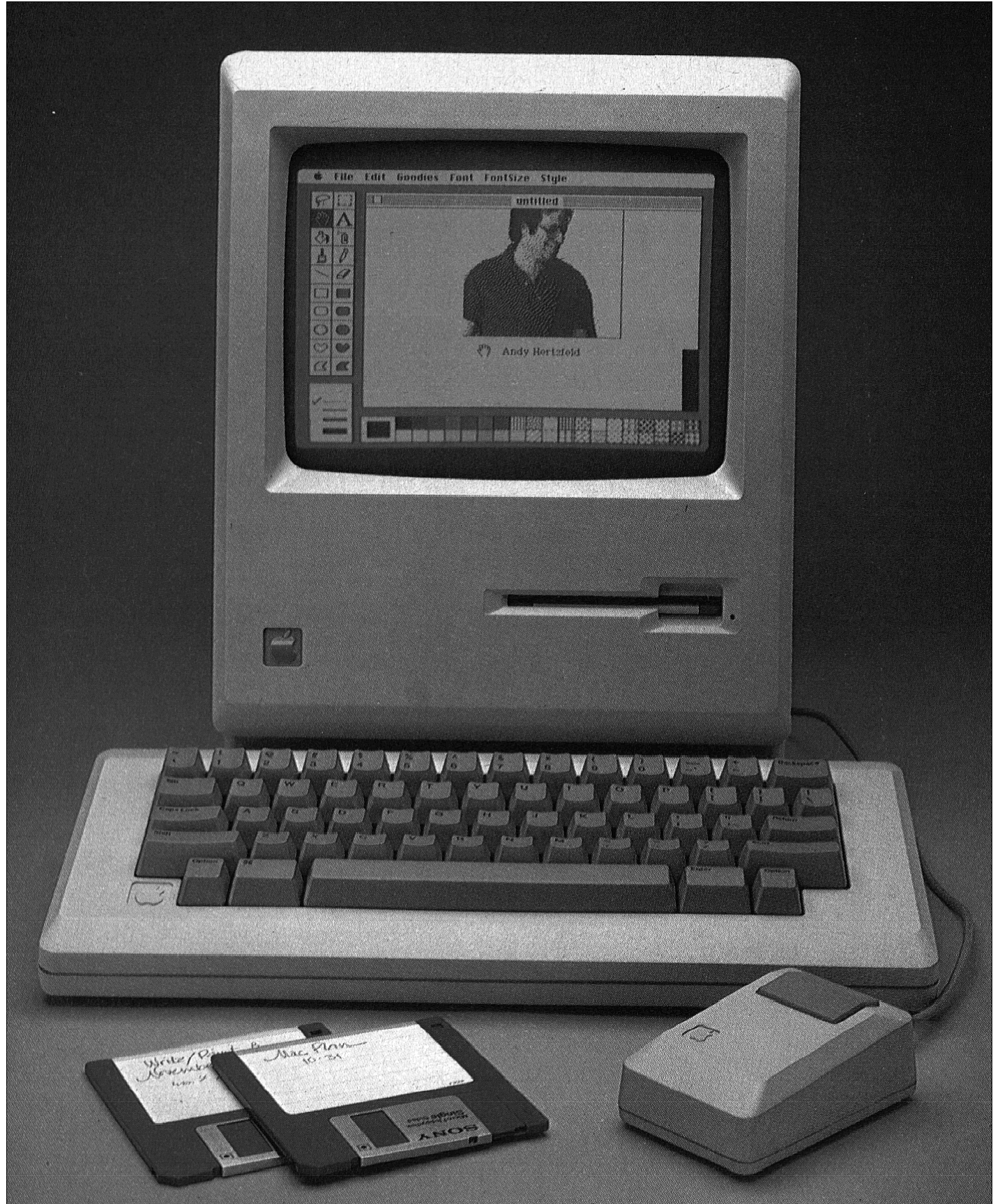
the small systems journal



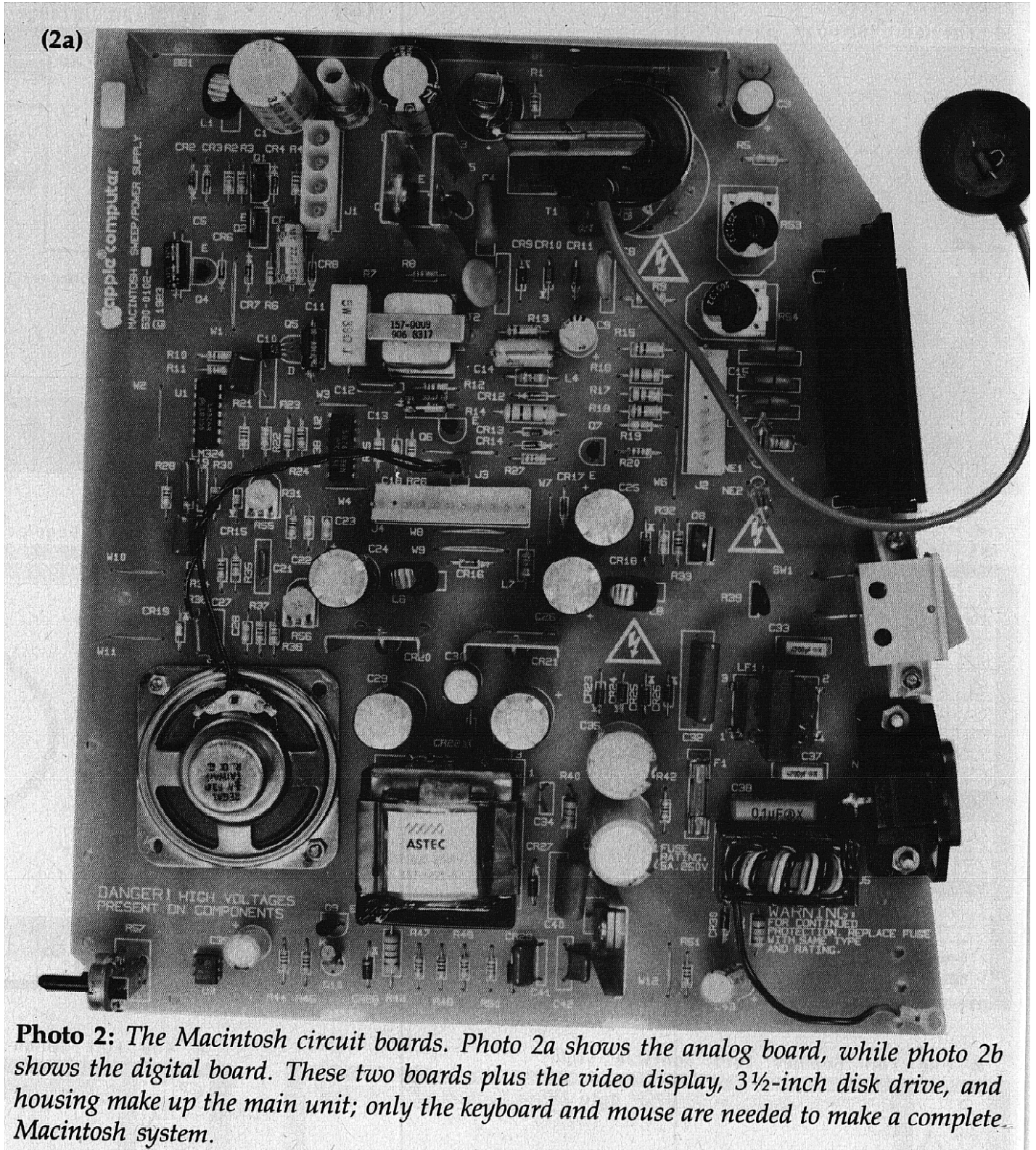
## APPLE'S MACINTOSH

**FEATURES:** Apple/IBM Communications, Calculating Overhead Costs, Inside a Compiler

**REVIEWS:** ProDOS for the Apple IIe, Knowledgeman, Savvy, the IBM 9000, the Rixon R212A Modem



(2a)



**Photo 2:** The Macintosh circuit boards. Photo 2a shows the analog board, while photo 2b shows the digital board. These two boards plus the video display, 3½-inch disk drive, and housing make up the main unit; only the keyboard and mouse are needed to make a complete Macintosh system.







