®

Macintosh® Macintosh Portable

®

Developer Notes

Developer Technical Publications

20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, APDA will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to APDA.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures and tables

Preface About These Developer Notes

These developer notes provide guidelines for developers of hardware and software for the Macintosh Portable computer. It is assumed that the hardware and/or software developer is already familiar with both the functionality and the programming requirements of Macintosh® computers. If you are unfamiliar with the Macintosh, or would simply like more technical information on the hardware, you may want to obtain copies of related technical manuals. For information on how to obtain these manuals, see the paragraph titled "Supplemental reference documents."

These developer notes do not constitute a manual and are not complete in their present form. While every attempt has been made to verify the accuracy of the information presented, it is subject to change without notice. The primary reason for releasing product information is to provide the development community with essential product specifications, theory, and application information for the purpose of stimulating work on compatible third-party products. •

Supplemental reference documents

To supplement the information in this document, hardware/software developers might wish to obtain related documentation, such as the *Guide to Macintosh Family Hardware, Designing Cards and Drivers for the Macintosh Family,* and *Inside Macintosh*, Volumes I through V. Copies of these technical manuals are available through the Apple Programmers and Developers Association (APDA™). APDA is an excellent source of technical information for anyone interested in developing Apple-compatible products. For information about APDA, please contact

APDA
Apple Computer, Inc.
20525 Mariani Avenue, Mailstop 33-G
Cupertino, CA 95014
800-282-APDA (800-282-2732)
AppleLink: APDA
Fax: 408-562-3971
Telex: 171-576

Terminology: Sleep State, Idling State, and the Operating State

The Macintosh Portable ROM software supports the ability to put the computer into the sleep state (clock to DC, all RAM and registers retained) and to bring it back to the operating state. These functions are implemented in the power manager firmware and the power manager processor. The OS requests the sleep state through a time-out scheme or direct user action. Return to the operating state (waking) is due to an event such as a keystroke or wake-up timer going off.

The criterion for the idle state is 15 seconds without user activity of any kind (including communication through the serial port; for example, modem use). In idle, the 68000 processor inserts 64 wait states into RAM and other accesses to lower the processor effective frequency to near 1 MHz, even though its clocking continues at 16 MHz. Interrupts still get processed at 16 MHz, full speed, in the interrupt handler.

For a detailed discussion of these states, see Chapter 6, “The Power Manager.”

Chapter 1 **Introduction**

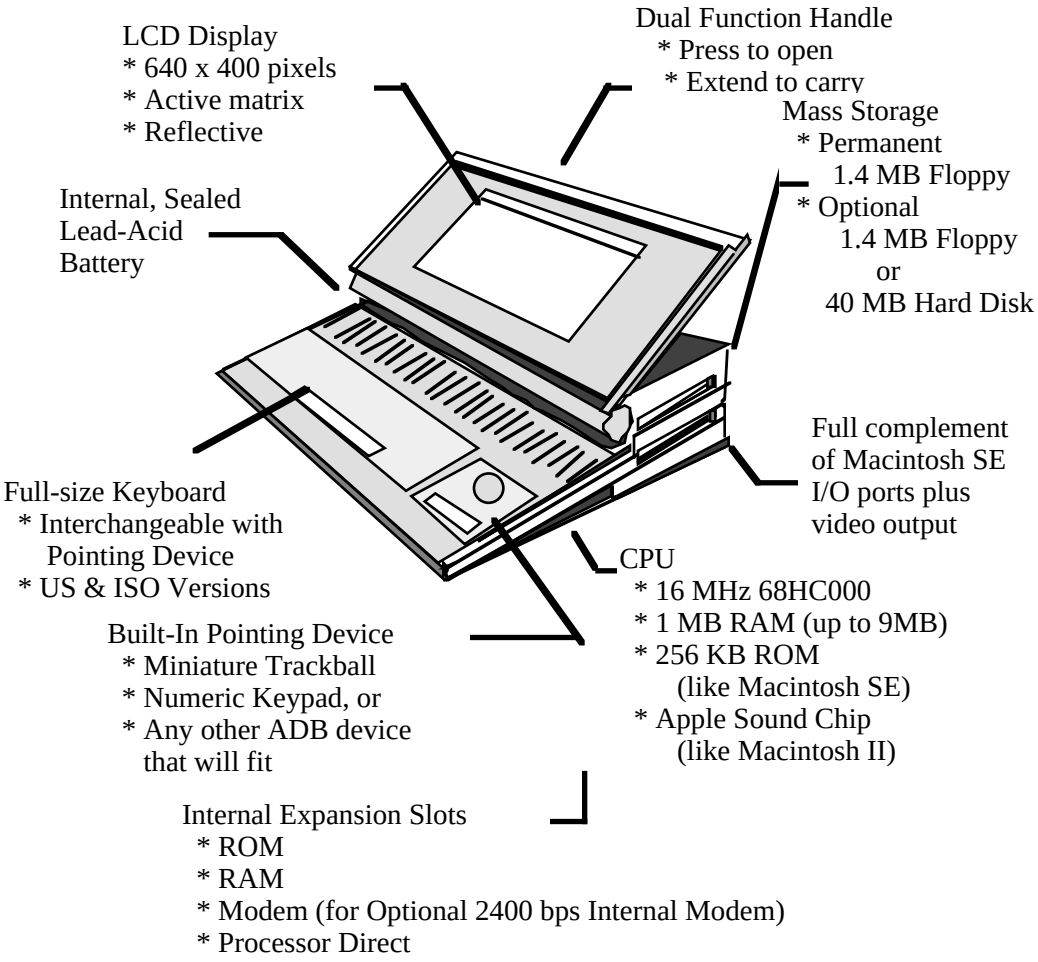
The introduction contains a brief description of the Macintosh Portable, including performance features, appearance, peripherals, and add-ins. •

1.1 Features

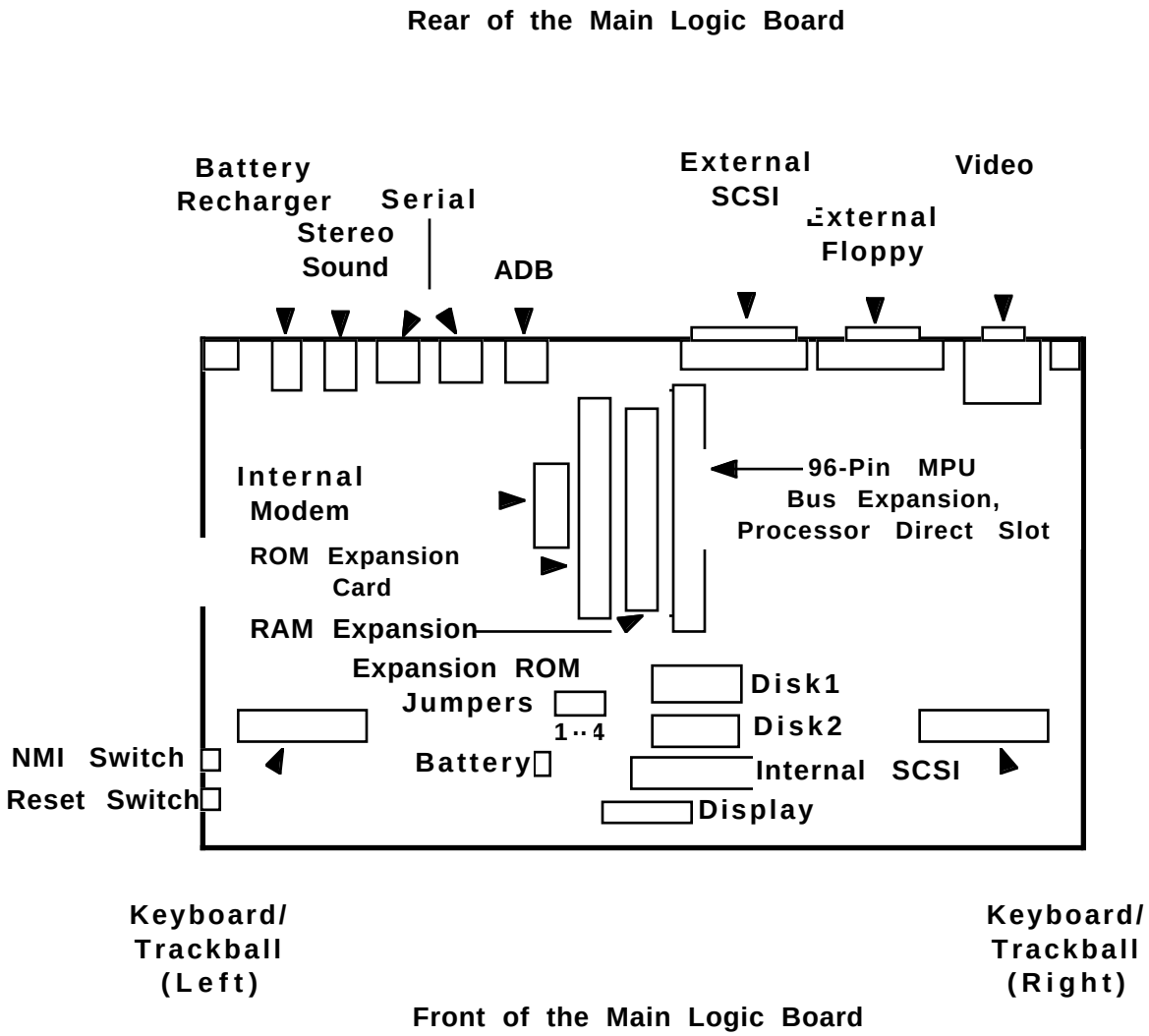
The Macintosh Portable is a full-featured, portable computer designed to meet a wide range of business and personal needs; it is a portable evolution of the Macintosh SE. The Macintosh Portable retains all of the Macintosh SE characteristics and adds several new characteristics for both portability and performance.

Figures 1-1 and 1-2 and Table 1-1 provide a brief summary of some key features of this new machine. Subsequent chapters provide further detail.

- **Figure 1-1** The Macintosh Portable Feature Summary



- **Figure 1-2** I/O Ports and Connectors on the Main Logic Board



- **Table 1-1** The Macintosh Portable Specifications

Characteristic	Specification
CENTRAL PROCESSING UNIT (CPU):	16-bit, CMOS 68HC000, 16 MHz (twice Macintosh SE speed and without video contention), 1 wait state
OPERATING SYSTEM (OS):	Enhanced Macintosh SE ROM
STANDARD MAIN MEMORY:	1MB RAM, 256 KB ROM
MEMORY EXPANSION:	Main memory (RAM) is expandable to 2 or 5 MB by using internal expansion cards (1 or 4 MB). ROM expansion space for Apple use in ROM revision and for international character sets is 1 MB, and for developer use is 4 MB (see Chapter 3, Figure 3-1.)
MASS STORAGE:	Built-in 1.4 MB floppy disk drive External floppy disk drive port Removable/optional second internal 1.4 MB floppy disk drive Optional internal low power SCSI hard disk, also external SCSI port
RAM DISK:	Ability to install system and application software in battery-backed-up RAM so that the machine is capable of functioning without resorting to the optional disk drive. This feature will provide a more rugged, lighter weight solution to portable applications, with longer battery operation and much faster access.
DISPLAY:	Flat-panel, 9.8" diagonal, active matrix reflective LCD, 640 x 400 pixels, 0.33 mm dot pitch, variable tilt
SOUND:	Apple stereo sound chip (same as Macintosh II)

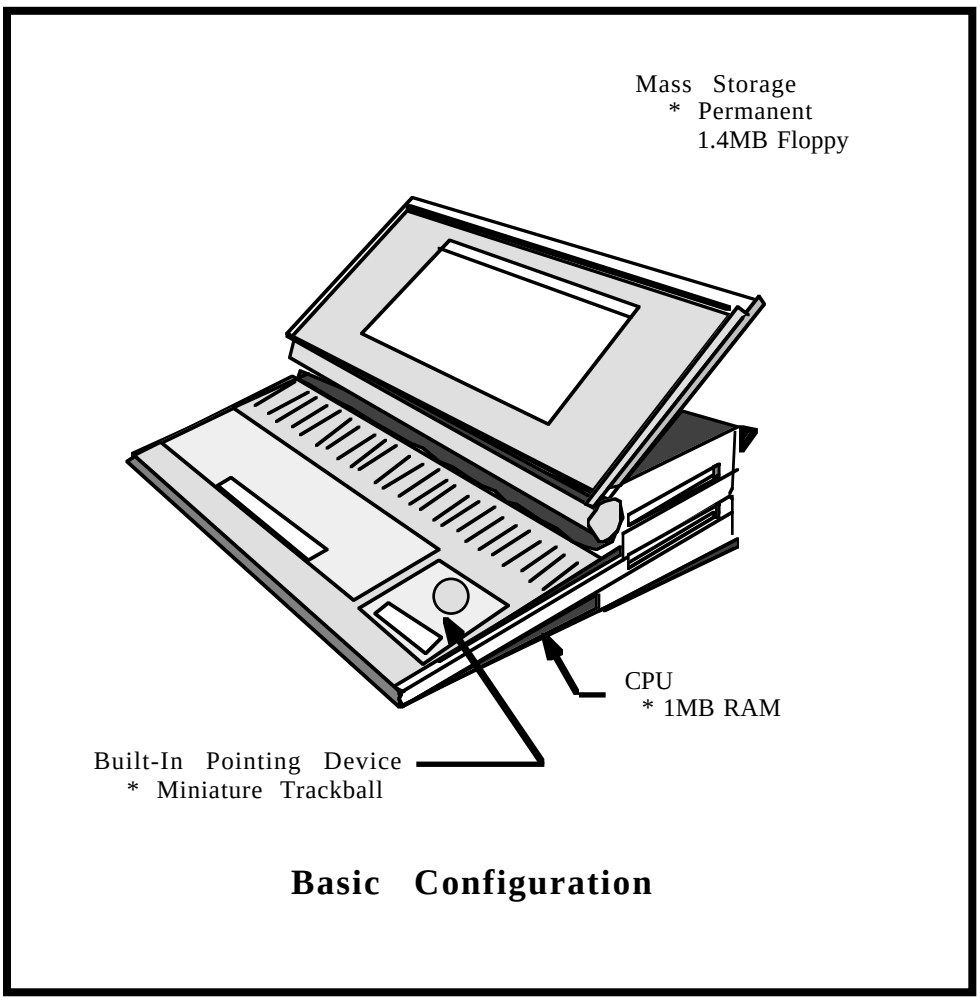
• **Table 1-1** The Macintosh Portable Specifications (Continued)

Characteristic	Specification
I/O PORTS:	DB-19 external floppy disk DB-25 SCSI Mini DIN-4 Apple Desktop Bus port Two Mini DIN-8 serial ports DB-15 for external video 96-pin Euro-DIN expansion interface (not compatible with Macintosh SE) Stereo audio phone jack Battery recharger
INPUT DEVICES:	Built-in keyboard. Built-in trackball replaceable by optional keypad. (The keypad or trackball may be positioned on either side of the keyboard—right side is standard.) Low-power Apple Desktop Bus mouse (optional) plugs into ADB port at the rear of the machine.
OPTIONAL INTERNAL MODEM:	300/1200/2400 bps (AT command set compatible)
WEIGHT:	14 lbs. (minimum configuration) up to 17 lbs. (hard disk option, 5 MB RAM, internal modem option)
SIZE:	15.2" wide x 13.75" deep x 2" to 4" thick (wedge shaped)
SHOCK:	The unit can withstand a 50 G, 12 millisecond shock pulse in any axis while non-operating. This is true of all configurations, for example, with or without a hard disk.
BATTERY USE:	Internal, sealed lead-acid battery provides 8 hours normal use (single floppy configuration), varies dependent on drive usage. Rechargeable overnight (8 hours) using AC power adapter Battery function is required during operation from AC power RAM contents are retained during main battery replacement

Fundamentally, the Macintosh Portable is designed to be rugged and portable, not merely transportable. The Macintosh Portable customer is expected to be anyone who wishes to use a Macintosh away from its usual environment (classroom, office, laboratories), or to move the Macintosh more often than is practical in typical desktop configurations (plugged into an AC wall outlet). Examples of the Macintosh Portable customer are mobile business professionals (for example, auditors, field sales, and field service people) and students.

- **Figure 1-3** The Macintosh Portable Standard Configuration

Basic machine—1 MB RAM, one 1.4 MB floppy disk drive



1.2 Optional additions

The following are optional items:

- 2400 bps domestic modem kit (internal, user installed) *
- 1 MB RAM card (to make a total of 2 MB) *
- second internal 1.4 MB floppy disk drive
- 40 MB internal, one-half height, low-power hard disk drive
- Low-power ADB mouse
- Numeric keypad
- International modem
- External video adapter (see “Peripherals” later in this chapter)

** Cards for internal connectors*

1.3 Internal expansion interfaces

The main logic board has mounted on it five internal connectors for add-in capabilities (see Figure 1-2). These connectors may be listed by function as

- ROM upgrade (4 MB of address space available to you)
- RAM expansion (up to 8 MB of address space available)
- Modem
- Processor direct expansion slot
- Internal SCSI (34-pin)

One connector is for ROM upgrade (see the memory address map in Chapter 3, Figure 3-1) and a second is for RAM expansion. A third connector allows the insertion of an optional 300/1200/2400 bps modem card. The fourth connector is the same 96-pin Euro-DIN type connector used in the Macintosh SE but with a different pinout; this connector provides direct access to the signal lines of the Motorola 68HC000 microprocessor. A fifth connector is used with an internal cable to connect to the optional internal SCSI hard disk drive. These connectors are shown in Figure 1-2. See Chapter 5, "Hardware", for details.

1.4 Peripherals

The Macintosh Portable can accommodate the full range of peripherals available to other members of the Macintosh family (ADB devices should be low-power versions, preferably). In addition, an external video adapter is available for driving larger, CRT displays. This unit is described in Chapter 8; it has three video modes:

- Macintosh II (640 x 400 pixels, with 40 lines blacked at top and bottom of active area)
- NTSC (512 x 400 pixels, with 64 lines blacked at left and right of active area)
- PAL (640 x 400 pixels, with 40 lines blacked at top and bottom of active area)

Chapter 2 **Software Developer Guidelines**

This chapter provides guidelines that will assist you in making sure that the applications you are developing will run on the Macintosh Portable computer when it is introduced. Included is a list of things you should and shouldn't do when writing your application programs. You are encouraged to pay close attention to these guidelines. If you have any questions about the list of Do's and Don'ts, immediately contact Apple Developer Technical Support (AppleLink address MACDTS) and mention the Macintosh Portable product name. •

These guidelines will help to ensure that the software you are developing is compatible with the Macintosh Portable computer. The following chapters describe the hardware and software of the Macintosh Portable and will make clearer some of the terms used in this chapter which may be unfamiliar. The following list stresses things you should be extremely careful about. The Macintosh Portable has some significant differences from the Macintosh SE, and even though the software you are developing may work on the Macintosh SE, it may not work properly on the Macintosh Portable unless you follow these rules.

1. • **Don't** go directly to the Power Manager trap if your software needs to put the Macintosh Portable in the sleep state.

   • **Do** use the new **Sleep** trap.

   A new trap has been added to the Macintosh Portable Operating System. If you are developing a new application and it includes software that can put the Macintosh Portable in the sleep state, such as a smart alarm, you must use the Sleep trap rather than directly accessing the Power Manager trap (see Chapter 6, "The Power Manager".)

2. • **Don't** go directly to the Power Manager trap to set and get time.

   • **Do** use the **SetDateTime** and **GetDateTime** traps.

   The clock in the Macintosh Portable differs slightly from the clock in the Macintosh SE. Use the traps to set the clock and read the time. If you try to access the Power Manager trap directly for these functions, your program will not work correctly.

3. • **Do** use the PmgrOp trap to access the **wake-up timer** commands. This is a new feature.

The Power Manager Op trap is currently the only way to enable the wake-up feature. If you want the Macintosh Portable to return to the operating state from the sleep state at a predetermined time, you can use this trap to access the wake-up timer command. This command enables the wake-up feature (implemented as a timer), and when the real-time clock reaches the preset value, the Macintosh Portable returns to the operating state. If you need to use the auto-wake-up feature, contact Apple Developer Technical Support.

4.
- **Don't** access the ADB (Apple Desktop Bus) hardware directly.
- **Do** use the ADB traps.

Access to the ADB hardware is completely different on the Macintosh Portable than the other Macintosh computers.

5.
- **Don't** talk directly to the SCC (Serial Communication Controller) chip.
- **Do** make normal communications calls to the Serial Driver.

You will be taking a large risk if your program attempts to talk directly to the SCC hardware. The serial chip in the Macintosh Portable is turned off whenever it's not in use. The serial driver knows how to turn the serial chip back on if your program makes normal serial communications calls. However, software that attempts to go directly to the serial chip will wind up talking to the chip when it is turned off, resulting in a loss of communications. If you need to directly access the SCC, you should contact Apple Developer Technical Support.

6.
- **Don't** access any hardware directly (by hardware address).

- **Do** use the appropriate trap to invoke a manager.

7. • **Don't** ignore error checking in your program.

   • **Do** continue to check for errors throughout your code.

   Always check for errors. You might be in the middle of a transaction and get errors that you have not previously experienced on the Macintosh SE. For example, if the Macintosh Portable goes into the sleep state in the middle of an AppleTalk® transaction, the session may have timed out when the execution of your code continues. This example should only be taken as a general warning. In the future, decisions could be made that would prevent the Macintosh Portable from going into the sleep state while programs are running.

   Remember! The user can put the machine into the sleep state at any time. The sleep feature is going to cause some errors that you may not expect, and they can occur any time in the middle of any code.

8. • **Don't** assume any screen size.

   • **Do** make sure your program checks for screen size.

   As an example, lots of older code, like AppleLink®, assumed that if it wasn't running on a Macintosh Plus it must be running on a Lisa® (Macintosh XL) and you ended up with a window that filled the Lisa screen. If you later ran this code on a Macintosh SE, you ended up with a window that was off the screen.

   Do not assume that if the screen size is 640 pixels wide, it's going to be 480 pixels high like the Macintosh II. In fact, don't assume its going to be any particular size; have your code read the size from the appropriate data structure. The Macintosh Portable screen size is actually 640 pixels wide by 400 pixels high.

9. • **Don't** assume only one size disk.

- **Do** be prepared to handle unusual disk sizes.

In addition to the 1.4 MB floppy disk drive, the Macintosh Portable has E-disks (electronic disks, also known as RAM disks) in a user-settable range of sizes to 2 MB. Make sure that your program is prepared to operate with unusual disk sizes.

10. - **Don't** attempt to talk to the IWM chip.

Remember! The Macintosh Portable doesn't have an IWM chip but instead uses a SWIM chip to control the functions of the 1.4 MB floppy disk drive. This is of particular importance to people doing copy protection schemes, and it will completely change the way in which copy protection works.

11. - **Don't** confuse "time" and "ticks"

On the current Macintosh computers, one "tick" is equal to 1/60 of a second in time; therefore, you can assume that if 60 ticks go by, one second in "time" will pass by. However, because of its sleep feature, this does not hold true for the Macintosh Portable. Two different timers run in the Macintosh Portable. If the Macintosh Portable is in the sleep state, "time" keeps running because of the real-time clock, but "ticks" stop until the Macintosh Portable returns to the operating state. If you attempt to use these two constants interchangeably, and the Macintosh Portable goes into the sleep state, your program may become completely confused.

12. - **Do** understand how SysEnvirons handles machine type.

Beginning with System 6.0.2, *all versions* of SysEnvirons may return values larger than those currently defined for machine type. Your program should treat machine types of Zero (= machine unknown), and all values larger than the current largest value, as an unknown machine. *Do Not* check just a specific set of values. For more information, see the October '88 revision of Tech Note 129.

Chapter 3 **Firmware**

This chapter describes the firmware portion of the total software environment for the Macintosh Portable computer. Software will include both ROM-stored code (firmware) and disk-stored code (system software). The Macintosh Portable firmware is an outgrowth of that for the Macintosh SE. This chapter describes the changes from the Macintosh SE firmware (the ROM image). The ROM to which we refer is that on the 68HC000 I/O bus. Chapter 4 describes the changes in the contents of the system tools disk. Also, see "Software Developer Guidelines," Chapter 2. •

This page is a blank

3.1 Overview

The Macintosh SE software is extensively documented in *Inside Macintosh*, Volume V. The contents of Volume V that apply to the Macintosh SE describe its software in terms of changes from the Macintosh Plus, documented in *Inside Macintosh*, Volume IV. Volume IV, in turn, describes changes from the classic Macintosh as documented in *Inside Macintosh*, Volumes I, II, and III.

Terminology

The Macintosh Portable software comes in two components:

- Firmware—contents of the three ROMs, one for each of the three processors (68000 CPU, power manager processor, and the keyboard processor).
- System software—contents of an 800 KB, 3.5" disk, Version 6.0.3. System software includes the system file, ROM patches, cdevs (control devices), and more.

Reference to *Macintosh Portable ROM* will mean the 68000 processor ROM; discussion of ROM firmware for either of the peripheral processors will be explicitly labeled as such.

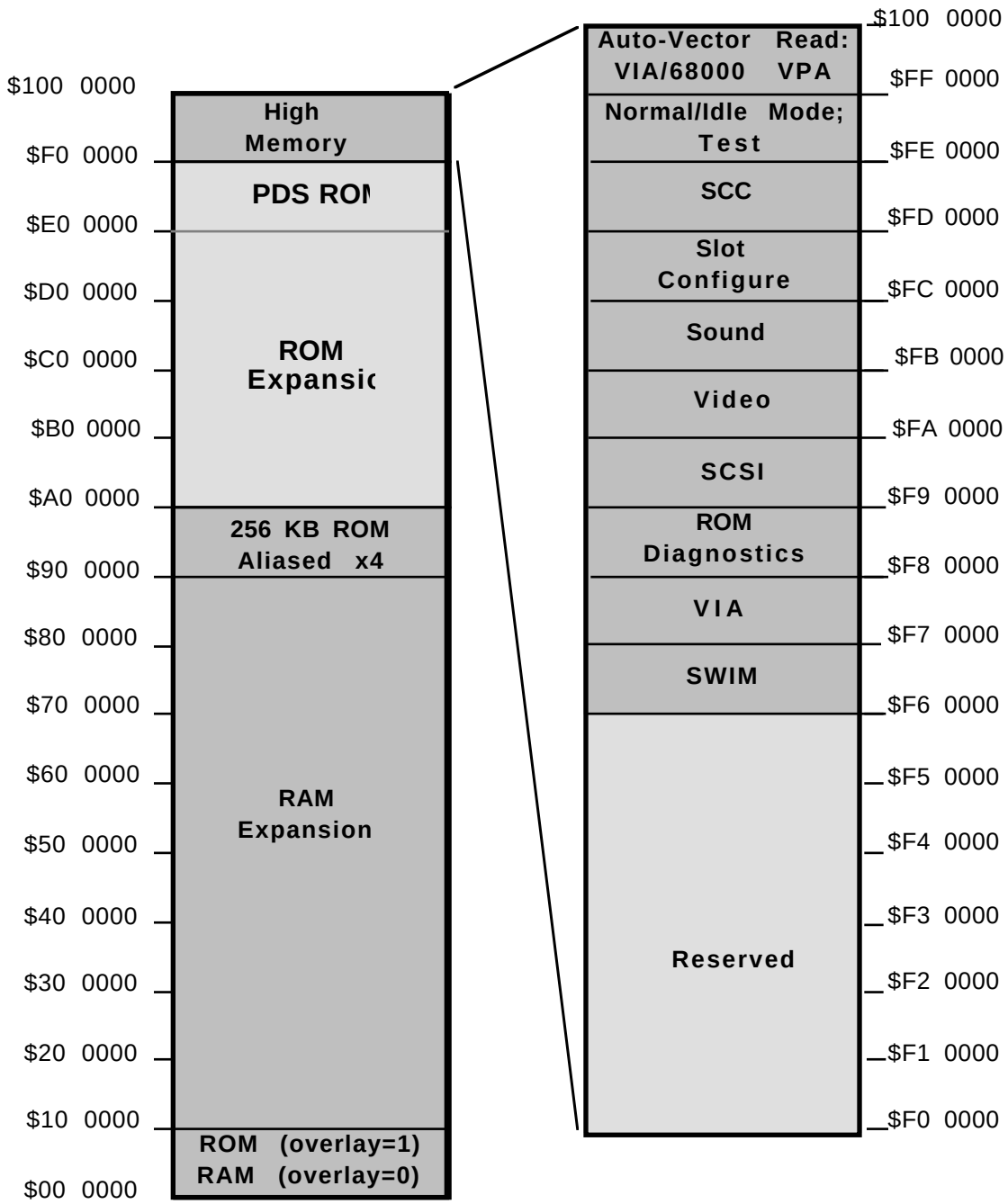
The Macintosh Portable ROM is fundamentally the same ROM as for the Macintosh SE; however, there are some important changes and additions. See section 3.3, “Changes to ROM,” for more detail.

See Chapter 4, “Software,” for information on updates to the system tools disk.

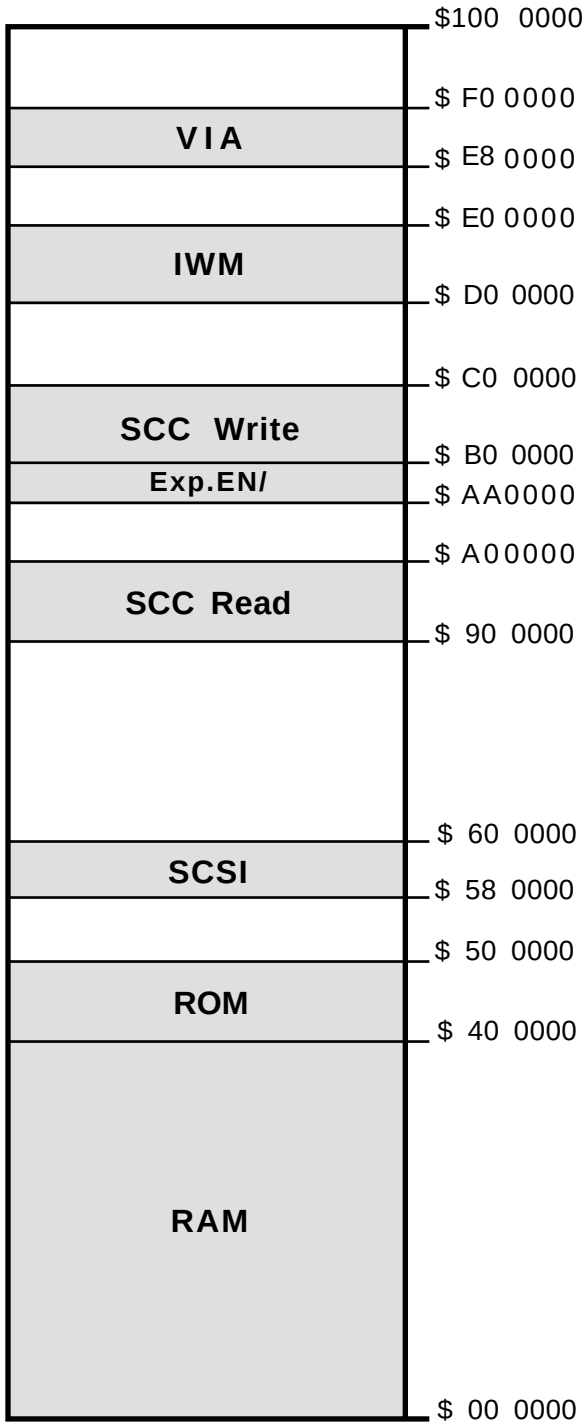
3.2 Address map

Figure 3-1 shows the address map of the Macintosh Portable. For comparison, Figure 3-2 shows the address map of the Macintosh SE.

• **Figure 3-1** The Macintosh Portable Address Map



- **Figure 3-2** The Macintosh SE Address Map



3.3 Changes to ROM

The Macintosh Portable ROM software is based on the 256 KB Macintosh SE ROM; all patches to the Macintosh SE ROM that are contained in disk-stored system software have been incorporated in the ROM image of the Macintosh Portable and no longer need reside in RAM. The Macintosh Portable ROM image is 256 KB in size. It is stored in either masked ROM on the main logic board or (during development) in two 1 megabit ROMs on the ROM expansion board.

Power manager processor

The hardware interface to the power manager processor is through a port on the VIA (see Chapter 6, "The Power Manager," for details); the software interface is through the A trap mechanism (see *Inside Macintosh* Volume I, page I-88). Drivers have been modified to call the power manager to turn on and off their respective peripheral chips.

- **Table 3-1** Power Manager Processor

Replaces	Provides
• Real time clock	-
• Apple Desktop Bus transceiver	-
-	• Power and clock control for peripheral subsystems
-	• A computer wake-up timer facility
-	• LCD screen contrast control
-	• Control of internal modem connection to serial ports
-	• Parameter RAM
-	• Monitoring and control of the battery and charger system

Apple Desktop Bus

Some changes have been made to the code for the ADB state machine to use the power manager as a smart transceiver. Externally, there is no visible change, the same functionality has been provided.

Real-time clock (RTC) and Parameter RAM

As the RTC and parameter RAM functions have been taken over by the power manager, a new interface to the clock is provided. A one-second timer, based on the 60 Hz oscillator used by the power manager processor, is used to generate the real-time clock.

There are two real-time clock functions: one to set and one to read the clock. The clock data is stored as a count of the number of seconds since midnight, 1 January 1904.

Parameter RAM is the storage location for various settings (such as those specified by the user on the Control Panel desk accessory) that need to be preserved during sleep or power off. Only 128 bytes of extended parameter RAM are supported by the power manager. See Chapter 6, "The Power Manager," for further details.

Applications should not use parameter RAM assuming it to be the same as earlier Macintosh models, because it is not. The Macintosh Plus, Macintosh SE, and Macintosh II have 256 bytes as compared to 128 bytes for the Macintosh Portable.

Serial Driver

This driver software is modified for switching power to the SCC chip and the serial driver chips before accessing them.

FDHD, the high-density floppy disk drive

The FDHD™ disk drive is a new 3.5-inch floppy disk drive for the Macintosh II,
Macintosh IIx, Macintosh SE/30, and Macintosh Portable computers. FDHD provides the ability to read and write data in both group code recording (GCR) and modified frequency modulation (MFM) formats. This new MFM capability allows Macintosh users to read and write MS-DOS files and, potentially, other files that use the MFM disk formats.

FDHD provides 1400 KB (1.4 MB) of MFM storage. It also continues to support the standard storage capacities in GCR mode: single-sided 400 KB storage capacity, and double-sided 800 KB storage capacity disks.

The FDHD disk drive provides the high-density read/write hardware and read/write circuitry. The new ROM set incorporates the new disk driver. The SWIM chip provides Macintosh Portable with MFM disk read/write capability while maintaining HFS compatibility. The following information is needed to develop application programs that use the new FDHD 1.4 MB floppy disk drive. Also described are the data encoding techniques used in the drive, the disk driver firmware, and how to use both.

Data storage

The theory behind disk drive technology is relatively easy to understand. By making calls to the disk driver, you are causing the disk controller chip (the SWIM chip) to send control signals and data to the disk drive.

Data is recorded on a disk very much as a voice or music is recorded on magnetic tape: Current flow is varied through a read/write head that is placed close to the medium (the disk or tape). Changing the current flow in the read/write head results in a magnetic transition on the disk. It is these magnetic transitions that interest us.

Several techniques are used by computer designers to write data to disk media. All of these techniques share a common technology: encoding data as magnetic transitions on a disk. To represent individual bits, a pattern of ones and zeros is expressed as a series of magnetic transitions. The data format is the combination of magnetic transitions that represents a particular series of bits. FDHD uses two data formats: group code recording (GCR) and modified frequency modulation (MFM). Table 3-2 shows the four possible combinations of file systems and disk formats.

- **Table 3-2** Possible disk format

Disk formats	File systems
400 KB GCR †	MFS
800 KB GCR †	HFS
720 KB MFM†	MS-DOS
1440 KB MFM *	HFS or MS-DOS

† requires a standard 3.5-inch disk
* requires a high density 3.5-inch disk

GCR format

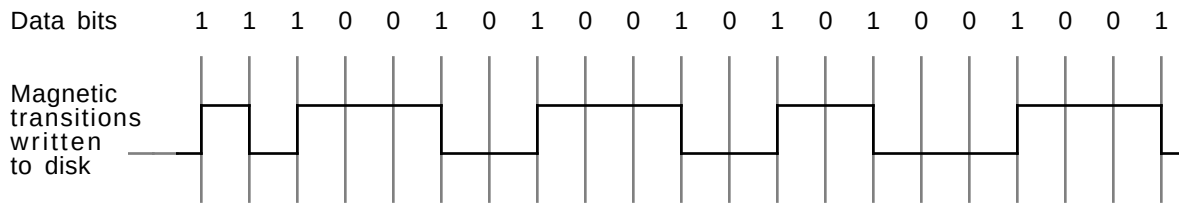
The GCR data format has been used in all Apple floppy disk drives to date, including the Macintosh single-sided and double-sided disk drives.

Data bits are represented by magnetic transitions in the following manner:

- A transition always occurs when a 1 is encountered.
- No transitions occur when a 0 is encountered.

Figure 3-3 shows the relationship between the data bits and the magnetic transitions written to the disk when using the GCR data format.

- **Figure 3-3** GCR data format



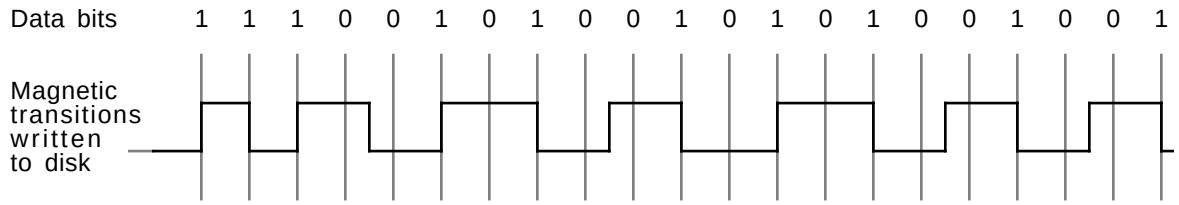
MFM format

Another standard data format is the MFM format, which is used by MS-DOS computers to store data. Data bits are represented by magnetic transitions in the following manner:

- A transition always occurs when a 1 is encountered.
- A transition always occurs when two adjacent 0's are encountered.

Figure 3-4 shows the relationship of data bits and magnetic transitions when using the MFM data format.

- **Figure 3-4** MFM data format

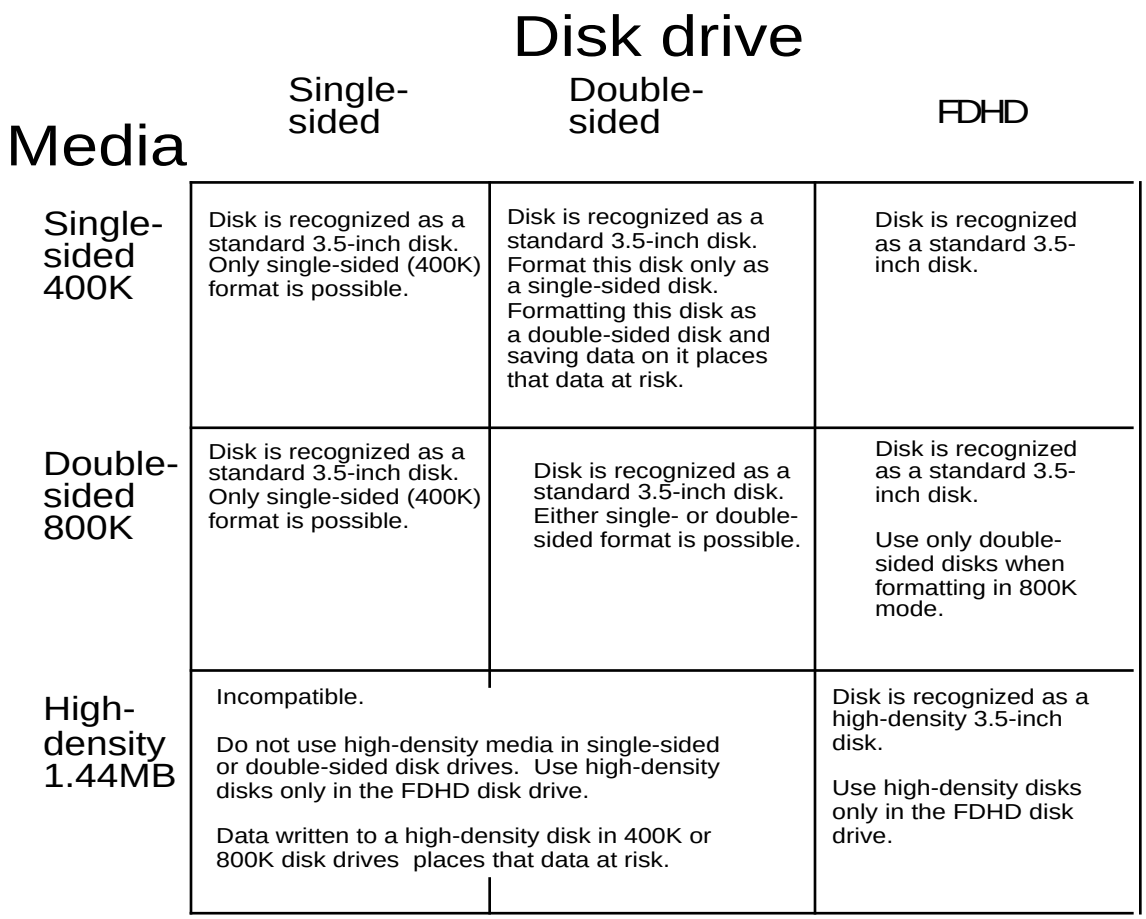


The disk media

The FDHD disk drive uses a special 3.5-inch 1440 KB disk. The disk has a hole cut in the upper-left corner that identifies it as high-density media. Do not format this disk as a 400 KB or 800 KB GCR disk in any Macintosh disk drive. Doing so will place your data at risk. Use only standard Macintosh single-sided and double-sided disks in single-sided (400 KB) and double-sided (800 KB) drives. When a GCR formatted 400K or 800K HD disk is inserted in a FDHD drive, an eject or initialize dialog box will be displayed.

Figure 3–5 shows media compatibility for different disk drives.

- **Figure 3-5** Disk media compatibility



The File System

Apple is working on full support in the file system for the FDHD disk drive in both MFM and GCR modes. Today, support for the FDHD is provided in the current version of the Apple File Exchange (version 1.1). Apple File Exchange is a Macintosh application program that allows users to transfer files from MS-DOS disks to the Macintosh, with the option of translating proprietary file formats. For more information on Apple File Exchange, refer to Chapter 7, "Using Apple File Exchange," in the *Macintosh Utilities User's Guide*.

High Density Floppy Disk Driver

The Macintosh Portable ROM includes a new disk driver. This driver supports the MFM data format and several other new features. This is a new driver to interface with the SWIM chip and support MFM, as well as GCR, data encoding; the driver also provides power control. The calls to this driver are described in the following several pages.

Control calls perform all of the disk operations except reading data and writing data. The control opcode is passed to the driver in the *csCode* field (byte 26) of the I/O parameter block. Refer to the Device Manager chapter in *Inside Macintosh*, Volume II. Control calls that return information pass it back in the I/O parameter block, beginning with the *csParam* field (byte 28).

Kill I/O (csCode=1)

Kill I/O is called to abort any current I/O request in progress. The driver does not support this control call and always returns a result code of –1.

Verify Disk (csCode=5)

This control call reads every sector from the selected disk to verify that all sectors have been written correctly. If any sector is found to be bad, the call aborts immediately and returns an error code.

Format Disk (csCode=6)

If the selected disk is a floppy disk, the driver writes address headers and data fields for every sector on the disk and (for GCR disks only) does a limited verification of the format by checking that the address field of the first sector on each track can be read. If the selected disk is a Hard Disk 20, the driver doesn't format the media, but instead initializes the data of each sector to be all 0's. If any error occurs (including write-protected media), the formatting is aborted and an error code is returned.

The *csParam* field is used to specify the type of format to be done on *floppy* disks only. In the SWIM and later versions of the driver, this value is an index into a list of possible formats for the given drive/media combination. (See the Return Format List status call under "Status Calls," later in this chapter, for values.)

- *Note*: In previous versions of the driver, setting *csParam* to $0001 creates a single-sided disk. Setting *csParam* to a value other than $0001 creates a double-sided disk.

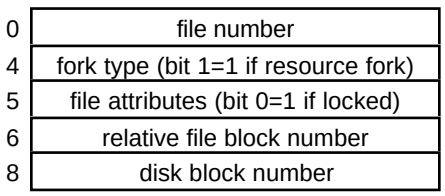
Eject Disk (csCode=7)

This call ejects the disk in the selected drive if that drive supports removable media. Since hard disks are not removable, if a hard disk is ejected, the driver posts a diskInserted event and remounts the drive.

Set Tag Buffer (csCode=8)

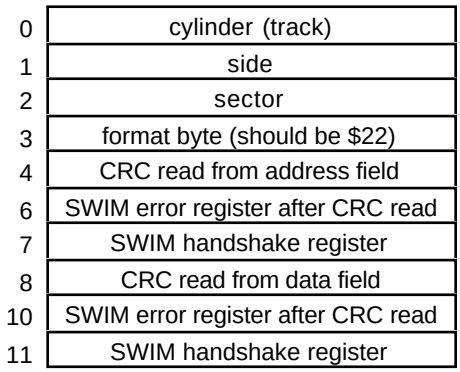
If *csParam* is zero, no separate tag buffer is used. If *csParam* is nonzero, it is assumed to contain a pointer to a buffer into which tag bytes from each block are read or into which they are written on each Prime call. Every time a block is read from the disk, the 12 tag bytes are copied into the file tag buffer at TagData+2 ($2FC), and then are copied into the user's tag buffer. When a block is written, tag bytes are copied into the file tag buffer from the user's tag buffer, and then written to the disk with the rest of the block. The position of a particular block's tag bytes in the user tag buffer is determined by that block's position relative to the first block read/written on the current Prime call. The file tags for GCR disks include information that a scavenging utility can use to rebuild a disk if the directory structure gets trashed. Figure 3–6 shows the tag format.

- **Figure 3-6** GCR file tag format



MFM disks don't support file tags, so instead of scrapping the whole idea, information about the sector itself is returned. Most of it is read from the disk, but the error register bytes show any error conditions that exist after the address or data field is read. Figure 3–7 shows the format of a sector information block.

- **Figure 3-7** MFM sector information block



Track Cache Control (csCode=9)

When the track cache is enabled, all of the sectors on the last track accessed during a read request, and those requested by the user, are read into a RAM buffer. On future read requests, if the track is the same as the last track on the last read request, the sector data is read from the cache rather than from disk. Write requests to the driver are passed directly to the disk, and any of the sectors written that are in the cache are marked invalid. To control the cache, 2 bytes are passed at *csParam* to control the cache, located at *csParam* and *csParam*+1. These codes are shown in Table 3–3 and Table 3–4.

- **Table 3-3** Cache enable codes

csParam	Result
0	Disable the cache.
≠0	Enable the cache.

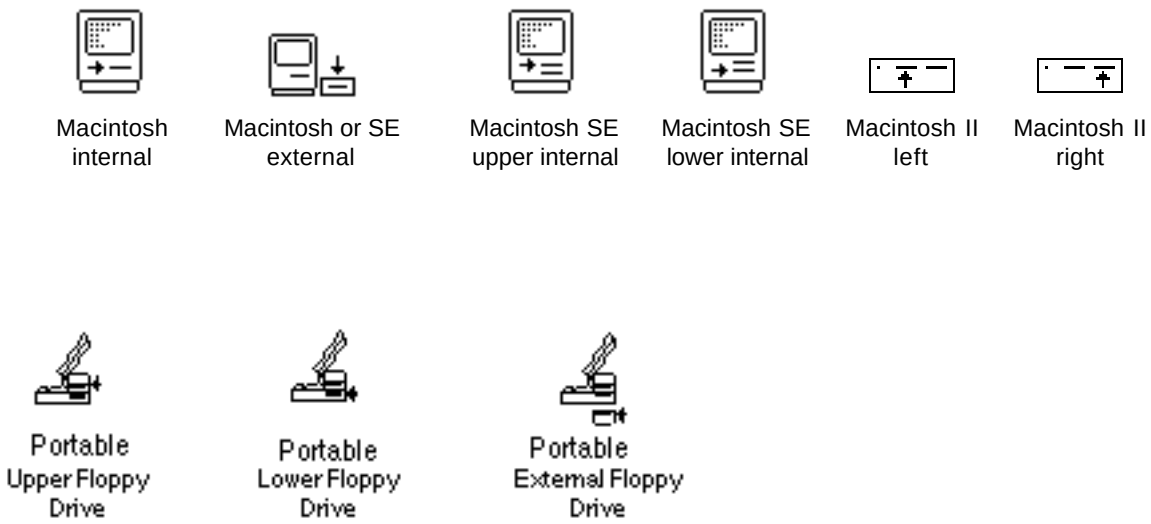
- **Table 3-4** Cache control codes

csParam+1	Result
<0	Remove the cache.
0	Don't remove or install the cache.
>0	Install the cache.

.Return Physical Drive Icon (csCode=21)

This call returns a pointer to an icon that shows the selected drive's physical location. The supported icons are shown in Figure 3–8. Note that only the icons for a particular machine are included in that machine's disk driver.

- **Figure 3-8** Drive icons



Return Media Icon (csCode=22)

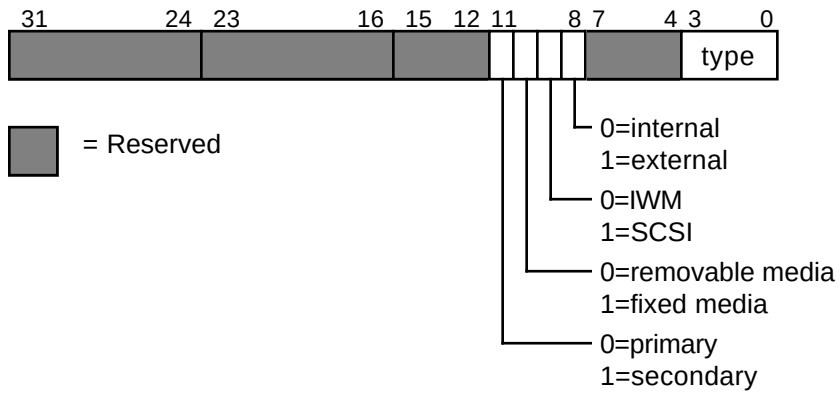
This call returns a pointer to an icon that shows the selected drive's media type. The floppy disk icon is stored in the driver. The Hard Disk 20 icon is stored in the disk drive's ROM. Figure 3–9 shows the icons.

- **Figure 3-9** Media icons

Return Drive Info (csCode=23)

This control call returns a 32-bit value in *csParam* that describes the location and attributes of the selected drive. Figure 3-10 shows the format of the Return Drive Information

- **Figure 3-10** Return drive information format



The attributes field occupies bits 8 to 11 and describes the location (internal/external, primary/secondary), drive interface (IWM/SCSI), and media type (fixed/removable).

Most of the bits are currently not used and are reserved for future expansion. The drive type field occupies bits 0 to 3 and describes the kind of drive that is connected. Currently six different types are supported; they are listed in Table 3–5.

- **Table 3-5** Drive types

Type	Description
0	no such drive
1	unspecified drive
2	400 KB
3	800 KB
4	FDHD (400 KB/800 KB GCR, 720 KB/1440 KB MFM)
5	reserved
6	reserved
7	Hard Disk 20
8-15	reserved

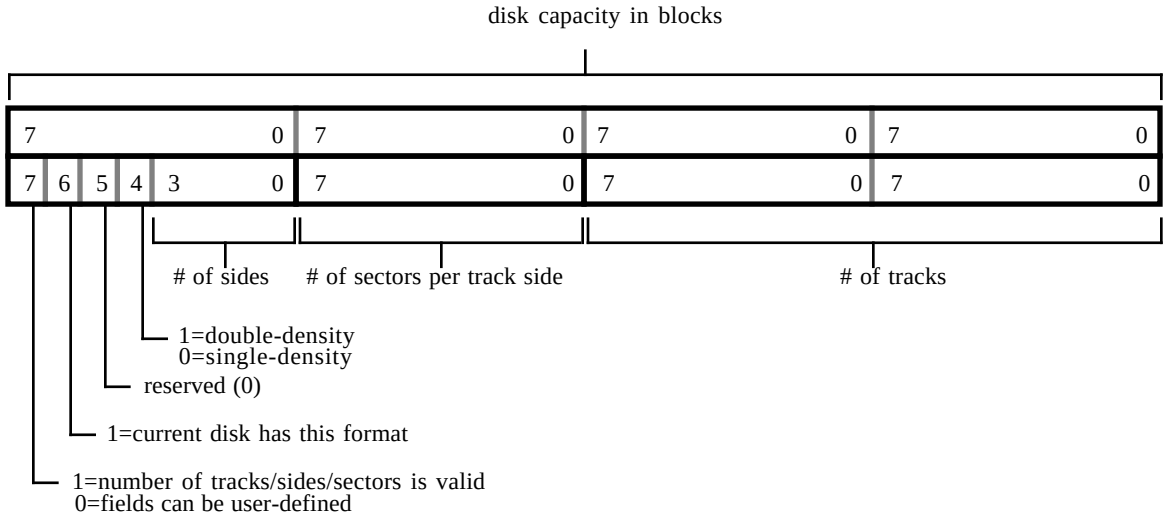
Status calls

The disk driver currently supports three status calls, which are described below. As with the control calls, the status opcode is passed to the driver in the *csCode* field of the I/O parameter block (byte 26). The returned status information is passed back starting at the *csParam* field of the I/O parameter block (byte 28).

Return Format List (csCode=6)

This call is supported in the SWIM-compatible disk driver and will be supported in future versions of the disk driver, whether or not MFM disks are supported. This call returns a list of all disk formats possible with the current combination of disk controller, drive, and media. Upon entry, *csParam* contains a value specifying the maximum number of formats to return, and *csParam+2* contains a pointer to a table that will contain the list. On exit, *csParam* will contain the number of formats returned (no more than specified), and the table will contain the list of formats. If no disk is inserted in the drive, the call will return a *noDriveErr* code. The format information is given in an 8-byte record as shown in Figure 3-11.

• **Figure 3-11** Return format record



If a track, side, or sector field is zero when the *valid* bit is set to 1, the field is considered to be a "don't care" as far as describing the format of the disk. The formats supported by the driver are listed in Table 3–6.

• **Table 3-6** Combinations of drives and media

Format	Capacity (in blocks)	TSS valid[1]	SD/DD	Sides	Sectors[2]	Tracks
400 KB GCR	800	yes	SD	1	10	80
800 KB GCR	1600	yes	SD	2	10	80
720 KB MFM[3]	1440	yes	SD	2	9	80
1440 KB MFM[3,4]	2880	yes	DD	2	18	80
Hard Disk 20	38965	no	SD	0	0	0

Notes:

[1] track, sector, and side information

[2] average number of sectors

[3] requires SWIM and FDHD

[4] requires HD media

Drive Status (csCode=8)

Drive Status returns information about a particular drive, starting at *csParam*; the values returned are listed in Table 3–7.

- **Table 3-7** Drive status return values

Offset	Description	Parameters
0	current track	value of current track
2	write protect	bit 7 = 1, write-protected
		bit 7 = 0, write-enabled
3	disk in place?	<0 = disk is being ejected
		0 = no disk is currently in the drive
		1 = disk was just inserted but no read/write requests have been made for this disk
		2 = OS has tried to mount the disk (that is, read request to driver)
		3 = same as "2" except that this is a high-density disk formatted as 400KB or 800KB GCR
		8 = same as "2" except except that this is a Hard Disk 20 (8 means disk is nonejectable)
4	drive installed?	–1 = no drive installed
		0 = don't know
		1 = drive installed
5	number of sides	0 = single-sided
		–1 = double-sided

• **Table 3-7** Drive status return values (Continued)

Offset	Description	Parameters
6	drive queue element	6 = qLink—pointer to next queue element
		10 = qType—type of queue (drvQType)
		12 = dqDrive—drive number
		14 = dqRefNum—disk driver's reference number
		16 = dqFSID—file system ID
18	double-sided format?	0 = current disk has single-sided format
		–1 = current disk has double-sided format
19	new interface	0 = old drive interface (400KB)
		–1 = new interface (800KB and later)
20	soft error count (2 bytes)	number of soft errors encountered

MFM Status (csCode=10)

This call is supported in the SWIM-compatible disk driver and will be supported in future versions of the disk driver. By making this call and then checking the returned error code, it is possible to determine whether or not the version you are using can read and write MFM disks. Also, the information returned is helpful in determining the installed hardware configuration. The information is returned starting at *csParam*. Table 3-8 lists the values returned.

- **Table 3-8** MFM status return values

Offset	Description	Parameters
0	drive type	–1 = FDHD (MFM/GCR) 0 = 400KB or 800KB GCR
1	disk format	–1 = MFM 0 = GCR (valid only when installed)
2	MFM format	–1 = 1440KB disk 0 = 720KB disk
3	disk controller	–1 = SWIM 0 = IWM

A sample program

As an example of how to use the new disk driver, a sample application program is included here.

```
{
_____________________________________________________________________

    FDHD™ Driver Demo Application

    Copyright © 1988 by Apple Computer, Inc.


This is a short application to show off both the old and new control and status calls
for the FDHD disk driver.  The application does this by presenting a list of
information about each 3.5-inch floppy or original Hard Disk 20 drive connected to
the Macintosh, and the format of any disks that are in those drives.

_____________________________________________________________________
}

PROGRAM HDFDDriverDemo;

 USES       MemTypes,QuickDraw,OsIntf,ToolIntf,PackIntf,PasLibIntf;

 CONST      DemoMenu    = 256;                  {menu's resource ID and item numbers}
            AboutItem   = 1;
            QuitItem    = 3;
```

```
DemoDLOG        = 256;                  {"demo" dialog's resource ID and item
                                            numbers we use:}
    Eject1Btn       = 1;                {disk eject buttons}
    Eject2Btn       = 2;
    Eject3Btn       = 3;
    ChipTypTxt= 11;                     {disk controller type}
    BaseDrvLoc= 12;                         {base drive location}
    BaseDrvIcn= 15;                     {base drive icon item}
    BaseDskIcn= 18;                     {base disk icon item}
    BaseDrvNum= 21;                     {base drive number}
    BaseDrvTyp= 24;                     {base drive type}
    BaseDskFmt= 27;                     {base disk format}

StrRsrcID     = 256;                    {descriptive string list resource ID and
                                            string numbers:}
    IWMStr          = 1;                    {chip types}
    SWIMStr         = 2;
    Int1DrvStr = 3;                     {primary internal drive}
    Int2DrvStr = 4;                     {secondary internal drive}
    Ext1DrvStr = 5;                     {primary external drive}
    Ext2DrvStr = 6;                     {secondary external drive}
    Drive400K   = 7;                    {drive types}
    Drive800K   = 8;
    FDHD = 9;
    Hard Disk 20 = 12;

DefDiskIcon     = 256;                  {default drive/disk icon in case we have
                                            errors}

    HDFDRefNum      = -5;               {FDHD driver's reference number}
    FmtListCode     = 6;                {csCode for "format list" status call}
    DrvStsCode  = 8;                    {csCode for "drive status" status call}
    MFMStsCode      = 10;               {csCode for "MFM status" status call}
    DrvIconCode     = 21;               {csCode for "drive icon" control call}
    DskIconCode     = 22;               {csCode for "disk icon" control call}
    DrvInfoCode     = 23;               {csCode for "drive info" control call}
```

```
    TYPE    DrvFmtRec = PACKED RECORD                   {disk format description:}
        capacity: LONGINT;                              { number of blocks on the disk}
        flagsNHeads: SignedByte;                        { [flags][number of heads]}
        sectors: SignedByte;                            { number of sectors per track side}
        cylinders: INTEGER;                             { number of tracks [cylinders]}
        END;

        FmtInfoRec    = RECORD                          {format list:}
            numFormats  : INTEGER;                      { number of formats we want/are returned}
            fmtBlock    : Ptr;                          { where to put them}
            END;

        FmtInfoPtr    = ^FmtInfoRec;

        MFMSts          = PACKED RECORD                 {info about chips, drives, disks:}
            isHDFD: SignedByte;                         {-1=HDFD, 0=400K/800K drive}
            diskFormat  : SignedByte;                   { -1=MFM, 0=GCR}
            twoMegFmt   : SignedByte;                   {-1=1440K, 0=720K (if diskFormat=-1)}
            isSWIM          : SignedByte;               {-1=SWIM, 0=IWM}
            END;

        MFMStsPtr   = ^MFMSts;

        DrvStsPtr     = ^DrvSts;


    VAR     theEvent      : EventRecord;
        demoDialog  : DialogPtr;                        {"the" window}
        whichWindow     : WindowPtr;                    {the window FindWindow is talking
about}
        pb: ParamBlockRec;                              {parameter block for control/status
calls}
        itemHit       : INTEGER;                        {item number returned by DialogSelect}
        driveIcon,                                      {drive icon for each column}
        diskIcon      : ARRAY[0..2,0..31] OF LONGINT;   {disk icon for each column}
        driveNum      : ARRAY[0..2] OF INTEGER;         {drive number for each column}
```

```
{
_____________________________________________________________________

                        DIALOG ROUTINES
_____________________________________________________________________
}

 { hides the specified dialog control }

PROCEDURE HideDControl(theItem:INTEGER);
    VAR     theType             : INTEGER;
            theControl    : ControlHandle;
            theRect       : Rect;
    BEGIN
     GetDItem(demoDialog,theItem,theType,Handle(theControl),theRect);
     HideControl(theControl);
    END;


 { sets the highlighting of the specified dialog control }

PROCEDURE HiliteDControl(theItem,theHilite:INTEGER);
    VAR     theType             : INTEGER;
            theControl    : ControlHandle;
            theRect       : Rect;
    BEGIN
            GetDItem(demoDialog,theItem,theType,Handle(theControl),theRect);
            HiliteControl(theControl,theHilite);
    END;


 { changes the text of a staticText item to theText }

PROCEDURE SetDText(theItem:INTEGER; theText:Str255);
    VAR     theType             : INTEGER;
            theHandle     : Handle;
            theRect       : Rect;
    BEGIN
            GetDItem(demoDialog,theItem,theType,theHandle,theRect);
            SetIText(theHandle,theText);
    END;
```

```
{ draws a drive icon for the given drive }

PROCEDURE DrawDriveIcon(theDialog:DialogPtr; theItem:INTEGER);
    VAR     theBits         : BitMap;
        theType             : INTEGER;
        theHandle     : Handle;
    BEGIN
        IF driveNum[theItem-BaseDrvIcn]<>0 THEN BEGIN
            theBits.baseAddr:=@driveIcon[theItem-BaseDrvIcn];
            theBits.rowBytes:=4;
            GetDItem(theDialog,theItem,theType,theHandle,theBits.bounds);
            CopyBits(theBits, theDialog^.portBits, theBits.bounds, theBits.bounds,srcCopy,NIL);
        END;
    END;


{ draws a disk icon for the given drive }

PROCEDURE DrawDiskIcon(theDialog:DialogPtr; theItem:INTEGER);
    VAR     theBits         : BitMap;
        theType             : INTEGER;
        theHandle     : Handle;
    BEGIN
        IF driveNum[theItem-BaseDskIcn]<>0 THEN BEGIN
            theBits.baseAddr:=@diskIcon[theItem-BaseDskIcn];
            theBits.rowBytes:=4;
            GetDItem(theDialog,theItem,theType,theHandle,theBits.bounds);
            CopyBits(theBits,theDialog^.portBits,theBits.bounds, theBits.bounds,srcCopy,NIL);
        END;
    END;
```

```
{
_____________________________________________________________________

              MISCELLANEOUS DISK STUFF
_____________________________________________________________________
}

 { Pascal doesn't support    a SWAP operation, and since the Hard Disk 20's }
 { drive size fields are in the opposite order from what we need, we'll }
 { have to swap them ourselves. The hex code following the function        }
 { translates to:                          }

 {     $205F   MOVEA.L    (SP)+,A0;                          Get the pointer to "driveSize" }
 {     $2010   MOVE.L      (A0),D0;                               D0.L=[driveSize][driveS1] (backwards)
}
 {     $4840   SWAP    D0;                                        D0.L=[driveS1][driveSize] (how we
want it) }
 {     $2E80   MOVE.L      D0,(SP);                               Put the result on the stack }


FUNCTION GetHD20Size(rawSize:Ptr):LONGINT; INLINE $205F, $2010, $4840, $2E80;

 { check what format this disk has and enable the Eject button }

PROCEDURE NewDisk(theDrive:INTEGER);
      VAR      theFormat     : LONGINT;
           I                 : INTEGER;
           formatList    : FmtInfoPtr;
           formatInfo    : ARRAY[0..3] OF DrvFmtRec;
           driveStatus   : DrvStsPtr;
           theString     : Str255;
```

```
BEGIN
    IF theDrive IN [1..3] THEN                          {it might be one of ours}
        IF driveNum[theDrive-1]<>0 THEN BEGIN           {it is, so continue}
            theFormat:=0;                               {no format yet}

{ First, assume we're working with a recent enough }
{ version of the driver to support the Format List call.  If }
{ so, we can determine how big this disk is…                 }

            formatList:=FmtInfoPtr(@pb.csParam);        {type coercion (hazard of
                                                        Pascal)…}
            formatList^.numFormats:=4;                  {we want up to 4 entries}
            formatList^.fmtBlock:=@formatInfo;          {and here's where to put
                                                        them}

            pb.ioCompletion:=NIL;                       {no completion routine}
            pb.ioRefNum:=HDFDRefNum;                    {FDHD driver's reference
                                                        number}
            pb.ioVRefNum:=theDrive;                     {drive number}

            pb.csCode:=FmtListCode;                     {go get the list}
            IF PBStatus(@pb,FALSE)=NoErr THEN BEGIN     {got something back}
                I:=formatList^.numFormats;
                WHILE (I>0) AND (theFormat=0) DO BEGIN  {scan the list for this disk's
                                                        format}
                    I:=I-1;
                    IF BTST(formatInfo[I].flagsNHeads,6) THEN   {bit 6=1 means this is the
                                                        current format}
                        theFormat:=formatInfo[I].capacity DIV 2;    { save the disk's size in K-
                                                        bytes}
                END;
            END;
```

```
    { If theFormat=0 then the Format List call }
    { isn't supported (because it's an old driver version)…  }

                IF theFormat=0 THEN BEGIN                       {haven't figured it out yet}
                    pb.csCode:=DrvStsCode;                      {get drive status}
                    IF PBStatus(@pb,FALSE)=NoErr THEN BEGIN
                        driveStatus:=DrvStsPtr(@pb.csParam);    {type coercion…}
                        IF driveStatus^.diskInPlace=8 THEN      {Hard Disk 20}
                            theFormat:=GetHD20Size(@driveStatus^.driveSize) DIV 2       {blocks-
                                                > K-bytes}
                        ELSE
                        IF driveStatus^.twoSideFmt=0 THEN
                            theFormat:=400                      { 400K}
                        ELSE theFormat:=800;                    { 800K}
                    END;
                END;
                NumToString(theFormat,theString);               {convert the disk size to a
                                                                string,}
                theString:=CONCAT(theString,'K');               {append a special K to the
                                                                size,}
                SetDText((BaseDskFmt-1)+theDrive,theString);    { and display it}
                HiliteDControl((Eject1Btn-1)+theDrive,0);       {enable the Eject button}
            END;
        END;

{
_____________________________________________________________________

                        I N I T I A L I Z A T I O N
_____________________________________________________________________
}

PROCEDURE Initialize;
    CONST  ControlErr   = -17;
    VAR column,I,driveType    : INTEGER;
        theHandle,defIcon           : Handle;
        theString       : Str255;
        mfmStatus       : MFMStsPtr;
        driveStatus     : DrvStsPtr;

    { sets a dialog userItem's draw procedure to theProc }

PROCEDURE SetUserProc(theItem:INTEGER; theProc:ProcPtr);
```

```
    VAR      theType   : INTEGER;
         theHandle : Handle;
         theRect   : Rect;
    BEGIN
         GetDItem(demoDialog,theItem,theType,theHandle,theRect);
         SetDItem(demoDialog,theItem,theType,Handle(theProc),theRect);
    END;

    BEGIN
         InitGraf(@thePort);                                        {initialize the managers}
         InitFonts;
         FlushEvents(everyEvent,0);
         InitWindows;
         InitMenus;
         TEInit;
         InitDialogs(NIL);
         InitCursor;

         demoDialog:=GetNewDialog(DemoDLOG,NIL,WindowPtr(-1));          {load in the demo
                                                                  dialog window}

    { find out what kind of disk controller chip we're using }

         pb.ioCompletion:=NIL;                                      {no completion routine}
         pb.ioRefNum:=HDFDRefNum;                                   {FDHD driver's reference
                                                                  number}
         pb.ioVRefNum:=1;                                           {any drive number will do}

         I:=IWMStr;                                                 {assume we're working
                                                                  with an IWM}
         pb.csCode:=MFMStsCode;                                         {get MFM status}
         IF PBStatus(@pb,FALSE)=NoErr THEN BEGIN
              mfmStatus:=MFMStsPtr(@pb.csParam);                    {type coercion…}
              IF mfmStatus^.isSWIM<0 THEN I:=SWIMStr;               {we've got a SWIM chip}
         END;

         GetIndString(theString,StrRsrcID,I);                      {get the chip type string}
         SetDText(ChipTypTxt,theString);                           { and display it}

{ set the icon draw procedures and fill in all drive information }

         defIcon:=GetResource('ICON',DefDiskIcon);                 {get the default disk icon}
                                                                   { in case we get errors}
```

```
        theHandle:=Handle(@pb.csParam);                         {a little type coercion…}
        driveStatus:=DrvStsPtr(@pb.csParam);                    {here too…}

        FOR column:=0 TO 2 DO BEGIN
            SetUserProc(BaseDrvIcn+column,@DrawDriveIcon);      {set the drive icon's draw
                                                                proc}
            SetUserProc(BaseDskIcn+column,@DrawDiskIcon);           {set the disk icon's
                                                                draw proc}

{ find out if the drive is installed or not, and if it belongs to the FDHDdriver }

            driveStatus^.installed:=-1;                             {make sure this one is
                                                                inited}
            pb.ioVRefNum:=column+1;                             {drive number}
            pb.csCode:=DrvStsCode;                              {find out about this drive}
            IF (PBStatus(@pb,FALSE)=NoErr) AND(driveStatus^.installed>=0)
                THEN BEGIN          {if the drive is installed then:}
                    driveNum[column]:=pb.ioVRefNum;             { save its number,}
                    NumToString(pb.ioVRefNum,theString);        { convert it to a string,}
                    SetDText(BaseDrvNum+column,theString);      { and display it}

                    IF driveStatus^.diskInPlace<2 THEN          {the disk isn't quite
                                                                mounted,}
                        HiliteDControl(Eject1Btn+column,255)    { so disable its Eject
                                                                button}
                    ELSE
                        NewDisk(column+1);                      { otherwise find out about
                                                                it}

{ Check the drive type here in case the "get drive info" }
{ call isn't supported on this particular machine…      }

                    IF driveStatus^.diskInPlace=8 THEN BEGIN    {only Hard Disk 20 is not
                                                                ejectable}
                        driveType:=HD20;                        {save its drive type}
                        HideDControl(Eject1Btn+column);         { and get rid of its Eject
                                                                button--}
                    END                                         {  it's not ejectable,
                                                                remember?}
                    ELSE
                    IF driveStatus^.sides=0 THEN                    {otherwise go by number of
                                                                sides}
                        driveType:=Drive400K
```

```
                ELSE driveType:=Drive800K;

{ Get the drive's icon or use our generic one if we can't get it }

                pb.csCode:=DrvIconCode;
                IF PBControl(@pb,FALSE)=NoErr THEN
                    BlockMove(theHandle^,@driveIcon[column],128)
                ELSE BlockMove(defIcon^,@driveIcon[column],128);

{ Get the disk's icon or use our generic one if we can't get it }

                pb.csCode:=DskIconCode;
                IF PBControl(@pb,FALSE)=NoErr THEN
                    BlockMove(theHandle^,@diskIcon[column],128)
                ELSE
                    BlockMove(defIcon^,@diskIcon[column],128);

{ Find out the drive's type and location }

                pb.csCode:=DrvInfoCode;
                IF PBControl(@pb,FALSE)=NoErr THEN BEGIN
                    driveType:=LOWRD(BAND(LONGINT(theHandle^),$0000000F))+
                                    (Drive400K-2);
                    IF BTST(LONGINT(theHandle^),8) THEN          {internal or external drive}
                        I:=Ext1DrvStr
                    ELSE
                        I:=Int1DrvStr;
                    IF BTST(LONGINT(theHandle^),11) THEN
                        I:=I+1;{secondary drive}
                END
                ELSE
                IF column=0 THEN
                    I:=Int1DrvStr                                {older drivers may not
                                                        support this}
                ELSE
                    I:=(Ext1DrvStr-1)+column;                    { call, so just go by drive
                                                        number}

                GetIndString(theString,StrRsrcID,driveType);     {get the type's name}
                SetDText(BaseDrvTyp+column,theString);           { and display it}
                GetIndString(theString,StrRsrcID,I);             { get the drive location
                                                        string}
                SetDText(BaseDrvLoc+column,theString);           { and display it}
```

```
                END
                ELSE BEGIN                                          {if no drive is installed}
                    driveNum[column]:=0;                            { then set the drive# to
                                                                    zero}
                    HideDControl(Eject1Btn+column);                 {and make the control
                                                                    invisible}
                END;
            END;

            ShowWindow(demoDialog);                                 {let 'em see the window
                                                                    now}
        END;

{
_____________________________________________________________________

                                    M A I N
_____________________________________________________________________
}

    BEGIN
        Initialize;                                                 { initialize everything }

        REPEAT
            IF GetNextEvent(everyEvent,theEvent) THEN BEGIN         { handle an event}
                IF theEvent.what=diskEvt THEN
            { a disk was just inserted, so…}
                    NewDisk(LOWRD(theEvent.message));
            { find out about it}

                IF theEvent.what=mouseDown THEN
            { mouse click in the goAway box?}
                    IF FindWindow(theEvent.where,whichWindow)=inGoAway THEN
                        IF TrackGoAway(demoDialog,theEvent.where) THEN BEGIN
                            DisposDialog(demoDialog);               { get rid of our window}
                            EXIT(HDFDDriverDemo);                       { and return to the
Finder}
                        END;

                IF IsDialogEvent(theEvent) THEN
        {it's in our dialog window}
                    IF DialogSelect(theEvent,whichWindow,itemHit) THEN
```

```
                    IF itemHit IN [Eject1Btn..Eject3Btn] THEN BEGIN      {it's an Eject button, so…}
                        IF Eject(NIL,itemHit-(Eject1Btn-1))=0 THEN ;     { eject the disk,}
                        SetDText((BaseDskFmt-Eject1Btn)+itemHit,'');     { erase the disk format
                                                                                entry,}
                        HiliteDControl(itemHit,255);                     { and disable the eject
button}
                    END;
                END;
            UNTIL FALSE;
        END.
```

FDHD Driver Demo resources

Here are the resources required by the sample application program FDHD Driver Demo.

```
/*
_____________________________________________________________________

       Driver Demo Resource Source File

       25-Apr-88

       Copyright © 1988 by Apple Computer, Inc.
_____________________________________________________________________
*/

#include "Types.r";

/* "demo" dialog window containing drive and disk info */

resource 'DLOG' (256, preload) {
       { 40, 30,295,475}, documentProc, invisible, goAway, 0, 256, "Driver Demo"
};


/* "demo" dialog's item list */

resource 'DITL' (256, preload) {
       {
              {180,120,210,205}, button {enabled, "Eject"};     /* drive 1's eject button */
              {180,230,210,315}, button {enabled, "Eject"};     /* drive 2's eject button */
              {180,340,210,425}, button {enabled, "Eject"};     /* drive 3's eject button */
              { 10, 10, 26,110}, staticText {disabled, "Drive Location"};
              { 38, 10, 54,100}, staticText {disabled, "Drive Icon"};
              { 78, 10, 94,100}, staticText {disabled, "Disk Icon"};
              {110, 10,126,100}, staticText {disabled, "Drive #"};
              {130, 10,146,100}, staticText {disabled, "Drive Type"};
              {150, 10,166,100}, staticText {disabled, "Disk Format"};
              {230, 10,246,145}, staticText {disabled, "Disk Controller Chip:"};
              {230,150,246,190}, staticText {disabled, ""};     /* disk controller type */
              { 10,125, 26,200}, staticText {disabled, ""};     /* drive locations */
              { 10,235, 26,310}, staticText {disabled, ""};
              { 10,345, 26,420}, staticText {disabled, ""};
              { 30,145, 62,177}, userItem {disabled};           /* drive icons */
              { 30,255, 62,287}, userItem {disabled};
              { 30,365, 62,397}, userItem {disabled};
              { 70,145,102,177}, userItem {disabled};           /* disk icons */
              { 70,255,102,287}, userItem {disabled};
              { 70,365,102,397}, userItem {disabled};
              {110,155,126,167}, staticText {disabled, ""};     /* drive numbers */
              {110,265,126,277}, staticText {disabled, ""};
              {110,375,126,387}, staticText {disabled, ""};
              {130,120,146,210}, staticText {disabled, ""};     /* drive types */
              {130,230,146,320}, staticText {disabled, ""};
              {130,340,146,430}, staticText {disabled, ""};
              {150,135,166,190}, staticText {disabled, ""};     /* disk formats */
              {150,245,166,300}, staticText {disabled, ""};
              {150,355,166,410}, staticText {disabled, ""}
       }
```

```
};
/* descriptive strings */

resource 'STR#' (256, preload) {
       {
              "IWM";                                    /* disk controller types */
              "SWIM";
              "Internal 1";                             /* primary internal drive */
              "Internal 2";                             /* secondary internal drive */
              "External 1";                             /* primary external drive */
              "External 2";                             /* secondary external drive */
              "Single-Sided";                           /* drive types */
              "Double-Sided";
              "SuperDrive";
              "";                                       /* (fillers) */
              "";
              "Hard Disk 20"
       }
};


/* default drive and/or disk icon to use if we get an error trying to get an */
/* icon from the driver (probably because it's a REALLY old driver version)  */

resource 'ICON' (256, preload) {
       $"7FFFFF78"           /*   xxxxxxxxxxxxxxxxxxxxxx xxxx     */
       $"98FFFF84"           /*  x  xx   xxxxxxxxxxxxxxxx    x    */
       $"80FFC702"           /*  x       xxxxxxxxxx   xxx      x   */
       $"90FFC702"           /*  x  x    xxxxxxxxxx   xxx      x   */
       $"80FFC702"           /*  x       xxxxxxxxxx   xxx      x   */
       $"90FFC702"           /*  x  x    xxxxxxxxxx   xxx      x   */
       $"80FFC702"           /*  x       xxxxxxxxxx   xxx      x   */
       $"90FFC702"           /*  x  x    xxxxxxxxxx   xxx      x   */
       $"80FFC702"           /*  x       xxxxxxxxxx   xxx      x   */
       $"90FFFF02"           /*  x  x    xxxxxxxxxxxxxxxx      x   */
       $"8B7FFE02"           /*  x   x xx xxxxxxxxxxxxxx       x   */
       $"80000002"           /*  x                             x   */
       $"80000002"           /*  x                             x   */
       $"8AAAAAA2"           /*  x   x x x x x x x x x x x x   x   */
       $"90000012"           /*  x  x                       x  x   */
       $"80000002"           /*  x                             x   */
       $"90000012"           /*  x  x                       x  x   */
       $"80000002"           /*  x                             x   */
       $"90000012"           /*  x  x                       x  x   */
       $"80000002"           /*  x                             x   */
       $"90000012"           /*  x  x                       x  x   */
       $"80000002"           /*  x                             x   */
       $"90000012"           /*  x  x                       x  x   */
       $"80000002"           /*  x                             x   */
       $"90000012"           /*  x  x                       x  x   */
       $"E0000002"           /*  xxx                           x   */
       $"F0000012"           /*  xxxx                       x  x   */
       $"80000002"           /*  x                             x   */
       $"90000012"           /*  x  x                       x  x   */
       $"7FFFFFFC"           /*   xxxxxxxxxxxxxxxxxxxxxxxxxxxx   */
};
```

Sound Manager

The Sound Manager for Macintosh Portable is the same as that documented for the Macintosh II in *Inside Macintosh*, Volume V, and supplemented by any applicable Technical Notes. The Sound Manager has incorporated the functions of the Sound Driver.

Modem

Support for an internal modem is provided. See Chapter 6, "The Power Manager," and Chapter 9, "Options."

Sleep State and Operating State

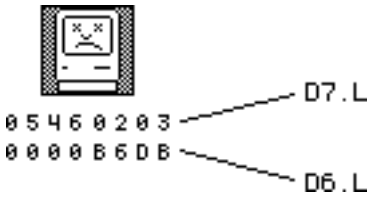
The Macintosh Portable ROM software supports the ability to put the computer into the sleep state (clock to DC, all RAM and registers retained) and to bring it back to the operating state. These functions are implemented in the power manager firmware and the power manager processor. The OS requests the sleep state through a time-out scheme or direct user action. Return to the operating state (waking) is due to an event such as a keystroke or wake-up timer going off. See Chapter 6, "The Power Manager."

RAM and ROM Expansion

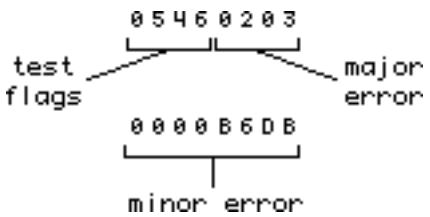
Memory expansion is done using internal RAM expansion cards in the machine and is supported by the ROM. ROM expansion/replacement is likewise available by using an internal expansion connector (slot) to which are brought the necessary signals (see Chapter 5, "Hardware", for an explanation). The 4 MB of ROM address space between $A0 0000 and $DF FFFF is available to you See the address map, Figure 3-1. Refer to Macintosh Technical Note #255, "Macintosh Portable ROM Expansion," for additional information.

Diagnostics—The “sad Macintosh” icon

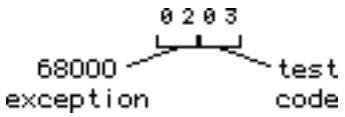
The bootup code in the Macintosh contains a series of startup tests that are run to insure that the fundamental operations of the machine are working properly. If any of those tests fail, a “sad Macintosh” icon appears on the screen with a code below that describes what failure occurred. Here is a typical example of a “sad Macintosh” display with an error code below it:



The two codes are actually the contents of the two CPU data registers D6 and D7. The upper word (upper 4 hex digits, in this case 0546) of D7 contains miscellaneous flags that are used by the start-up test routines and are unimportant to just about everybody except a few test engineers within Apple. The lower word of D7 is the major error code. The major error code identifies the general area the test routines were in when a failure occurred. D6 is the minor error and usually contains additional information about the failure, something like a failed bit mask.



The major error is further broken into the upper byte that contains the number of any 68000 exception that occurred ($00 meaning that no exception occurred), and the lower byte that usually contains the test that was being run at the time of failure. If an unexpected exception occurred during a particular test, then the exception number is logically ORed into the major error code. This way both the exception that occurred as well as the test that was running can be decoded from the major error code:



In this example, the code says that an address error exception ($0200) occurred during the RAM test for Bank A ($03); $0200 ORed with $03 = $0203.

Major error codes

Below is a brief description of the various test codes that might appear in the major error code:

♦ **Warning** Some of these codes may mean slightly different things in Macintosh models other than the Macintosh Portable. These descriptions describe specifically how they are used in the Macintosh Portable. ♦

$01 - ROM test failed. Minor error code is $FFFF, means nothing.

$02 - RAM test failed. Minor error code indicates which RAM bits failed.

$05 - RAM external addressing test failed. Minor error code indicates a failed address line.

$06 - Unable to properly access the VIA 1 chip during VIA initialization. Minor error code not applicable.

$08 - Data bus test at location 8 bytes off of top of memory failed. Minor error code indicates the bad bits as a 16–bit mask for bits 15–00. This may indicate either a bad RAM chip or data bus failure.

$0B - Unable to properly access the SCSI chip. Minor error code not applicable.

$0C - Unable to properly access the IWM (or SWIM) chip. Minor error code not applicable.

$0D - Not applicable to Macintosh Portable. Unable to properly access the SCC chip. Minor error code not applicable.

$0E - Data bus test at location $0 failed. Minor error code indicates the bad bits as a 16–bit mask for bits 15–00. This may indicate either a bad RAM chip or data bus failure.

$10 - ***Macintosh Portable only.*** Video RAM test failed. Minor error code indicates which RAM bits failed.

$11 - ***Macintosh Portable only.*** Video RAM addressing test failed. Minor error code contains the following:

upper word = failed address (16-bit)
msb of lower word = data written
lsb of lower word = data read

Data value written also indicates which address line is being actively tested.

$12 - ***Macintosh Portable only.*** Deleted

$13 - ***Macintosh Portable only.*** Deleted

$14 - ***Macintosh Portable only.*** Power manager processor was unable to turn on all the power to the board. This may have been due to a communication problem with the power manager. If so, the minor error code will contain a power manager error code, explained in the next section.

$15 - ***Macintosh Portable only.*** Power manager failed its self-test. Minor error code contains the following:
msw = error status of transmission to power manager (see "Power manager processor failures (Macintosh Portable only)".
lsw = power manager self-test results (0 means it passed, non-zero means it failed)

$16 - ***Macintosh Portable only***. A failure occurred while trying to size and configure the RAM. Minor error code not applicable.

Minor error codes—Power manager processor failures (Macintosh Portable only)

If a communication problem occurs during communication with the power manager, the following error codes will appear somewhere in the minor error code (usually in the lower half of the code, but not always):

$CD38 Power manager was never ready to start handshake.
$CD37 Timed out waiting for reply to initial handshake.
$CD36 During a send, power manager did not start a handshake.
$CD35 During a send, power manager did not finish a handshake.
$CD34 During a receive, power manager did not start a handshake.
$CD33 During a receive, power manager did not finish a handshake.

Diagnostic Code Summary

Below is a summarized version of the sad Macintosh error codes:

Test Codes

$01	ROM checksum test.
$02	RAM test.
$05	RAM addressing test.
$06	VIA 1 chip access.
$08	Data bus test at top of memory.
$0B	SCSI chip access.
$0C	IWM (or SWIM) chip access.
$0D	Not applicable to Macintosh Portable. SCC chip access.
$0E	Data bus test at location $0.
$10	Macintosh Portable only. Video RAM test.
$11	Macintosh Portable only. Video RAM addressing test.
$14	Macintosh Portable only. Power manager board power on.
$15	Macintosh Portable only. Power manager self-test.
$16	Macintosh Portable only. RAM sizing.

Power manager communication error codes

$CD38	Initial handshake.
$CD37	No reply to initial handshake.
$CD36	During send, no start of a handshake.
$CD35	During a send, no finish of a handshake.
$CD34	During a receive, no start of a handshake.
$CD33	During a receive, no finish of a handshake.

CPU exception codes (as used by the startup tests)

$0100	Bus error exception code
$0200	Address error exception code
$0300	Illegal error exception code
$0400	Zero divide error exception code
$0500	Check inst error exception code
$0600	cpTrapcc,Trapcc,TrapV exception code
$0700	Privilege violation exception code
$0800	Trace exception code
$0900	Line A exception code
$0A00	Line F exception code
$0B00	Unassigned exception code
$0C00	CP protocol violation
$0D00	Format exception
$0E00	Spurious interrrupt exception code
$0F00	Trap inst exception code
$1000	Interrupt level 1
$1100	Interrupt level 2
$1200	Interrupt level 3
$1300	Interrupt level 4
$1400	Interrupt level 5
$1500	Interrupt level 6
$1600	Interrupt level 7

Script Manager

The Script Manager is part of the ROM image.

Notification Manager

The Notification Manager is part of the ROM image. See Macintosh Technical Note #184, April 2, 1988.

Chapter 4 **System Software**

This chapter describes the system software portion of the total software environment for the Macintosh Portable computer. The total software environment includes both ROM-stored code (firmware) and disk-stored code (system software). This chapter describes the contents of the system tools disk. See also Chapter 2, "Software Developer Guidelines." •

4.1 Overview

These notes primarily describe the *changes* from the previous version of the system tools disk.

The Macintosh SE software is extensively documented in *Inside Macintosh,* Volume V and the *Macintosh Technical Notes*. The contents of Volume V that apply to the Macintosh SE describe its software in terms of changes from the Macintosh Plus, documented in *Inside Macintosh*, Volume IV. Volume IV, in turn, describes changes from the classic Macintosh as documented in *Inside Macintosh*, Volumes I, II, and III.

Terminology

The Macintosh Portable software comes in two components:

- Firmware—contents of the three ROMs, one for each of the three processors (68000 CPU, power manager processor, and the keyboard processor). See Chapter 3, “Firmware”.
- System software—contents of an 800 KB, 3.5" disk, Version 6.0.3, plus the Macintosh Portable-specific extensions. The disk containing the system software is called the *system tools disk* (also commonly referred to as the *system disk*).

System tools software conversion

The Macintosh system software first seeded will be Version 6.0.3 plus Macintosh Portable-specific extensions, all contained on one system tools disk. This chapter describes the Macintosh Portable-specific elements. The description of Version 6.0 and the change history to Version 6.0.3 are provided in other documents that are a part of this seeding package.

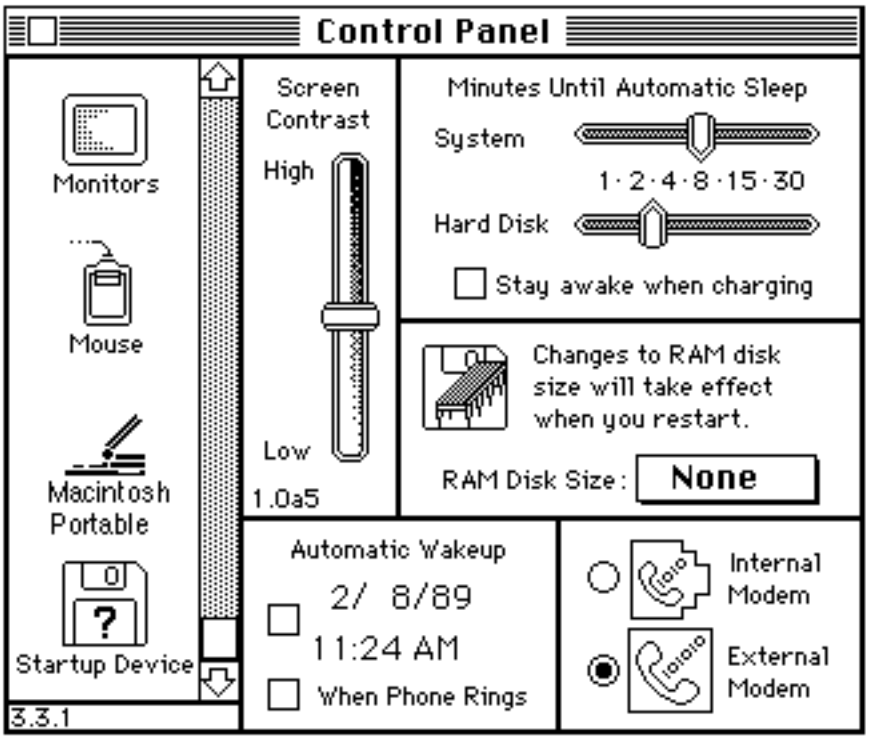
The system tools disk to be provided with customer shipments will be part of the standard Macintosh system software kit applicable to all members of the Macintosh family of computers. The Macintosh Portable installation script included will allow installation of the appropriate software onto the users' floppy or hard disk.

4.2 The Macintosh Portable control panel cdev resource

This cdev will generate the screen display elements shown in Figure 4-1. The display elements are

- a Macintosh Portable icon, the opening of which produces the window shown
- a screen contrast control slide
- a pair of sliding controls for adjusting the time to automatic sleep (one for the hard disk and the other for the remaining power manager controlled subsystems)
- a checkbox for disabling the automatic sleep functions while the battery recharger is connected
- a control to select RAM disk size
- a means of setting the time for automatic wakeup from the sleep state
- a set of two buttons to control modem operation if a modem card is present in the machine

- **Figure 4-1** Control Panel

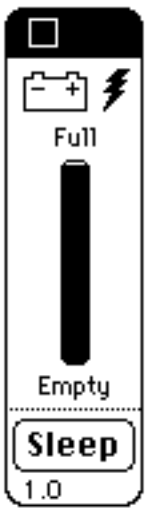


4.3 The Macintosh Portable battery desk accessory

This desk accessory will generate the screen display elements shown in Figure 4-2. The display elements (accessible from the Apple menu) are

- an indicator to show the battery charge level
- an icon to indicate whether battery recharging is in progress
- a button to toggle the sleep state on and off

- **Figure 4-2** Battery desk accessory



4.4 Macintosh Portable battery monitor

This resource is called by the one-second interrupt and has the following functions:

- Monitors the state of battery charge and applies the criterion for system shutdown to avoid battery damage.
- Defines and applies the criteria of inactivity to cause the CPU to instruct the power manager to put the computer in the idle or sleep states.
- Monitors the sound chip usage, looking for opportunities to turn off the sound circuit. Any use of the ASC automatically enables all the sound circuitry with its heavy current load. The monitoring code looks for 10 seconds without any ASC accesses; when such an interval is found the sound circuit is turned off.

Activities that prevent sleep (and also idle) are

- Any operating system Read or Write call
- A call to a post event trap
- A call to set a cursor trap that changes the cursor
- Executing an ADB completion routine
- Having a trackball or mouse button down (pressed)

See Chapter 6, "*The Power Manager*," for a more detailed description of the functions of the battery monitor resource.

Chapter 5 **Hardware**

This chapter describes the architecture and functional elements of the Macintosh Portable portable computer. From the user's point of view the Macintosh Portable operates much like a Macintosh SE. However, the Macintosh Portable hardware is different in significant ways from that of the Macintosh SE, reflecting the differences in operating and transportation environments and the power source.

These notes primarily describe the changes from the Macintosh SE hardware. The Macintosh SE hardware is extensively documented in the *Guide to Macintosh Family Hardware, Second Edition* manual. •

This page is a blank

5.1 The Macintosh Portable Specifications

The Macintosh Portable contains the Motorola MC68HC000 microprocessor (CPU) operating at nearly 16 megahertz in a multiprocessing environment with two other processors:

1) a keyboard scanning and decoding microprocessor
2) a power management microprocessor

The Macintosh Portable is equipped with all the standard Macintosh SE architectural features, namely the VIA, SCC, SWIM, and SCSI chips. Sound is generated by the same sound circuitry as in the Macintosh II (Apple Sound Chip and dual Sony Sound Chips). Video is generated by a separate circuit and memory that drives the input to a flat-panel display.

The Macintosh Portable is also a loosely coupled multiprocessor. The power manager is implemented with an 8-bit CMOS microprocessor that not only handles power management duties, but is also responsible for the interface between the built-in trackball, a low-power mouse or other input device, and the Apple Desktop Bus. The interface between the power manager and the 68HC000 is implemented using one of the 8-bit bi-directional ports of the VIA and two asynchronous handshake lines.

Table 5-1 repeats the Macintosh Portable specifications given in Chapter 1. Table 5-2, in the next section, compares the hardware features of the Macintosh Portable with the Macintosh SE.

• **Table 5-1** The Macintosh Portable Specifications

Characteristic	Specification
CENTRAL PROCESSING UNIT (CPU):	16-bit, CMOS 68HC000, 16 MHz (twice SE speed and without video contention), 1 wait state
OPERATING SYSTEM (OS):	Enhanced Macintosh SE ROM
STANDARD MAIN MEMORY:	1MB RAM, 256 KB ROM
MEMORY EXPANSION:	Main memory (RAM) is expandable to 2 or 5 MB by using internal expansion cards (1 or 4 MB). ROM expansion space for Apple use in ROM revision and for international character sets is 1 MB, and for developer use is 4 MB (see Chapter 3, Figure 3-1.)
MASS STORAGE:	Built-in 1.4 MB floppy disk drive External floppy disk drive port Removable/optional second internal 1.4 MB floppy disk drive Optional internal low power SCSI hard disk, also external SCSI port
RAM DISK:	Ability to install system and application software in battery-backed-up RAM so that the machine is capable of functioning without resorting to the optional disk drive. This feature will provide a more rugged, lighter weight solution to portable applications, with longer battery operation and much faster access.
DISPLAY:	Flat-panel, 9.8" diagonal, active matrix reflective LCD, 640 x 400 pixels, 0.33 mm dot pitch, variable tilt
SOUND:	Apple stereo sound chip (same as Macintosh II)

• **Table 5-1** The Macintosh Portable Specifications (Continued)

Characteristic	Specification
I/O PORTS:	DB-19 external floppy disk DB-25 SCSI Mini DIN-4 Apple Desktop Bus port Two Mini DIN-8 serial ports DB-15 for external video 96-pin Euro-DIN expansion interface (not compatible with Macintosh SE) Stereo audio phone jack Battery recharger
INPUT DEVICES:	Built-in keyboard. Built-in trackball replaceable by optional keypad. (The keypad or trackball may be positioned on either side of the keyboard—right side is standard.) Low-power Apple Desktop Bus mouse (optional) plugs into ADB port at the rear of the machine.
OPTIONAL INTERNAL MODEM:	300/1200/2400 bps (AT command set compatible)
WEIGHT:	14 lbs. (minimum configuration) up to 17 lbs. (hard disk option, 5 MB RAM, internal modem option)
SIZE:	15.2" wide x 13.75" deep x 2" to 4" thick (wedge shaped)
SHOCK:	The unit can withstand a 50 G, 12 millisecond shock pulse in any axis while non-operating. This is true of all configurations, for example, with or without a hard disk.
BATTERY USE:	Internal, sealed lead-acid battery provides 8 hours normal use (single floppy configuration), varies dependent on drive usage. Rechargeable overnight using AC power adapter RAM contents are retained during main battery replacement

5.2 Comparison of the Macintosh Portable and the Macintosh SE

This section calls out the significant differences between the Macintosh Portable and the Macintosh SE.

Improvements

1. Portable: smaller size, lighter weight, greater shock and vibration resistance
2. Larger RAM expansion to allow larger application programs
3. Higher speed due to faster clock, removal of RAM contention for video
4. Floppy and hard disk drives are low profile, one-third height configuration
5. Low power hard disk with an internal connector to SCSI interface

Variations

1. The 96-pin expansion connector does not have the same pinout as in the Macintosh SE, but the same signals are accessable.
2. Macintosh SE expansion cards will not physically fit in the Macintosh Portable.
3. A *very limited* amount of internal battery power is available to supply any expansion card inserted into the 96-pin expansion connector.
4. The Macintosh Portable does not provide an external device access port for a custom connector.
5. The Macintosh Portable contains only one ADB port and it supports low-power input devices (preferred) or normal Macintosh input devices at some battery performance penalty.
6. A trackball replaces the numeric keypad in the basic configuration of the Macintosh Portable; the keypad is a user installable option.

7. The Macintosh Portable does not provide the termination power to the SCSI bus, and hence to any external SCSI devices.

Table 5-2 is a side-by-side comparison of the hardware features of the Macintosh Portable and the Macintosh SE.

- **Table 5-2** Macintosh Portable vs. Macintosh SE Hardware Comparison

Feature	Riviera	Macintosh SE
Processor:	CMOS 68HC000 CPU	68000 CPU
Clock Frequency:	15.6672 MHz	7.8336 MHz
Auxiliary Processor:	Power Manager Processor	None
Floppy Disk Drive:	1.4 MB Internal Floppy Drive, Optional Second Internal	800 KB Internal Floppy Drive, Optional 2nd
External 800 KB	1.4 MB Floppy Drive, Optional External 800 KB or 1.4 MB Floppy Drive	800 KB Internal Drive, Optional Floppy Drive
Hi Speed Periph.:	SCSI Port	SCSI Port
Hard Disk:	Optional low-power SCSI HD40 Internal	Optional SCSI HD20 (Internal)
connector	External SCSI connector.	External SCSI
Serial Ports:	2 Mini-8 Built-In Ports,with extended input handshake capability	2 Mini-8 Built-In Ports, with extended input handshake capability
Hardware Expansion:	Access to 68000 pins No power available, No removable door at rear	Access to 68000 pins, Customizable I/O Port in removable door at rear
Sound:	Macintosh II Sound, Stereo	Macintosh Sound
RAM :	1MB, Expandable on Internal SRAM card- 1 MB, or 4 MB expansion	1 MB Expandable to 4 MB DRAM (SIMMs)
ROM :	256 KB ROM with Hierarchical File System, ROM supports SCSI, ADB, AppleTalk, Power Manager	256 KB ROM with Hierarchical File System, ROM supports SCSI, ADB, AppleTalk
Operator Input:	Alphanumeric Keyboard, optional keypad or pointing device	Alphanumeric Keyboard via Apple Desktop Bus, Allows

	(left or right side) via ADB, Allows additional input devices, e.g. graphics tablet.	additional input devices, tablet

- **Table 5-2** Macintosh Portable vs. Macintosh SE Hardware Comparison (Continued)

Feature	Riviera	Macintosh SE
Video Display	Built-In LCD, Flat Panel Display, 9.8", 640 x 400 B/W 75 dots per inch	Built-In Monitor, 9" 512 x 342 B/W, 72 dots per inch

5.3 Block diagrams of the Macintosh Portable and Macintosh SE

This section provides a functional overview of the Macintosh Portable architecture. It gives block diagrams of both the Macintosh Portable and the Macintosh SE, then goes on to briefly describe the function of each block in the Macintosh Portable diagram.

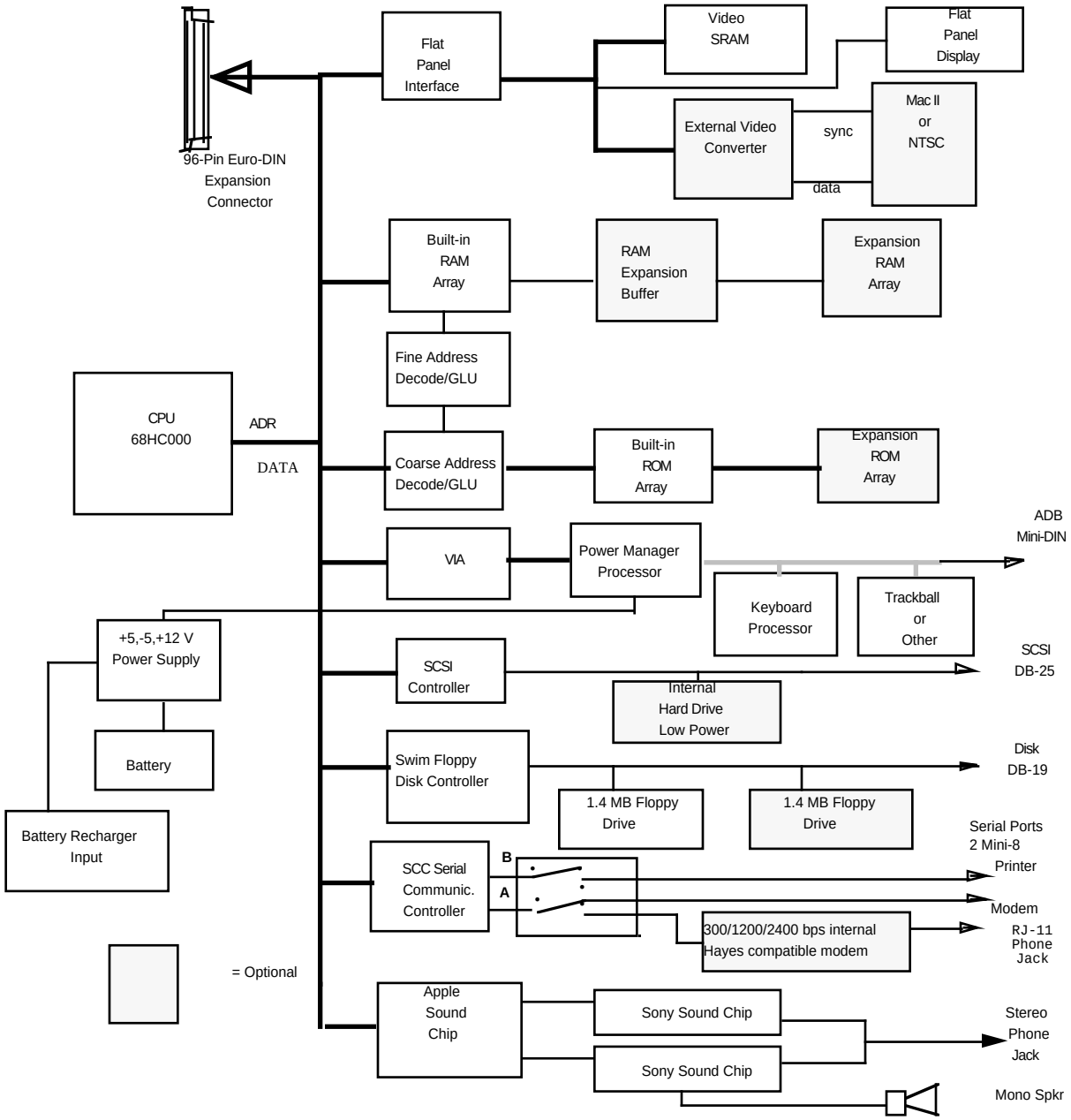
Figure 5-1 is a block diagram of the Macintosh Portable. Figure 5-2 is a block diagram of the Macintosh SE, for comparison. Most of the unshaded elements in Figure 5-1 are located on the main logic board of the Macintosh Portable. The 68000 central processor communicates over the system bus (address and data), indicated by the heavy black line. Also connected to the system bus are

- random access memory (RAM)
- read-only memory (ROM)
- an Apple custom VLSI chip that performs coarse address decoding and generalized logic unit (GLU) functions
- six additional interface and controller chips.

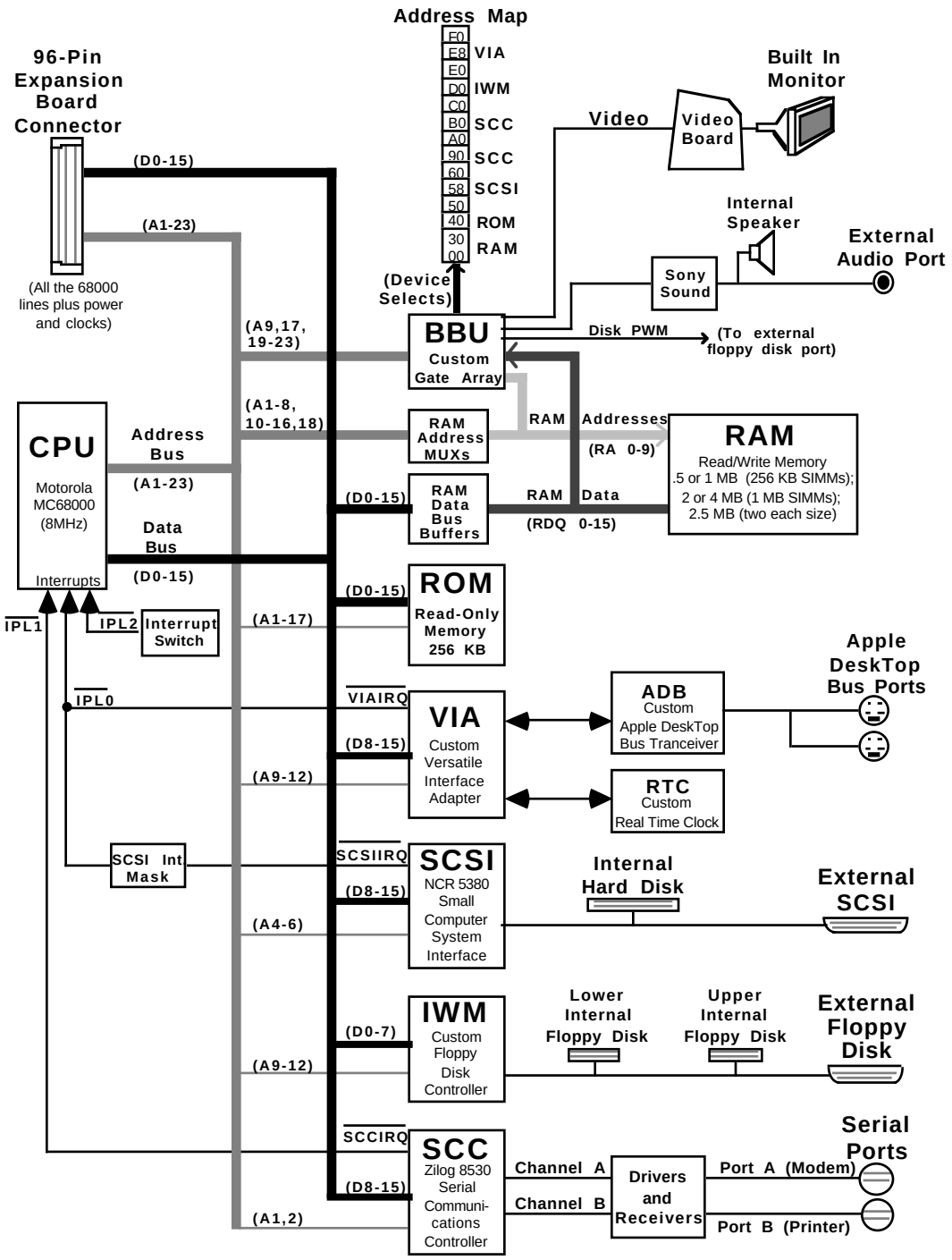
Each of the devices connected to the system bus is accessed (written to or read from) through a range of addresses in memory mapped address space. The address space is diagrammed in Chapter 3, “Firmware.”

The rest of the sections in this chapter describe the functions of the blocks in the Macintosh Portable diagram.

- **Figure 5-1** The Macintosh Portable Architecture



• **Figure 5-2** Macintosh SE Architecture



5.3 The central processing unit (CPU)

The Macintosh Portable replaces the standard Macintosh 8 MHz HMOS MC68000 with a CMOS MC68HC000 for reduced power consumption and higher speed.

The 68HC000 has a 16-bit data bus and a 24-bit address bus.

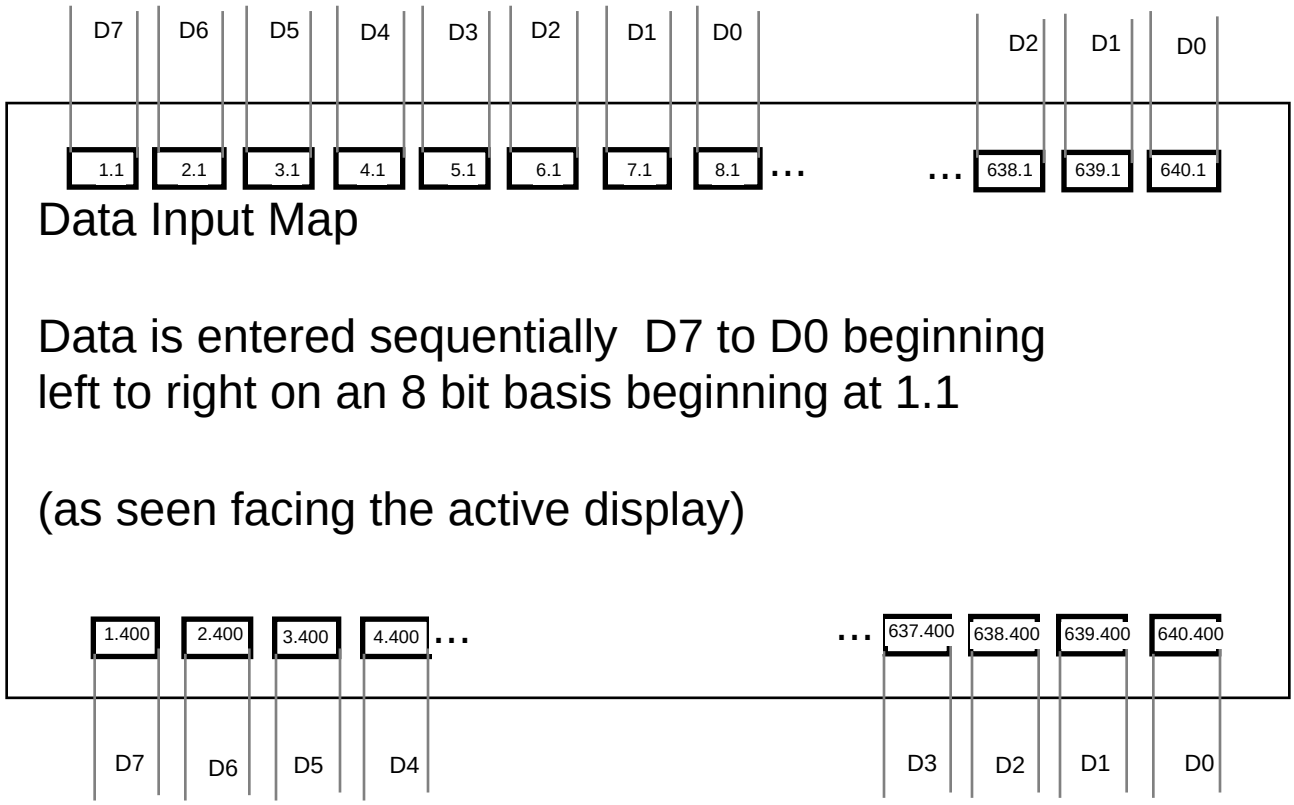
The frequency at which the Macintosh Portable system performs useful work is 15.667 MHz. However, to reduce power consumption during idle periods the Macintosh Portable is designed with a two state variable wait state system that effectively makes the clock frequency either approximately 1 MHz or 15.667 MHz. (See Chapter 6, "The Power Manager.")

5.4 Video Display Interface chip

The Video Display Interface chip is a custom IC designed to keep the Macintosh Portable 68HC000 CPU from having to refresh the screen (LCD display), thereby allowing the CPU to do more useful work. It also generates all the signals necessary for the flat-panel display, including the vertical and horizontal synchronization pulses. The difference between the Macintosh Portable and the Macintosh SE due to the function of this chip will have no effect on compatibility of your software.

From the point of view of the 68HC000, the video interface is seen as a continuous RAM array of 32,000 bytes (768 bytes reserved for later use). The video controller interface is nominally 16 bits wide from the 68HC000 to the Video Display Interface chip, but behaves like main memory in that it is also byte addressable. The first pixel displayed on the screen is the most significant bit of the first byte of video RAM (at Screen_base), while the last pixel displayed on the screen is the least significant bit of the last byte (at Screen_base+32,000–1). Figure 5-3 shows the entry order of data in terms of the pixels on the LCD. The pixels that are displayed between the first and last are addressed in a similar fashion; the display can be thought of as a linear array of bits.

- **Figure 5-3** Display Pixel Map



Flat panel display description

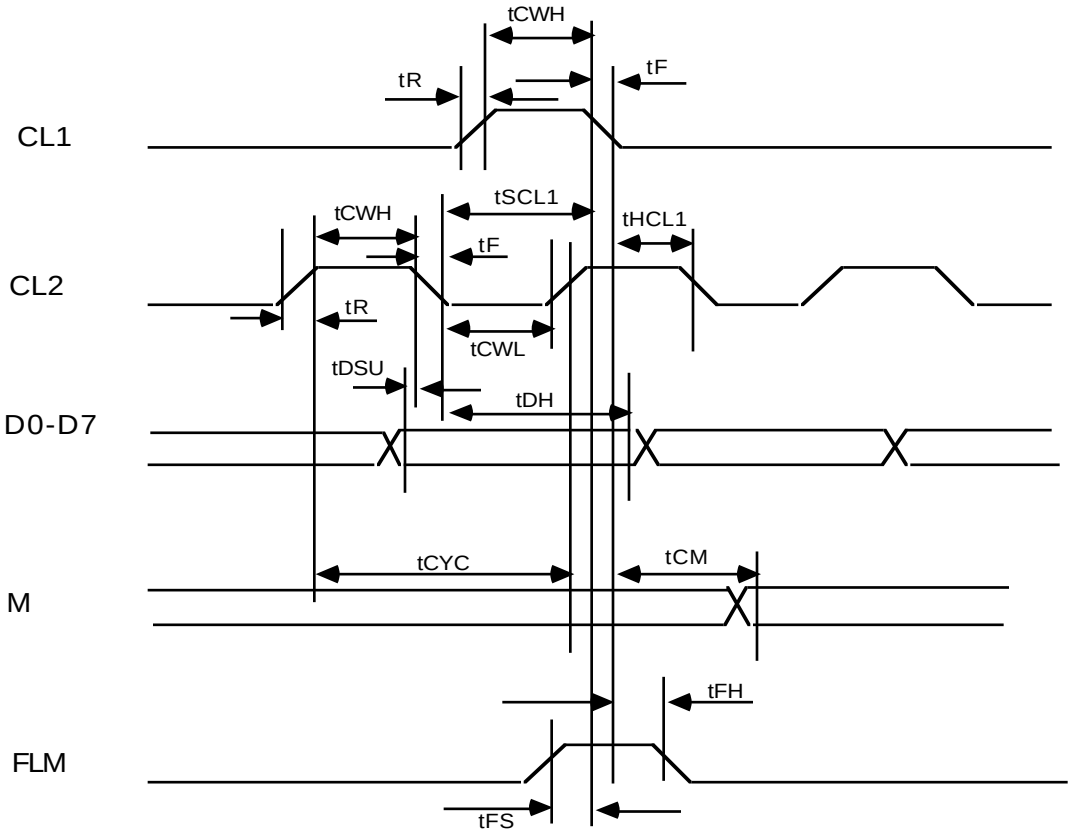
The display gives a high quality presentation of alphanumeric and graphic information on a 211 mm x 132 mm (8.31 in x 5.20 in) active display area. Display intensity, contrast ratio, and pixel turn on and turn off time are similar to CRT parameters (P4 phosphor).

Display Capacity	640 x 400 dots
Display Mode	Reflective
Driving Method	8-bit Parallel
Interface	CMOS Logic

Video signal timing

The Video Display Interfacechip generates the synchronization signals (FLM, CL1, CL2) and data signals as shown in Figure 5-4. The signal M is generated by the chip and is derived by dividing the FLM frequency by two. CL1 is the horizontal synchronization signal; it marks the end of a 640 pixel line. FLM is the vertical synchronization signal; it marks the beginning of a new frame of video every 16.32 milliseconds. This means that the display is being refreshed at a 61.8 Hz rate.

- **Figure 5-4** Video Interface Timing Diagram



SYMBOL	ITEM	MIN	MAX
tCYC	CL2 cycle time	190 ns	
tCWH	CL2 pulse width (high)	95 ns	
tCWL	CL2 pulse width (low)	95ns	
tSCL1	CL1 setup time	90 ns	
tHCL1	CL1 hold time	90 ns	
tR, tF	clock rise/fall time		30 ns
tDSU	Data setup time	60 ns	
tDH	Data hold time	60 ns	
tCM	M delay time	-500ns	+500ns
tFS	FLM setup time	100 ns	
tFH	FLM hold time	100 ns	

Contrast control

The contrast control for the flat panel display is derived from the PWM (pulse width modulated) output of the power manager processor. The modulation has 256 steps, which the user can set in the Low to High range of the Screen Contrast control in the Control Panel (see Figure 4-1).

The PWM signal is filtered to a DC signal by an RC network before being applied to the display to control its contrast.

This page is a blank

5.6 Permanent RAM array

The RAM interface in the Macintosh Portable is designed to support between 1 MB and
5 MB of CMOS static RAM. The RAM is in two areas:

1) 1 MB permanent main memory soldered to the main logic board
2) Up to 4 MB of internally expandable RAM, on an optional memory expansion card

Permanent main memory is 1 MB total, and is arranged as a 512K x 16 bit array. This RAM array is located in the system memory map between addresses $00 0000 and $0F FFFF, and is overlayed by the system ROM after a system reset and before the first ROM access.

There is one 68HC000 processor wait state when accessing memory locations in permanent RAM. There is no device contention for permanent memory bandwidth other than the 68HC000, and because this memory array is built from static RAM there is no reason to refresh it, as would be the case for dynamic RAM.

Permanent main memory is battery backed-up when the Macintosh Portable is in the sleep state. This means that the contents of this memory array are retained when the computer is not in use, as long as the battery remains charged.

5.7 Permanent ROM array

The Macintosh Portable ROM is largely based on the Macintosh SE ROM, and includes bug fixes from the Macintosh SE ROM. It also includes a new ADB implementation, a new real-time clock implementation, code to support communication with the power manager, and code to support the various power-saving techniques that the Macintosh Portable incorporates.

The ROM interface on the Macintosh Portable is designed to support a minimum of 256 KB and an additional maximum of 4 MB of CMOS ROM. The ROM configuration is in two areas:

- 256 KB built-in, or permanent, ROM soldered to the printed circuit board
- Up to 4 MB of expansion ROM, using an internal connector. Expansion ROM address space is for you–the developer. See Macintosh Technical Note #255 for further details. (Also, see the "Internal ROM Expansion" section, later in this chapter).

Permanent ROM is 256 KB total, and is arranged as a 128K x 16 bit array. The array is physically made of two 128K x 8-bit devices.

This ROM array is word addressable and is located in the system memory map between addresses $90 0000 and $93 FFFF. Immediately after system reset, however, its starting address is located at both $90 0000 and $00 0000, to allow the 68HC000 to access a standard default set of exception vectors and trap addresses, as well as a starting address to begin executing code. This process is known as *RAM overlay* and is performed because it can be assumed that the contents of RAM are not in any known or deterministic state. The first access to $90 0000 (or any actual ROM addresses), however, will return RAM to $00 0000 and ROM will be located only at $90 0000.

There are two 68HC000 processor wait states when accessing memory locations in permanent ROM.

5.8 Memory Expansion

The growth of application program size makes it desirable to offer the RAM expansion cards described in this section, as well as others.

The Apple ROM expansion card has two functions which lead to its being called an upgrade and expansion card

- one side of the card contains a ROM to replace the one soldered to the main logic board, in case a code upgrade is necessary
- the other side of the card contains provisions for you, the developer, inserting ROM chips to utilize the address space shown as available in Figure 3-1

The RAM and ROM expansion connectors provide you with convenient access to the necessary signal lines in order to expand system memory.

Internal RAM Expansion

Internal RAM expansion is available through a single 50-pin connector (also called a slot). See Figure 5-5. All of the appropriate signals (address bus, data bus, and control) are brought up to the memory expansion board where they are decoded into chip selects, write signals, etc., by the RAM Expansion Buffer custom IC and routed to address and data buffers. Table 5-3 provides signal names and descriptions. Buffering the address and data bus is important to reduce capacitive loading.

The memory expansion board contains either 1 MB or 4 MB of static RAM, which allows for internal memory expansion to either a 2 MB machine or a 4 MB machine. Each expansion board is self-configured: no modifications (switches, jumpers, or other) to the main logic board are necessary to change the RAM configuration.

The 1 MB expansion card is arranged as a 512K x 16 bit array. The access time and cycle time for these devices is 100 ns.

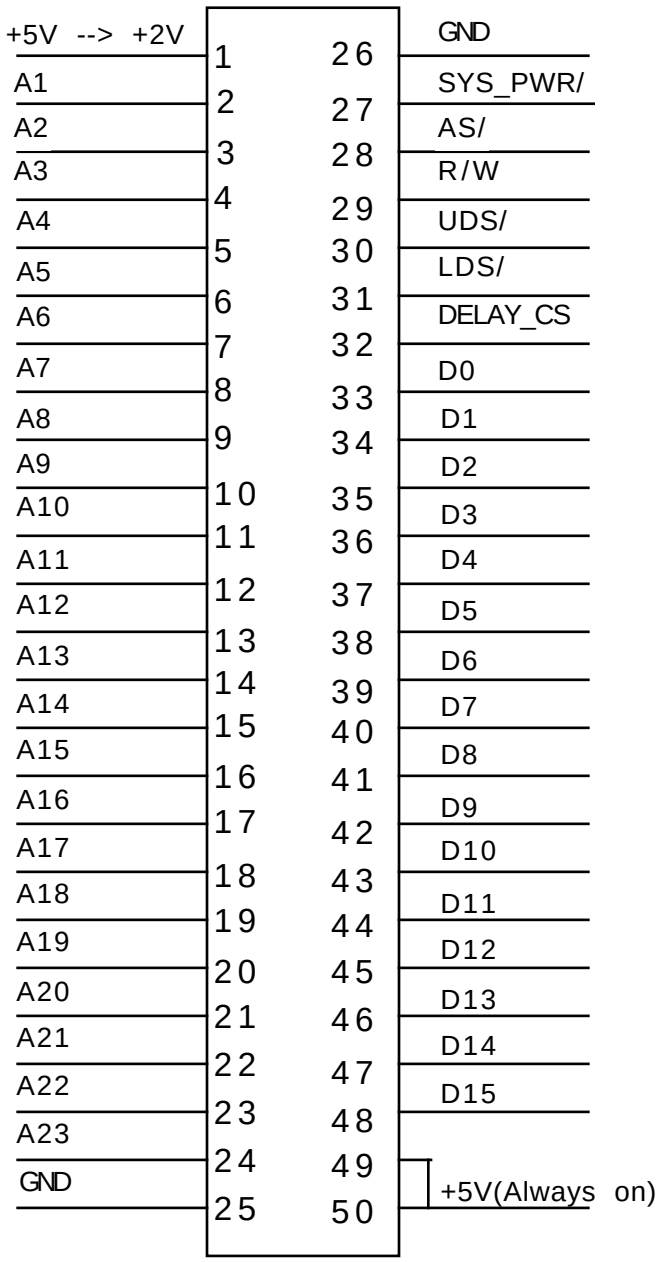
The 4`MB expansion card is arranged as a 1.5 M x 16 bit array. The access time and cycle time for these devices is 100 ns.

This RAM array is located in the system memory map between addresses
$10 0000 and $1F FFFF (for the 1 MB expansion card) and between $10 0000 and $3F FFFF (for the 3 MB expansion card). When installed, this memory array is always available and is unaffected by the state of the overlay bit (unlike permanent main memory).

There is one 68HC000 processor wait state when accessing memory locations in internal expansion RAM. Such an access requires a bus cycle time of nominally 320 ns. Like permanent main memory, there is no device contention for bandwidth other than the 68HC000, and because this memory array is built from static RAM there is no reason to refresh it, as would be the case for dynamic RAM.

Also like permanent main memory, internal expansion memory is battery backed-up when the Macintosh Portable is in the sleep state. This means that the contents of this memory array are retained when the computer is not in use, as long as the battery remains charged.

- **Figure 5-5** Internal RAM expansion connector



- **Table 5-3** Internal RAM Expansion Connector Signals

Pin Number	Name	Description
1	+5V	Vcc
2–24	A1–23	Unbuffered 68HC000 address signals A1–23
25–26	GND	Logic ground
27	/SYS_PWR	This signal controls whether the CPU is in the active or sleep state.
28	/AS	68HC000 address strobe signal
29	R/W	Permanent ROM CS/ signal
30	/UDS	16 MHz system clock
31 the Coarse	/LDS	External DTACK/ signal that is an input to Address Decode chip
32	DELAY_CS	This signal is generated by the Coarse Address Decode chip and is used to put the RAM array into the idle mode
33–48	D0–15	68HC000 unbuffered data signals D0:15
49–50	+5V	Vcc

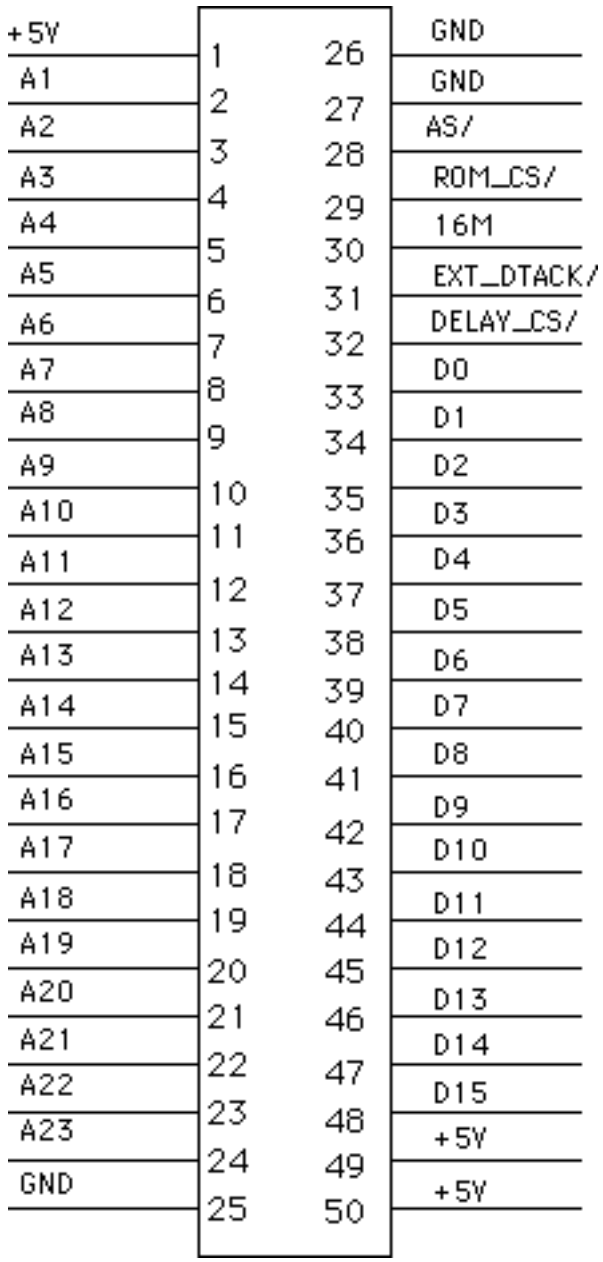
Internal ROM Expansion

Up to 4 MB of expansion ROM address space is available. The expansion ROM array is expandable to any size or capacity that fits into the expansion address space. The implementation of expansion ROM is very similar to that of internal expansion RAM (i.e., circuitry for decoding, control, and buffering must be part of the expansion board) except that the number of wait states is controlled by the expansion board (instead of the Coarse Address Decode and GLU chip, as is the case for RAM) via an external DTACK signal that is an output of the ROM expansion card.

ROM expansion is available through a single 50-pin connector (slot). See Figure 5-6. All of the appropriate signals (address bus, data bus, and control) are brought up to the memory expansion slot where they are decoded into chip selects, and also routed to address and data buffers. Table 5-4 provides signal names and descriptions. Buffering the address and data bus is important to reduce capacitive loading.

Replacing or upgrading the permanent ROM soldered to the main logic board is possible by using this same mechanism. By installing a board into the ROM expansion slot, and using the same /ROM CS signal that controls permanent ROM, a virtual replacement for the on-board ROM can be implemented.

• **Figure 5-6** Internal ROM Expansion Connector



- **Table 5-4** Internal ROM Expansion Connector Signals

Pin Number	Signal Name	Signal Description
1	+5V	Vcc
2–24	A1–23	Unbuffered 68HC000 address signals A1–23
25–27	GND	Logic Ground
28	/AS	68HC000 address strobe signal
29	/ROM.CS	Permanent ROM chip select signal
30	16M	16 MHz system clock
31	/EXT.DTACK	External /DTACK signal that is an input to Coarse Address Decode chip
32	DELAY.CS	This signal is generated by Coarse Address Decode chip and is used to put the RAM array into the idle mode
33–48	D0–15	68HC000 unbuffered data signals D0–15
49–50	+5V	Vcc

ROM expansion jumper on the main logic board

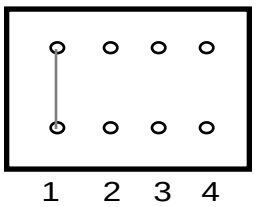
Figure 5-7 shows a detailed view of the jumper block on the Macintosh Portable main logic board. Jumpers are required in positions 1 or 2, and in position 3 when ROM on an expansion card is to replace the ROM soldered on the main logic board. Jumper 4 is used when an alternative power manager processor chip is mounted on a card inserted into the ROM expansion slot.

Figure 5-7 Internal ROM Expansion Jumper (see also Figure 1-2)

Jumper(s) Inserted
In Position Number

1) 2 wait-state ROM access
or
2) 1 wait-state ROM access

3) system ROM on expansion card
4) on-board power manager processor access



5.9 Coarse Address Decode and GLU

The Coarse Address Decode and GLU chip is a custom gate array designed as a generalized logic chip for the Macintosh Portable. It does coarse address decodes to system RAM and ROM as well as the SCC, SCSI, VIA, SWIM, and Video Display Interface peripheral chips. It controls the transmission direction of the RAM data buffers and generates upper and lower RAM write strobes. It also provides several support functions such as the CPU clock generator, sleep state circuit, and DTACK generator.

5.10 Fine Address Decode and GLU

The Fine Address Decode and GLU chip is a custom CMOS gate array chip that accepts partially decoded addresses from the Coarse Address Decode and GLU chip and further decodes them to generate the chip selects for RAM. It also provides power manager timing signals, floppy disk drive enable signals, and serial port output signals.

5.11 VIA interface

The VIA is a 65C22A or 65C23 (depending on the manufacturer) CMOS Versatile Interface Adapter containing multiple registers. Which of these registers drives the 68000 data lines (a CPU read) or is driven by them (a CPU write) is determined by the four RS (Register Select) pins which are connected to four CPU address lines. Register A connects to the 8-bit I/O bus of the power manager processor, and it is through this port that the communication between the power manager and the central processor takes place. Another function of the VIA is synchronous and asynchronous generation of interrupts (for display vertical blanking every page scan, one-second intervals from the real-time clock, SCSI pseudo-DMA, and power manager initiated interrupts of the CPU). Control functions affect floppy disk head selection and serial communications controller write signaling.

5.12 SCSI Interface

The Small Computer System Interface (SCSI) consists of the NCR 53C80 chip between the CPU and an external DB-25 connector. Multiple SCSI devices may be connected to the SCSI bus through this connector; the 53C80 accomplishes the bus arbitration to allocate access to the 68HC000 central processor. An internal SCSI connector is used for connection to the optional internal hard disk drive.

The pinout for the external connector is shown in Table 5-5, and the pinout for the internal connector in Table 5-6.

♦ **WARNING:** Any internal hard drive connected to the Macintosh Portable SCSI bus should be a low-power version in order to maximize operating time per battery charge. ♦

The NCR 53C80 is a CMOS device designed to support the SCSI as defined by the American National Standards Institute (ANSI) X3T9.2 Committee. This device supports arbitration of the SCSI bus, including reselection. The chip is controlled through a set of read and write registers that are byte addressable only, and which are accessed through the SCSI Manager

The SCSI bus consists of eight data lines, a parity line, and nine control signal lines; the remaining connector pins are for power (supplied by the terminating device) or ground. A pseudo-DMA mode allows read and write transfers at the faster bus-limited speed, without processor control of every byte transfer. This interface is unchanged from that in the Macintosh SE, except that the Macintosh Portable does *not* supply the termination power.

The NCR 53C80 is connected directly to both the internal and external connectors and is capable of sinking 48 mA through each of the pins connected to the bus. The data and control lines on the SCSI bus are active low signals driven by open-drain outputs.

Termination power is *not* supplied by the Macintosh Portable, therefore it cannot be expected that the bus will remain in a known state if all external SCSI devices are powered off. The normal termination configuration connection is a 220Ω resistor to +5V and a 330Ω resistor to ground on each of the active signals. The internal SCSI connector is an exception to this scheme, however. Internal termination is supplied by the drive itself and is configured as a single-ended 1.3KΩ pull-up resistor to +5V.

• **Table 5-5** SCSI External Connector Pinout

Connector Pin Number	SCSI Bus Name	53C80 Pin Name
1	/REQ	/REQ
2	/MSG	/MSG
3	I/O	/I/O
4	/RST	/RST
5	/ACK	/ACK
6	/BUSY	/BSY
7	GND	
8	/DATA0	/DB0
9	GND	
10	/DATA3	/DB3
11	/DATA5	/DB5
12	/DATA6	/DB6
13	/DATA7	/DB7
14	GND	
15	C/D	/C/D
16	GND	
17	/ATN	/ATN
18	GND	
19	/SEL	/SEL
20	/PARITY	/DBP
21	/DATA1	/DB1
22	/DATA2	/DB2
23	/DATA4	/DB4
24	GND	
25	NO CONNECTION	

- **Table 5-6** SCSI Internal Connector Pinout

Connector Pin Number	SCSI Bus Name
1	/REQ
2	GND
3	/MSG
4	/C/D
5	/I/O
6	GND
7	/ACK
8	/ATN
9	/BSY
10	/RST
11	GND
12	/SEL
13	/DBP
14	/DB0
15	/DB1
16	GND
17	/DB2
18	/DB3
19	/DB4
20	/DB5
21	/DB6
22	/DB7
23	+5V
24	+5V
25	+12V
26	+12V
27	GND
28	GND
29-30	MOTOR GND
31-32	+12 V (MOTOR)
33-34	+5 V (MOTOR)

5.13 SWIM floppy disk interface

The SWIM interface is a combination MFM/GCR controller that connects directly to the CPU data bus. It is designed to replace the Integrated Woz Machine (IWM) and is pin and function compatible with the IWM. The SWIM chip is a combination of the traditional IWM chip, an ISM chip, and a combination logic chip.

The SWIM (Super Woz Integrated Machine) supports all IWM extensions including a status register, a mode register, and the following modes of operation:

- asynchronous mode
- fast mode
- optional 1-second one-shot

See the section in Chapter 3, "FDHD, the high-density floppy disk drive," for a description of the application of the SWIM floppy disk interface.

The SWIM also extends the IWM by providing an ISM mode supporting a high-speed rate twice that of the IWM, and programmable input clock frequencies. Other features of the ISM mode are

- Supports standard MFM format
- Supports Apple GCR format
- Write precompensation
- Read postcompensation
- Asymmetry and speed error compensation
- Programmable parameters for using both variable and fixed speed drives
- Two-byte data FIFO
- Motor time out
- Asynchronous mode with pollable handshake registers

The ISM makes it possible to read and write both MFM and Apple GCR formats on the same disk drive, and also makes it possible to write MFM format on a variable speed, 3.5-inch drive in such a way that it can be read back on a fixed speed 3.5-inch drive.

The ISM provides the ability to do write precompensation to correct for peak shift effects that occur in magnetically stored media.

The ISM also provides a very sophisticated, and rarely used, form of read postcompensation which corrects for peak shift effects on disks with insufficient precompensation.

The ISM uses a programmable parameter scheme that makes it possible to read and write 3.5-inch variable and fixed speed drives, as well as standard 5.25-inch drives.

The ISM contains a two-byte read and write FIFO stack to provide more software flexibility.

A Motor Time Out is included which will keep the drive enabled for 0.5 s to 1 s to provide time for software to begin another read or write operation without bringing the drive back up to speed.

The ISM makes it possible to program the phase lines as either inputs or outputs, which make it possible to interface with a wide variety of drives.

The SWIM interface consists of a single CMOS SWIM chip, an internal ribbon connector, and an external DB-19 connector. The pinout for the external connector is given in Table 5-7.

- **Table 5-7** SWIM Connector Pinouts

Pin Number	External DB-19
1	GND
2	GND
3	GND
4	GND
5	n.c.
6	+5V
7	+12
8	+12
9	n.c.
10	n.c.
11	PH0
12	PH1
13	PH2
14	PH3
15	/WREQ
16	HDSEL
17	/ENBL2
18	RD
19	W R

The SWIM is a CMOS device and is controlled through a set of read and write registers that are byte addressable only and which are to be accessed through the Device Manager and the driver as described in Chapter 3, "*Firmware*," under the heading "FDHD, the High Density Floppy Disk Drive".

The floppy disk drive can read and write on a 3.5-inch disk in any of the following modes: 1 MB Apple GCR (Group Code Recording) on a 1 MB disk, 1 MB MFM (Modified Frequency Modulation) on a 1 MB disk, and 2 MB MFM on a 2 MB disk.

The drive consists of two read/write heads, head positioning mechanism, disk motor, interface logic circuitry, read/write circuitry, and motor control circuitry. It includes auto inject/eject mechanisms and uses a 3.5-inch floppy disk for data storage.

5.14 SCC Interface

The Serial Communications Controller (SCC) is an 8 MHz CMOS Z8530 which has two independent ports for serial communication. Each port can be independently programmed for asynchronous, synchronous, or AppleTalk protocols.

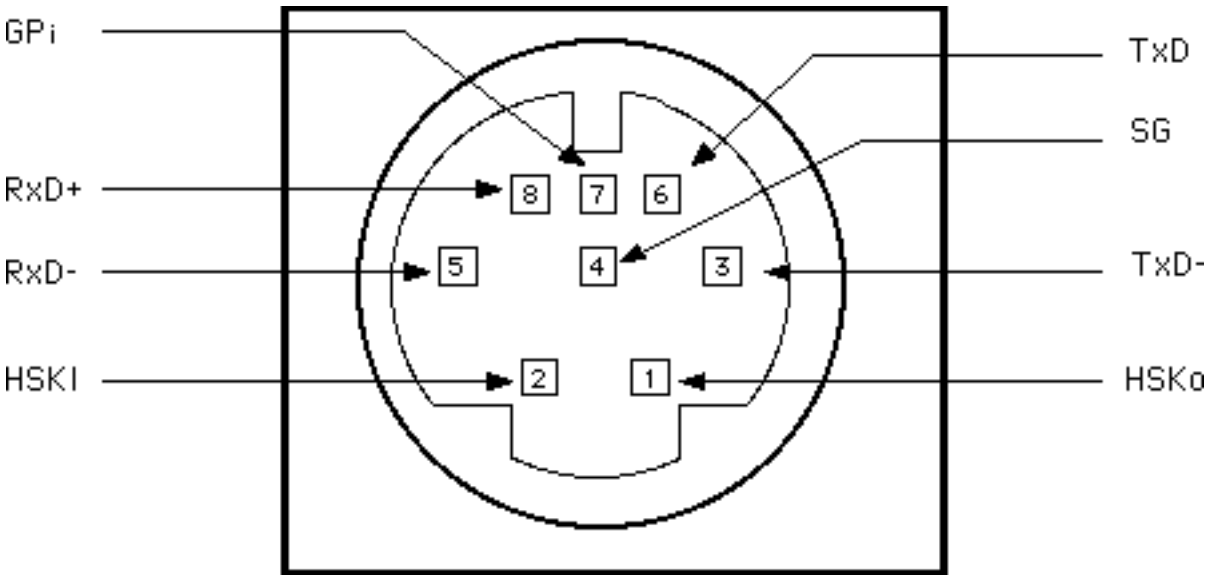
Power is applied to these ports under control of the power manager and this will occur only when you make the correct Toolbox calls. See Chapter 2, "Software Developer Guidelines," and Chapter 6, "The Power Manager."

The serial interface is connected to the output ports through two eight-pin miniature DIN connectors. Each signal line contains a 47-ohm series termination resistor. The pinout for the external connector is given in Table 5-8.

- **Table 5-8** SCC Connector Pinout

Pin Number	Port A	Port B	Description
1	HSKo	HSKo	Handshake output; Connected to SCC RTS; Tri-stated when DTR is inactive; Voh = 3.6V; Vol = -3.6V; Rl = 450Ω
2	HSKi	HSKi	Handshake input; Connected to SCC;CTS and TRxC; Vih = 0.2V; Vil = -0.2V; Ri = 12KΩ
3	TxD-	TxD-	Transmit data (inverted); Connected to SCC TxD; Tri-stated when DTR is inactive; Voh = 3.6V; Vol = -3.6V;Rl = 450Ω
4	SG	SG	Signal ground. Connected to logic and chassis ground.
5	RxD-	RxD-	Receive data (inverted); Connected to SCC RxD; Vih = 0.2V; Vil = -0.2V; Ri = 12KΩ
6	TxD+	TxD+	Transmit data; Connected to SCC TxD; Tri-stated when DTR is inactive; Voh = 3.6V; Vol = -3.6V; Rl = 450Ω
7	GPi	GPi	General purpose input; Connected to SCC DCD; Vih = 0.2V; Vil = -0.2V; Ri = 12KΩ
8	RxD+	RxD+	Receive data; Connected to SCC RxD. Vih = 0.2V; Vil = -0.2V; Ri = 12KΩ

- **Figure 5-8** SCC Mini-8 Connector



In the SCC, register addressing is direct for the *data* registers only. In all other cases (with the exception of WR0 and RR0), programming the Write registers requires two write operations and reading the Read registers requires both a write and a read operation. The first write is to WR0 (the Command Register) and contains three bits that point to the selected register. The second write (also to the Command Register) is the actual control word for the selected register; if the second operation is a read, the selected read register is accessed. All the registers in the SCC, including the data registers, may be accessed in this fashion. The pointer bits are automatically cleared after the read or write operation so that WR0 (or RR0) is addressed again. All address references to the SCC use offsets from the constant sccRBase ($FD 0000) for reads, and sccWBase for writes, as the base address and are byte-only addressable. These base addresses are also available in the global variables SCCRd and SCCWr. The offsets to the command and data registers are given in Table 5-9.

- **Table 5-9** SCC Address Offsets

Device	Base (R)	Base (W)	Register	Offset
SCC	$FD 0000 $0002	$FD 8000	Port A Command	
			Port A Data	$0006
			Port B Command	
	$0000			
			Port B Data	$0004

The SCC has a timing restriction in the time between accesses to the chip. Accesses to the chip must be at least 1.8 µsec from the end of the first access to the beginning of the second. The hardware implementation will prevent the next access until the appropriate wait has occurred, making this constraint transparent to the programmer.

5.15 Apple Desktop Bus (ADB)

The communication of data to the host 68HC000 from the keyboard, keypad, trackball and any other input device is via the Apple Desktop Bus (ADB), which consists of one serial, bi-directional data line, a 5 Vdc supply line, and ground, which is common for power and signal return. The ADB transceiver functions are implemented in the power manager processor while the keyswitch encoding is done by an ADB keyboard processor.

The standard configuration of the Macintosh Portable contains a miniature, low-power, ADB trackball. Input devices designated as *low-power* typically operate on about one tenth the current of standard types. The trackball module is designed to be installed—by the factory, dealer, or end user—into the position normally occupied by a numeric keypad in some other keyboards.

♦ **WARNING:** Any input devices connected, either internally or externally, to the Macintosh Portable ADB should be low-power versions in order to maximize operating time per battery charge. This means that normal keyboards and mouse devices from other members of the Macintosh family are usable with the Macintosh Portable, but at some sacrifice in reduced battery life between recharges. ♦

The operator input area of the Macintosh Portable has provision for, and comes equipped with, an alphanumeric keyboard module (keyboard) and a trackball module (see Figure 1-1). The trackball module may be placed by the user, either to the left or right of the keyboard. The trackball may also be replaced by an optional numeric keypad module (see Chapter 8, "Options").

The keyboard processor

The keyboard processor is a Mitsubishi M50740 8-bit microcomputer chip. The M50740's distinctive features are

- 3072 bytes of ROM
- 96 bytes of RAM
- 15 mW power dissipation
- 8-bit timer
- 32 programmable I/O ports

The keyboard processor is used to interpret the keyboard and numeric keypad matrix switch closures. The best way to visualize the internal actions of this processor is to recognize that it is the same as performed by the processors found in Apple's other keyboards. In the Macintosh Portable, however, the processor physically resides on the main logic board instead of the keyboard PCB.

The keyboard processor communicates with the power manager via the ADB in exactly the same fashion as it would if it were a separate, stand-alone keyboard.

Low-Power keyboard

The alphanumeric keyboard module is an array of switches only, without any active electronics. Each module has a steel plate into which have been inserted the keyswitches. The switches are interconnected by a PC board.

The alphanumeric keyboard has 63 keyswitches. The keyswitches are quiet-tactile, full-travel, and low-profile. The keycaps are a tapered style, and are platinum color.

The keyboard module is designed to be installable into the Macintosh Portable from the outside, by the customer. The module may be placed on the left or right side of the housing.

The alphanumeric keyboard module has two connectors, wired identically in parallel. The trackball/numeric keypad module has one connector. The connectors on the keyboard are wired with the pinout in Table 5-10.

• **Table 5-10** Keyboard Connectors Pinout

Signal Name	Pin Number			Pin Number	Signal Name
GND3	1	•	•	2	X0 (KEYMATRIX)
X1	3	•	•	4	X2
X3	5	•	•	6	X4
X5	7	•	•	8	X6
X7	9	•	•	10	X8
X9	11	•	•	12	X10
Y0	13	•	•	14	Y1
Y2	15	•	•	16	Y3
Y4	17	•	•	18	Y5
Y6	19	•	•	20	Y7
CAPS LOCK	21	•	•	22	SHIFT
CONTROL	23	•	•	24	OPTION
COMMAND	25	•	•	26	GND1
GND2	27	•	•	28	(SPARE)
ADB	29	•	•	30	BUTTON
See Note 8.	31	•	•	32	See Note 8.
(SPARE)	33	•	•	34	GND3

Notes:

1. This connector interfaces to the keyboard, the numeric keypad, the trackball, or any compatible ADB device.
2. GND1 is keyboard common for contact closures.
3. GND2 is ADB systems signal/power return ground.
4. GND3 is ESD/EMC/Chassis ground.
5. Disposition of grounds is handled on main logic board.
6. Two cables/connectors are required on main logic board, one at either side of computer housing.
7. Cable connectors from CPU board are to mate with 34-pin center-polarized headers. (Molex 5342-NGS2 series, No. 39-26-7349, or equivalent).
8. Pins 31 and 32 are connected together on the ISO (European) keyboards; this connection is used to identify such keyboards.

Low-power trackball

The trackball is electrically compatible with the ADB, although it uses few of the pins in a large, shared connector instead of the dedicated mini-circular type. Table 5-11 shows the pinout for the trackball connector.

In the trackball's intended application, the host controls the flow of power to the trackball, and it may be removed or restored at any time. Although not a strict ADB spec requirement, this unit is designed to be operational in the default mode, within 80 ms of the application of power.

The trackball's default handler ID is 0001 and its address is 0011, the same as a mouse.

Movement of the top of the ball's surface to the right is in the Positive X direction, and movement Down is in the Positive Y direction (see Figure 5-9).

There is one button for the trackball, which is the equivalent of the button on a mouse The trackball and the button are equally available to left- or right-handed users.

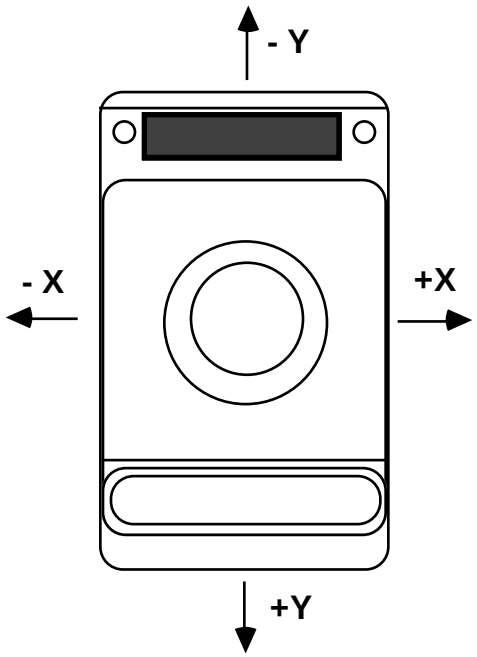
The 34-pin trackball connector, into which a cable from the 68HC000 CPU is plugged, is a dual-row, center-polarized header.

• **Table 5-11** Trackball Connector Pinout

Pin Number	Signal Name	Function
Pin 27	GND2	Signal power ground return
Pin 28	+5V	Power supply from host
Pin 29	ADB DATA	Bi-directional serial data
Pins 1 and 34	GND3	Shield ground

♦ All other connector pins should NOT be connected.

- **Figure 5-9** Trackball Direction Conventions



5.16 Sound interface

The sound interface is designed to be upward compatible with Macintosh sound using the Apple Sound Chip (ASC); it offers stereo sound and other enhancements not available in the classic Macintosh and Macintosh SE computers and available for the first time in the Macintosh II. This chip has two output channels and four major functional modes:

1. Four Voice Synthesis (mono)
2. Two Voice Synthesis (stereo)
3. Single Voice (stereo)
4. Single Voice (mono)

The hardware of the Apple Sound Chip subsystem consists of these components:

- the Apple Sound Chip (ASC)
- two analog sound-processing chips (the Sony sound chips)
- an internal high-impedance speaker
- a stereo phono jack

The speaker is a permanent magnet moving-coil type. The Macintosh Portable stereo-phone jack is located on the rear of the machine.

The features of this subsystem are available through the Macintosh Sound Manager.

5.17 Macintosh Portable expansion bus interface

The Macintosh Portable expansion interface follows closely that of the Macintosh SE. The 68HC000 signals are brought out on a processor direct slot (PDS), which is a 96-pin Euro DIN connector, allowing for expansion capabilities not available through other features of the machine. The pinout for the expansion connector is given in Table 5-12. Table 5-13 gives the signal descriptions.

- **Table 5-12** PDS Expansion Connector Pinout

Pin Number	Row A	Row B	Row C
1	GND	GND	GND
2	+5V	+5V	+5V
3	+5V	+5V	+5V
4	+5V	+5V	+5V
5	/DELAY.CS	SYS.PWR/	VPA/
6	/VMA	/BR	/BGACK
7	/BG	/DTACK	R/W
8	/LDS	/UDS	/AS
9	GND	+5/0 V (Pmgr Switched)	A1
10	A2	A3	A4
11	A5	A6	A7
12	A8	A9	A10
13	A11	A12	A13
14	A14	A15	A16
15	A17	A18	Reserved
16	Reserved	Reserved	Reserved
17	Reserved	Reserved	Reserved
18	Reserved	Reserved	Reserved
19	Reserved	+12 V	D0
20	D1	D2	D3
21	D4	D5	D6
22	D7	D8	D9
23	D10	D11	D12
24	D13	D14	D15
25	+5/3.7 V	+5V	GND
26	A19	A20	A21
27	A22	A23	E
28	FC0	FC1	FC2
29	/IPL0	/IPL1	/IPL2
30	/BERR	/EXT.DTACK	/SYS.RST
31	GND	16M	GND
32	GND	GND	GND

• **Table 5-13** PDS Expansion Connector Signal Descriptions

Signal Designator	Signal Description
GND	Logic ground
D0–D15	68HC000 unbuffered data bus 0–15
A1–A23	68HC000 unbuffered address bus 1–23
16M	16 MHz clock
/EXT.DTACK	External DTACK. This signal is an input to the CPU logic glue and allows for external generation of /DTACK.
E	68HC000 ECLK
/BERR	68HC000 Bus Error
IPL2–IPL0	68HC000 Interrupt Priority Level 2–0
/SYS.RST	68HC000 Reset
/AS	68HC000 Address Strobe
/UDS	68HC000 Upper Address Strobe
/LDS	68HC000 Lower Address Strobe
R/W	68HC000 Read/Write
/DTACK	68HC000 Data Acknowledge
/BG	68HC000 Bus Grant
/BGACK	68HC000 Bus Grant Acknowledge
/BR	68HC000 Bus Request
/VMA	68HC000 Valid Memory Address
/VPA	68HC000 Valid Peripheral Address
FC2–0	68HC000 Function Code 2:0

The current available to an expansion card inserted in the processor direct slot (PDS) is given in Table 5-14. This current allocation is part of a worst-case current budget that is estimated to reduce the system operating time per battery charge by fifty percent.

- **Table 5-14** Current Available To The Processor Direct Slot

Power Supply	Operating State	Sleep State
+ 5 V, Always-on	50 ma maximum	1 ma maximum
+ 5 V, Switched	*	0 ma maximum
+ 12 V	25 ma maximum	0 ma maximum

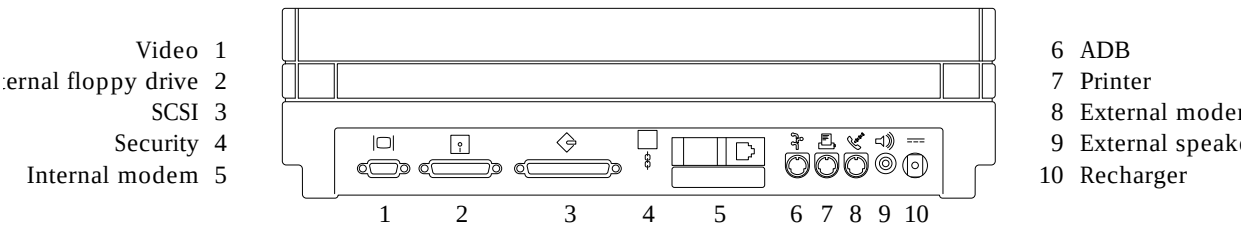
* The 50 ma maximum applies to the sum of the loads on the switched and unswitched (by the power manager) +5 V supplies.

5.18 The Macintosh Portable I/O port connectors

This section describes the connectors on the rear of the Macintosh Portable. Figure 5-10 shows these connectors. Descriptive names for these connectors, from left to right in the figure, are

- Video
- External disk drive
- External SCSI device
- RJ-11 Telephone receptacle
- Apple Desktop Bus (ADB)
- Serial port (Modem)
- Serial port (Printer)
- Stereo sound
- Battery charger, DC power input

- **Figure 5-10** I/O port connectors



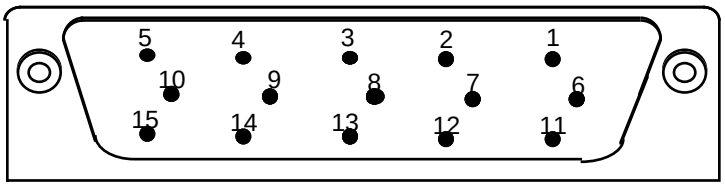
The connectors are described in the following sections. For each I/O connector, there is a drawing showing the pin or socket location and numbering. Each drawing is followed by a tabular description of signal names and functions.

Video connector

The Macintosh Portable has a DB-15 connector at the rear that provides the video output signals to an external video adapter. The adapter produces video signals with Macintosh II, NTSC, or PAL formats.

The physical connection is a "D" subminiature receptacle connector. The pinout of this connector is shown in Figure 5-11. The signal names and descriptions are given in Table 5-15.

- **Figure 5-11** External video connector



- **Table 5-15** Video connector signal assignments

Pin number	Signal name	Signal description
1	D0	Data bit 0
2	D1	Data bit 1
3	+5V	
4	D2	Data bit 2
5	D3	Data bit 3
6	D4	Data bit 4
7	GND	Signal Ground
8	Vbb	Battery Voltage
9	GND	Signal Ground
10	D5	Data bit 5
11	D6	Data bit 6
12	D7	Data bit 7
13	+5V	
14	New_Frame	FLM from Video Display Interface chip, Begin frame scan over

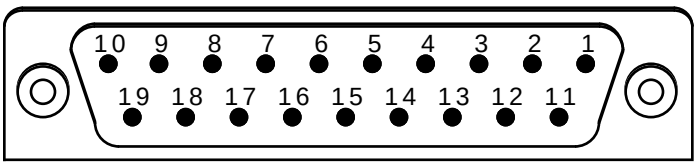
15	New_Byte	CL2 from Video Display Interface chip, Byte clock

External disk drive connector

This port can be used to support the external 3.5-inch disks used by all Macintosh computers, as well as the Apple Hard Disk 20.

The pinout of the external disk drive connector is given in Figure 5-12. The connector is a receptacle, DB-19 type.

- **Figure 5-12** External Disk Drive Connector



The signal names and descriptions for the external disk connector are given in Table 5-16

- **Table 5-16** External disk drive connector signal assignments

Pin number	Signal name	Signal Description
1	GND	Logic ground
2	GND	
3	GND	
4	GND	
5	no connection	
6	+5V	
7	+12	
8	+12	
9	no connection	
10	no connection	
11	PH0	Phase 0
12	PH1	Phase 1
13	PH2	Phase 2
14	PH3	Phase 3
15	/WREQ	
16	HDSEL	Head select
17	/ENBL2	Enable
18	RD	Read
19	WR	Write

External SCSI connector

Like the Macintosh SE,the the Macintosh Portable has a built-in SCSI port for high-speed parallel communications. The SCSI interface can communicate with up to seven SCSI devices, such as hard disks, streaming tapes, and high-speed line printers. The external SCSI port is a DB-25 connector as shown in Figure 5-13.

- **Figure 5-13** External SCSI connector

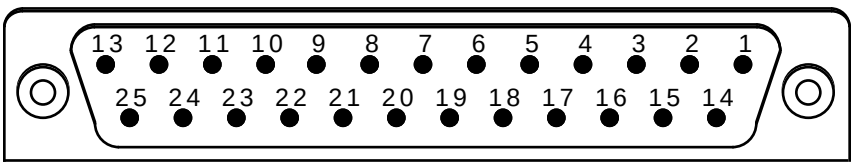


Table 5-17 shows the signal assignments for the SCSI DB 25-pin external connector. These signals are described in detail in the *NCR 5380 SCSI Interface Chip Design Manual* and the IEEE SCSI specification—Section D, ANSIX3T9.2 (version 17B).

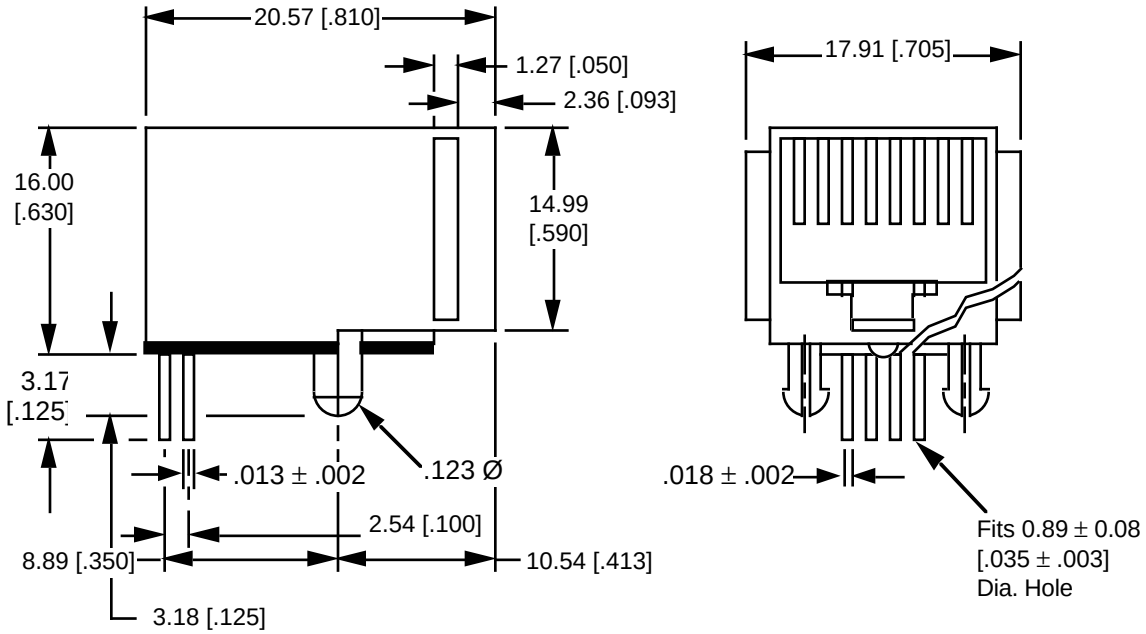
- **Table 5-17** External SCSI Connector Signal Assignments

Pin number	Signal name	Signal description
1	/REQ	Request for a REQ/ACK data transfer handshake
2	/MSG	Indicates the Message phase
3	/I/O	Controls the direction of data movement
4	/RST	SCSI bus reset
5	/ACK	Acknowledge for a REQ/ACK data transfer handshake
6	/BSY	SCSI data bus is busy
7	GND	Ground
8	/DB0	Bit 0 of SCSI data bus
9	GND	Ground
10	/DB3	Bit 3 of SCSI data bus
11	/DB5	Bit 5 of SCSI data bus
12	/DB6	Bit 6 of SCSI data bus
13	/DB7	Bit 7 of SCSI data bus
14	GND	Ground
15	/C/D	Indicates whether control or data is on the SCSI bus
16	GND	Ground
17	/ATN	Indicates an attention condition
18	GND	Ground
19	/SEL	Select a target or an initiator
20	/DBP	Parity bit for SCSI data bus
21	/DB1	Bit 1 of SCSI data bus
22	/DB2	Bit 2 of SCSI data bus
23	/DB4	Bit 4 of SCSI data bus
24	GND	Ground
25	NC	No connection

RJ-11 telephone receptacle

The RJ-11 receptacle, which is mounted on (and a part of) the optional modem card, is accessable through the rear of the Macintosh Portable. Figure 5-14 is a drawing of this connector.

- **Figure 5-14** RJ-11 Telephone Receptacle



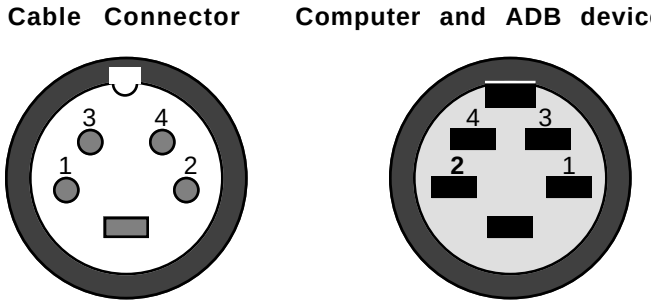
Dimensions are in units of mm [inch].

Apple Desktop Bus (ADB) connector

The ADB is a serial communications bus designed to accommodate low-speed input devices. The ADB port is a 4-pin mini DIN connector as shown in Figure 5-15.

♦ **WARNING:** Any input devices connected, either internally or externally, to the Macintosh Portable ADB should be low-power versions in order to maximize operating time per battery charge. This means that normal keyboards and mouse devices from other members of the Macintosh family are usable with the Macintosh Portable, but at some sacrifice in reduced battery life between recharges. ♦

• **Figure 5-15** The Macintosh Portable ADB connector



• **Table 5-18** The Macintosh Portable ADB signal assignments

Pin number	Signal name	Signal description
1	ADB	The bidirectional data bus used for input and output. It is pulled up to +5V through a 470 ohm resistor on the logic board, and is an open collector type signal.
2	n.c.	Not connected
3	+5V	+5V power
4	GND	The logic ground and power return

Serial ports (modem/printer)

The Macintosh Portable has two RS–422 serial I/O ports for printers, modems, and other standard serial I/O devices. Two mini-8 connectors on the computer's back panel are used for the serial ports. As shown in Figure 5-10, the modem port is labeled with a phone handset icon while the printer port is labeled with a printer icon. Figure 5-16 details the pin numbering of the mini-8 connectors.

The Macintosh Portable incorporates an internal modem connector. A compatible modem inserted into the modem slot connector is connected to the modem port. (Hardware supports the internal modem being switched to operate through either of the two ports, but firmware accomodates operation only through the modem port.)

An *external* modem may be connected to either the modem or the printer port when an internal modem is *not* used; the user of the Macintosh Portable makes a selection through the Control Panel. See Chapter 8, "Options," for details on the Apple modem and the interface to any card you might design to use in the modem slot connector.

The two serial ports are identical except that the modem port has a higher interrupt priority, making it more suitable for high-speed communication. See *Inside Macintosh*, Volume III or *Macintosh Family Hardware Reference* for additional details.

- **Figure 5-16** Serial ports

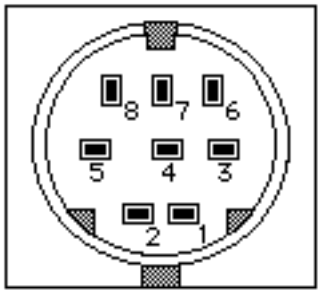


Table 5-19 shows the signal assignments for the serial ports.

- **Table 5-19** Serial Port Signal Assignments

Pin number	Signal name	Signal description
1	HSKo	Output handshake
2	HSKi	Input handshake or external clock
3	TxD–	Transmit data –
4	SG	Signal ground
5	RxD–	Receive data –
6	TxD+	Transmit data +
7	GPi	General purpose input
8	RxD+	Receive data +

Stereo phone jack

The stereo phone output is capable of driving headphones of 8 to 600 ohms and is short-circuit protected.

The connector is a miniature phone jack (receptacle) connector.

- **Table 5-20** Stereo phone jack pinout

Pin Number	Function
1	Ground
2	Left channel
3	Right channel

DC power input for the battery recharger

The power jack is the interface between the Macintosh Portable and the power adapter which recharges the battery. Table 5-21 provides the pinout.

- **Table 5-21** Power Jack Pinout

Pin Number	Function
1	+ Battery voltage
2	– Battery return
3	No connection

5.19 Battery recharger

The recharger is an enclosed, wall-mounted module provided with an AC input plug; a DC output cable terminates in a connector that mates with the battery recharger connector, whose location is shown in Figure 1-2.

- **Table 5-22** Electrical Requirements

Parameter	Minimum	Nominal	Maximum	Units
AC Input Voltage Range	85	120/240	270	VACrms
AC Input Frequency Range		48	50/60	62 Hz
DC Output Voltage Range		7.0	7.5	7.6 Volts
DC Output Current Range		0.005	1.5	2.0 Amperes

...

Chapter 6 The Power Manager

This section includes and introduction, a description of the power manager states, power management hints to hardware developers (software developers, see Chapter 2), and a list of power manager ROM calls. •

6.1 Introduction

The power manager is an intelligent assistant to the 68000 CPU. The Macintosh Portable power manager provides these functions:

- Power management activities (i.e., enabling or disabling clocks to peripheral chips—like the SWIM chip—to reduce power consumption during idle or sleep periods, and physically enabling or disabling the various power planes).
- Performing the Macintosh real-time clock (RTC) functions.
- Performing the transceiver functions for the Apple Desktop Bus (ADB).
- Generating the contrast signal level (via the PWM function) for the flat panel display.
- Monitoring the temperature (via an A/D input and a thermistor) inside the case to enable the CPU to either warn the customer, or take some other protective action, in the event that the machine becomes too hot.
- Monitoring the level of battery charge so that the customer can be warned if it becomes too low, or if the machine should shut down all activity in order to preserve its memory contents.
- Generating the 1 sec interrupt signal to the CPU.

The power manager communicates with the CPU by using an asynchronous handshake mechanism and an 8-bit parallel data bus in conjunction with the VIA. The power manager primarily responds to commands from the 68HC000.

6.2 Power manager states—idling, sleeping, and waking

An important requirement of any portable computer is to conserve battery power. Macintosh Portable accomplishes this in a number of ways: powering down peripheral subsystems when not needed, slowing (idling condition) or stopping (sleeping condition) the 68000 clock when full speed is not required.

The power manager goes into the sleep state for one of two reasons: it detects a very low battery condition, or it receives the Sleep command from the 68000. The 68000 sends the Sleep command when the Macintosh Portable operating system has determined that there is no user activity or when the user decides to stop work and to shut down the system. Before sending the command, the operating system and drivers save their state variables in RAM. These variables are used later to restore the system when power is restored. The power manager sets a flag indicating that the Sleep command has been received and confirmed, then turns down all system power with the exception of the keyboard processor and the ring-detect circuitry of a modem card, if one is installed.

At the end of the power manager sleep code are two instructions that toggle one of the power manager processor's output lines. This line is connected to a circuit that disables the power manager processor's own 3.9 MHz clock. This essentially "stops time" for the processor, halting its execution and lowering its power consumption by two orders of magnitude. The processor internal state is frozen, with all internal RAM and control registers remaining intact. This is defined as the power manager's **sleep** state—powered, yet stopped.

The 60 Hz external clock used as the basic power manager processor time base, is also used to re-enable the 3.9 MHz clock. The rising edge of the 60 Hz clock re-enables the 3.9 MHz clock, which starts the power manager processor executing code. The power manager resumes execution at the exact spot where it turned itself off and begins the wake-up check loop. First, the clock and times are updated, then the environment is checked for a reason to wake up (return to the operating state).

There are three possible conditions that will end the sleep state:

1. If a key is down on the keyboard
2. If the wake-up timer is enabled and matches the real-time clock
3. If the Macintosh Portable internal modem is installed and set up to watch for "ring detect"

If any one of these conditions is met, the power manager restores itself and the system to normal activity (powers up the 68000, RAM, ROM, etc). When main power returns, the 68000 restores itself and the rest of the system to where it was before the sleep state. This is called **waking**.

BatteryMonitor Code on the 68000

The BatteryMonitor is a segment of code on the 68000 that is part of the ROM image. The BatteryMonitor is called by the 1 sec interrupt (from the power manager processor) and it monitors the CPU for activity. More specifically it

- checks the battery level and looks to see if it is necessary to alert the user that a
  low-power condition exists
- looks at sleep and hard disk spin-down times (user selected) to see if they've been changed, and updates them as necessary. These times are stored in parameter RAM, which is implemented in the power manager processor.
- turns off the high current drain sound circuit when it is not needed
- updates the real-time clock, which is implemented in the power manager processor
- checks the internal computer temperature to see if it is necessary to alert the user that a high-temperature condition exists
- watches for when it is time to go into the sleep state, that is, it waits for a sleep timeout to occur
- puts the system in the sleep state

Clearly, this code segment does a great deal more than its name implies.

Idle/sleep code function

Idle/sleep code monitors, with the help of other parts of the system ROM, the indications of activity. Any one of these types of activity will prevent a sleep timeout:

- a call to Read or Write through IOCore
- calling PostEvent
- OSEventAvail returning true (events are in the queue)
- making sound (any access of an ADB sound chip)
- execution od an ADB completion routine

- a call to SetCursor that changes the cursor
- whenever the watch cursor is the cursor

Criterion for idle

The criterion for idle is 15 seconds without user activity of any kind (including communication through the serial port; for example, modem use).

Software developers, you need to ensure that routines involving a large amount of calculation perform at least one of the listed types of activity to prevent the power manager commanding the 68000 into the idle state.

In idle, the 68000 processor inserts 64 wait states into RAM and other accesses to lower the processor effective frequency to near 1 MHz, even though its clocking continues at 16 MHz. Interrupts still get processed at 16 MHz, full speed, in the interrupt handler.

Criterion for sleep state

For the sleep state, the variable time used as an inactivity criterion is user selected in the Battery desk accessory (see Figure 4-2). The same measure of inactivity is used for the sleep state as for idle, but in considering the transition from idle to sleep, the user established sleep-time is counted down; if the countdown reaches zero without indication of activity, the Macintosh Portable goes into sleep state.

During sleep there are four sections that remain turned on (powered for full functionality):

1. RAM (both main memory and video memory) is backed up
2. Portions of a modem card, if one is installed
3. The VIA and the SCC are powered but their clocks have been reduced to zero frequency, thereby reducing their power consumption to almost zero.
4. The power manager processor, once the sleep call is made, saves some state variables, systematically powers down all the peripheral subsystems over which it has control (by stopping the system clock signal to them), and then puts itself to sleep (powers itself down).

1/60th of a second interrupts are generated continuously by the 60 Hz clock as long as the battery is charged. When the power manager processor receives an interrupt, it does the following:

- updates the real-time clock
- checks the wake-up timer to see if it matches the real-time clock
- checks for evidence of events that should end the sleep state, such as 1) any keystroke at the built-in keyboard, 2) wake-up timer timeout, or 3) Ring Detect from the modem is true and the modem feature has been enabled at the Control Panel.

If no indication of activity survives the validity check then the power manager powers itself down again. The periodic functions take perhaps 200 microseconds out of the 16.7 milliseconds between interrupts, so the power manager is powered down most of the time.

When someone presses a key or some other event occurs, the power manager will recognize that event, restore itself, hold down (assert) the RESET signal to the 68000, turn on power to the system, and then raise (deassert) the RESET signal.

The 68000 has now been reset and its program counter returned to location $00 0000; the execution of the second instruction asks the question “Is this a return from sleep (a wake-up) or a true reset (from activation of the reset switch) ”. If it's a return from sleep, the program jumps off to the wake-up code, which restores each subsystem to its state before it went into the sleep state and returns to the calling program.

6.3 Power management hints (hardware)

The following hints may be of interest to hardware designers: Chapter 2, “Software Developer Guidelines,” contains hints more likely to be of interest to application designers.

Microprocessor

The 68HC000 is a mostly static cell CMOS design and as such has a DC component to the current drawn during operation (as well as an AC component). This DC component (~10 mA) is always present while power is on to the processor and is the reason the power to the microprocessor is turned off during sleep. For the AC component, three factors determine the amount of current drawn during idle or full speed operation:

- Frequency of operation
- Number of gates switching
- Capacitance to be charged when switching

The frequency of operation cannot be taken to DC because there are some dynamic nodes present in the design. Instead there is a provision (in the Normandy ASIC, $FE xxxx space) to put the processor into an *idle mode* where there are 64 wait states injected into every RAM/ROM cycle. Introduction of the wait states reduces the effective clock frequency for a large portion of the chip. The result of power management is that the number of gates that are switching is reduced as is the frequency at which they switch. The frequency at which internal and external capacitive loads are being charged is also proportionately reduced; therefore the contribution to overall power consumption by address/data/control bus capacitance, and by the RAM and ROM arrays, is reduced.

Permanent main memory

The RAM is a static cell CMOS design and as such has a significant DC component (as well as an AC component) to the current drawn only while it is selected. The RAM array is configured using x8 parts in order to reduce this selected current (only 2 chips are selected in the array, the rest are in standby—no matter how deep the array is). The current drawn while operating is approximately proportional to the duty cycle of use. For instance, when the 68HC000 processor is put into idle mode (64 wait states are injected into each bus cycle) the RAM will be selected only for the minimum amount of time to guarantee proper operation (i.e., the select time remains about the same but the overall duty cycle is reduced dramatically).

ROM memory

The ROM is a static cell CMOS design and as such has a significant DC component to the current drawn only while it is selected (as well as having an AC component). The current drawn while operating is approximately proportional to the duty cycle of use. For instance, when the 68HC000 processor is put into 'idle' mode (64 wait states are injected into each bus cycle) the ROM will be selected only for the minimum amount of time to guarantee proper operation (i.e., the select time remains about the same but the overall duty cycle is reduced dramatically).

Floppy disk interface

The SWIM chip is a static cell CMOS design and as such has a negligible DC component to the current drawn during operation (the major contributing factor is the AC component). For the AC component, three factors determine the amount of current drawn:

- Frequency of operation
- Number of gates switching
- Capacitance to be charged when switching

The strategy for power management on the SWIM chip is to control the frequency of the clock (power is left on to this device even during the sleep state). Specifically it is either on and switching at 16 MHz or it is off (DC). Savings by using this technique have been measured to be about 35 mA. The state of the clock is controlled via the power manager processor through the Normandy ASIC. The normal state of the clock is to be off (thus providing the lowest average power). However, because the floppy drive is not able to interrupt the system when a disk is inserted (and subsequently notify the system to turn the SWIM chip clock on) the system must periodically turn the clock on and check for a disk insert event. Of course, while a disk is inserted and the drive is in operation, the clock should be on.

SCC Interface

The SCC chip is a static cell CMOS design and as such has a negligible DC component to the current drawn during operation (the major contributing factor is the AC component). For the AC component, three factors determine the amount of current drawn:

- Frequency of operation
- Number of gates switching
- Capacitance to be charged when switching

The strategy for power management on the SCC chip is to control the frequency of the clock (power is left on to this device even during the sleep state). Specifically it is either on and switching at 8 MHz or it is off (DC). The state of the clock is controlled via the power manager processor through the Normandy ASIC. The normal state of the clock is to be off (thus providing the lowest average power). However, when the serial ports are in use, the clock must be on 100% of the time.

Video Display Panel Interface

The active matrix display's current drain is dependent upon the screen image. For instance a gray screen may draw up to three times the current of an all white or all black screen. Savings of about 50-70 mA are achievable mainly if the gray desktop screen is avoided. The default screen pattern has been designed to reduce this current drain.

6.4 Operating system interface (ROM calls)

You should already know what the sleep state is and how it is used (see section 6.2 “Power Manager States-Idling, Sleeping, and Waking” for information on the sleep state). Only specific parts of the system should be calling sleep. It is more likely that you want to get into the sleep queue, which is described later in this section.

Calling Sleep

Sleep request

A sleep request is called using the Sleep trap (_Sleep, or $A08A) and it takes one parameter in D0, as shown below.

```
            MOVE.L      #SleepRequest,D0  ; SleepRequest = 1
            _Sleep                        ; $A08A
            BNE.S       @didnotsleep      ; Sleep indicator in the
 EQ                                       ; condition code
@didsleep   .........
```

All registers except D0 are preserved across the call.

A sleep request is only that, a request. If the system determines that there is a reason why the system should not go to sleep, the request is denied and the EQ condition code is set to not equal.

Entities that request sleep are sleeptimer, Finder, and Shutdown.

Sleep demand

A sleep demand is called using the Sleep trap (_Sleep, or $A08A) and it takes one parameter in D0, as shown below.

```
            MOVE.L      #SleepDemand,D0   ; SleepDemand = 2
            _Sleep                        ; $A08A
```

All registers except D0 are preserved across the call.

A sleep demand is unconditional. The Macintosh Portable will go into the sleep state even if some processes may be harmed. Sleep demand is generally used to power off the machine when a critical low-power condition arises.

The entity that demands sleep is low-power alert.

The Sleep Queue

As part of its preparation for sleep, the sleep trap goes through a queue of procedures, executing each one and informing the procedure of its intentions. The sleep trap may be trying to do any of three things: request permission for sleep, alert for impending sleep, or inform the procedure that wake-up is underway.

Entities that wish to be given CPU time before or after sleep must place themselves in this queue. The sleep queue is a standard OS queue. Shown below is the sleep queue record structure.

```
TYPE   SleepQRec = RECORD
                     SleepqLink:          QElemPtr;
                     SleepqType:          Integer;
                     SleepqProc:          ProcPtr;      {Pointer to sleep routine}
                     SleepqFlags:         Integer;
               END
```

The sqFlags field contains two flags to be set by the sleep routine before installing its record into the queue.

Flag	Bit	Indication
Sleep	0	Call procedure at sleep
WakeUp	1	Call procedure at wake up

The record owner may suspend calls to its sleep procedure by clearing these bits as desired.

Installing a record

To install a record in the queue:

1. Fill the flags, type, and proc fields in the record
2. Load A0 with a pointer to this record
3. Call SlpQInstall
4. Call enqueue

Example:

```
MOVE.L      MyRec(A2),A0                ; Pointer to my record
LEA         MySleepProc,A1              ; Load pointer to my
                                          handler
MOVE.L      A1,SleepqProc(A0)           ; Fill proc field
MOVE        #SlpQType,SleepqType(A0)    ; Fill type field (#16)
_SlpQInstall                            ; Add to queue ($A28A)
```

Removing a record

To remove a record from the queue:

1. Load A0 with pointer to the record
2. Call SlpQRemove

Example:

```
MOVE.L      MyRec(A2),A0            ; Pointer to my record
_SlpQRemove                         ; Remove from queue ($A48A)
```

Sleep queue procedures are called with A0 pointing to the procedure's sleep queue record. Procedures in the queue may be called with any of the following values in D0. y.

1 = Sleep request
2 = Sleep demand
3 = Wake up

Sleep Queue calls

A complete sleep call to each queue procedure consists of (in the case of SleepDemand or SleepNow) a single call to each sleep procedure with D0 containing the SleepRequest value or (in the case of SleepRequest) two calls to each sleep procedure, the first passing SleepRequest (in D0) followed by another call with SleepDemand (in D0).

These calls should not be confused with the system level _Sleep calls; the sleep queue procedures are merely called with the same parameter that was passed to the Sleep trap. Sleep queue procedures only see SleepRequest, SleepDemand, or SleepWakeUp passed to them; SleepNow calls are converted to SleepDemand calls when executing sleep queue procedures.

A test of the networking services in use is performed before the sleep queue is traversed. (See "Network Services" below.)

Sleep Request

For sleep request, each queue entry gets called with D0 containing the SleepRequest parameter and A0 pointing to the procedure's sleep queue record. The sleep queue procedure may either allow or deny this request and return its answer in D0. If the procedure allows sleep, the queue is traversed and the next sleep queue procedure is executed. If any one of the procedures denies the sleep request, sleep queue traversal is halted and all procedures that had been called so far get a SleepWakeUp call indicating that the sleep request is being aborted.

It is probably best for entries not to actually do sleep preparation at the time of request, as it is possible that the request may be denied by another sleep queue entry. Instead the queue entry should wait until the sleep demand call is made. SleepRequest is used to check with all entries to see if sleep is OK and to indicate to sleep queue procedure owners that they should lock out any activity that might make it not OK.

To OK the sleep request, the sleep queue entry procedure must clear D0. To deny the sleep request, D0 must be returned as a nonzero.

Once all the sleep queue procedures are called and none of them denies the sleep request, each is called again , this time with the SleepDemand parameter passed in D0. The SleepDemand call cannot be denied.

Sleep Demand

For a SleepDemand or SleepNow system level call, the sleep queue procedures are called once. In either case the SleepDemand parameter is passed in D0 (SleepNow is converted to SleepDemand by the _Sleep trap before executing the queue procedures.) The SleepDemand call cannot be denied. All entries in the sleep queue must do the best they can to prepare for a sleep demand and cooperate fully.

Since the _Sleep trap is not called at interrupt, the procedure can do quite a bit to make sleep possible (even use the Memory mManager!). In some cases it may be desirable to ask the user for an OK or to alert the user of some problems that may occur, however this should be avoided if possible. If many sleep queue procedures decide to put up a dialog, things can get confusing. In addition, if no user is present to answer your dialog, the machine will not go to sleep.

Sleep Wake Up

The SleepWakeUp call is made to each sleep queue entry upon wake up and provides an opportunity to sleep queue procedures to restore their state. This call cannot be denied.

Remember, choosy programs choose sleep.

Network services

A SleepRequest or SleepDemand call must pass a set of network condition tests before the sleep operation can be granted. The network tests take place before sleep queue procedures are called. If the network test fails, the sleep queue procedures are not called. The following matrix illustrates what happens.

Sleep Ca

Network Services	Reque	Deman	Now
MPI	If on battery, then close driv else denied	If user OKs, then close dr else denied	Close driv
XPI	If on battery, then close driv else denied	If user OKs, t close drivers; else denied	Close driv
XPP/ Volume Mounte	Denie	If user OKs, t unmount serv and close dri else denied	Unmount ser and close dri

For the (low priority) SleepRequest, sleep is denied if any network services are actually in use. The exception here is when running on battery where the overriding concern is conserving power. In the demand case things are more interesting.

In the SleepDemand case the user is presented with one of three alerts informing him of the possible consequences if sleep is chosen. Level one alert is shown if the MPP driver or the XPP driver is open, but no servers are mounted nor is the magic chooser bit set. The language of the alert box informs the user of possible loss of networking services. If a server volume is mounted, a more harsh message appears informing of the loss of the volume, and so on. In all cases the user is given the choice of canceling the sleep demand or going ahead with it.

The SleepNow case stomps on everyone and takes no prisinors. No dialogs appear.

Chapter 7 Expansion Card Design Guides

This chapter contains mechanical drawings to aid the expansion card hardware developer.

7.1 Main logic board with expansion connectors

Figure 7-1 shows the various elements of the main logic board.

- **Figure 7-1** Main logic board

7.2 Expansion card design guides

Figures 7-2 and 7-3 are design guides for Macintosh Portable expansion cards.

- **Figure 7-2** RAM card design guide

- **Figure 7-3** Modem card design guide

Chapter 8 OPTIONS

This chapter describes the options available to expand the functionality of the Macintosh Portable. •

8.1 The modem card

The following information is given to support developers who are

- Developing software to drive the Apple modem
- Developing hardware to fit into the modem slot, and the software to go with it

The modem card is an "AT" compatible 2400/1200/300 bps auto-dial, auto-answer modem. Its design is optimized for use in the Macintosh Portable. As a result, the modem requires little power while operating and has a sleep state in which only the ring detect circuitry is powered.

Summary of modem card features

The features include

- 2400/1200/300 bps data rates; meets Bell 103 and 212A, and CCITT V.21, V.22B and V.22 bis standards
- Automatic restoration of previous configuration at power up (warm start)
- Software selectable Bell 212A/103 or CCITT V.22/V.21 modes (B command)
- Data formats:

  7 data bits, 1 parity bit (even, odd, mark, or space parity), 1 stop bit

  7 data bits, no parity, 2 stop bits

  8 data bits, no parity, 1 stop bit
- Subset of “AT” command set (Hayes 2400)
- Automatic Adaptive Equalization on the receive channel
- Asynchronous operation only
- Auto-dial and auto-answer
- Pulse and DTMF dialing
- Full call progress detection: busy, dial tone, ring, ringback, 2nd dial tone

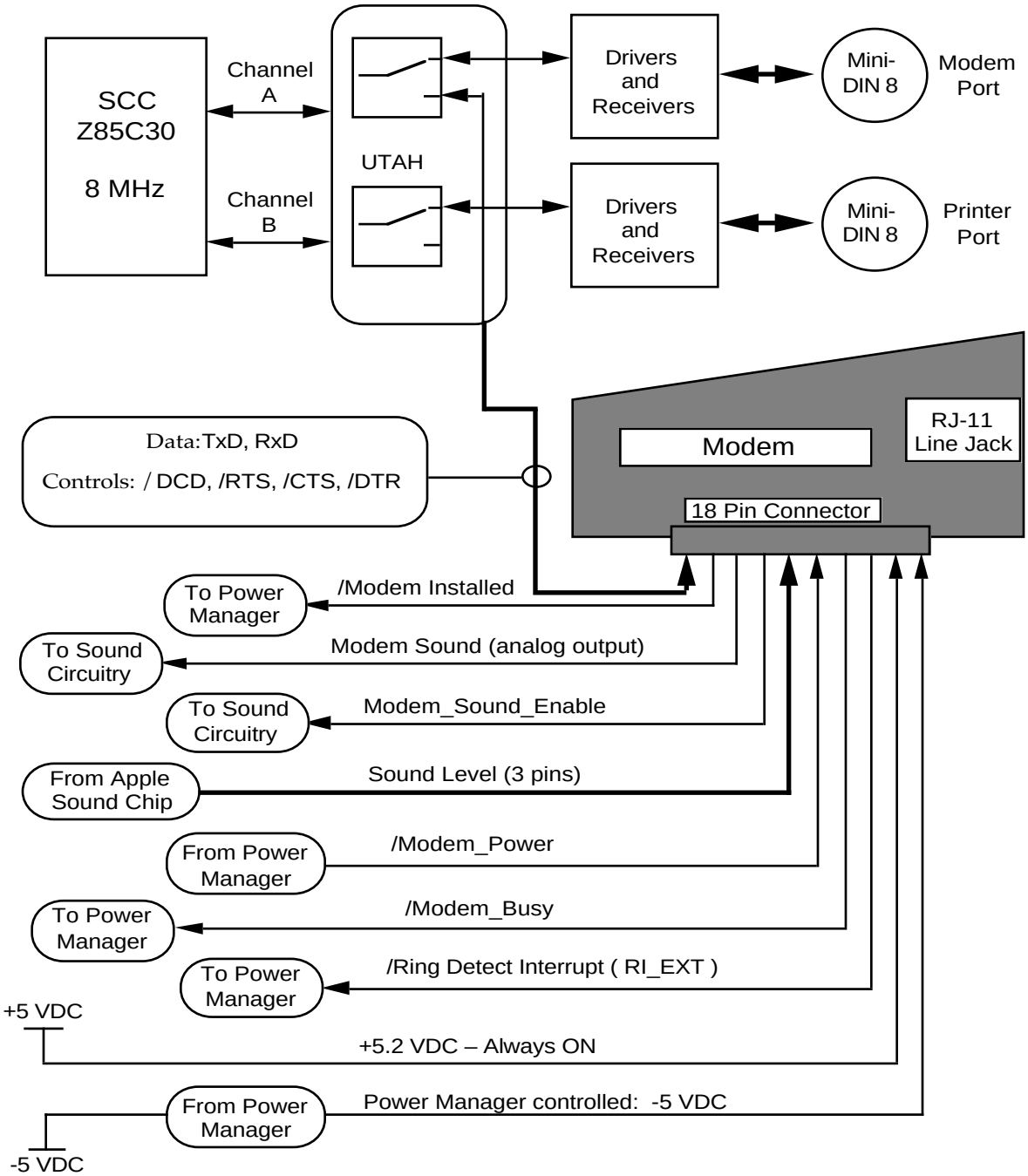
- Local analog and digital loopback self-test, and remote digital loopback self-test with test pattern generation (AT&Tn commands)

- 40-character buffer memory
- Designed to meet FCC Part 68 and Part 15 class B
- Low power dissipation: 500 mW maximum / 450 mW typical
- ± 5 V operation
- Compact size: 14 in^2 (3 in x 4.75 in)
- Single RJ-11 phone jack
- Analog output pin for audio monitoring of modem operation
- Sleep state: requires only +5 V for ring detect circuitry and maintenance of configuration at power down
- Domestic/Canadian Data Access Arrangement (DAA) design

Hardware interface

The hardware interface between the Macintosh Portable and a modem card inserted into the modem expansion slot is illustrated in Figure 8-1.

- **Figure 8-1** Macintosh Portable Serial Port Configuration and Modem Interface



The data access between the Macintosh Portable CPU and the modem is an 18–pin dual inline socket connector on the modem card. The data is at TTL levels (V_{IL} = 0 to 0.8 V; V_{IH} = 2.8 to V+; I_{OL}=1.6 mA; I_{OH}=-25 µA).

The internal connector that receives the modem card has pinouts as in Table 8-1. Table 8-2 relates signal names and functional descriptions.

- **Table 8-1** Modem Connector Pinout

Pin number	I/O Type	Signal Name
1	I (Input)	/Modem_Pwr (controlled through the power manager processor)
2	Ground	GND
3	I	/RTS
4	O (Output)	/DCD (Carrier Detect)
5	O	RxD (Receive Data) [to SCC RxD]
6	O	/CTS
7	O	Modem_Sound (Analog Out)
8	I	TxD (Transmit Data) [to SCC TxD]
9	O	/RI_EXT (Ring Detect Interrupt)
10	–5V	-5 VDC (controlled through the power manager processor)
11	+5 V	+5 VDC (always on)
12	I	/DTR (Data Terminal Ready)
13	I	V1 (volume control-LSbit)
14	I	V3 (volume control-MSbit)
15	I	V2 (volume control)
16	O	Modem_Ins
17	O	/Modem_Busy
18	O	MS_Enable (Modem Sound Enable)

• **Table 8-2** Modem Connector Signal Descriptions

Pin Number	Signal Description
Pin 1	/Modem_Pwr – This active low signal comes from the power manager. For details on the use of this pin, see the section headed "Power Control Interface".
Pin 2	GND – Electrical ground.
Pin 3	/RTS – The Request to Send signal from the Macintosh Portable is ignored by the the modem since it is meaningless in full duplex operation. This pin is a no-connect on the modem.
Pin 4	/DCD – The behavior of the Data Carrier Detect pin depends on the state of the &C command.
Pin 5	RxD – Data received by the modem is sent to the Macintosh Portable over the RxD pin.
Pin 6	/CTS – Clear to Send is asserted whenever the modem is powered. This pin is grounded on the modem.
Pin 7	Modem_Sound – This is the analog sound output for the modem.
Pin 8	TxD – Data for the modem to transmit and commands for the modem comes from the Macintosh Portable over this serial pin.
Pin 9	/RI_EXT – This pin is used to signal the Macintosh Portable that a ring is present (see the section "Ring Detect Signal"). If the Macintosh Portable is in the sleep state, the assertion of this pin will cause the Macintosh Portable to return to the operating state and power-up the modem.
Pin 10	-5VDC – The -5V supply is guaranteed to be present whenever the /Modem_Pwr pin is asserted. However, this pin may float or go to ground anytime following the negation of /Modem_Pwr.
Pin 11	+5VDC unswitched – Whenever the Macintosh Portable has power available, this pin will supply +5.2 VDC ± 5% (see the section "Power Supply and Dissipation").
Pin 12	/DTR – The behavior of the Data Terminal Ready pin depends on the state of the &D command (see "&D: DTR Options" under "Command Definitions")

• **Table 8-2** Modem Connector Signal Descriptions (Continued)

Pin Number	Signal Description
Pin 13	V1 – This is the least significant bit of the three volume control bits. This pin may remain high following the negation of /Modem_Pwr.
Pin 14	V3 – This is the most significant bit of the three volume control bits. This pin may remain high following the negation of /Modem_Pwr.
Pin 15	V2 – This is the second bit of the three volume control bits. This pin may remain high following the negation of /Modem_Pwr.
Pin 16	/Modem_Ins – This pin is always asserted (i.e., ground on the modem) whenever the modem is installed in the Macintosh Portable.
Pin 17	/Modem_Busy – This pin is asserted whenever the modem is "busy." For details on the use of this pin, see the section "Power Control Interface."
Pin 18	MS_Enable – The modem asserts the modem sound enable pin (active high) whenever it has its sound monitor "on." Sound may be heard on the speaker of the Macintosh Portable if the modem drives Modem_Sound (pin 7) without asserting MS_Enable.

Analog output

The analog output (Modem_Sound) is as follows:

Output impedance:	5 Ω Max.
Output load :	500 Ω Max.
Output voltage:	Typ. ± 0.65 V p-p
	Max. ± 1.5 V p-p
Offset voltage:	Typ. ± 3 mV
	Max. ± 15 mV

Power supply and dissipation

The modem card operates from a ± 5 VDC ± 5% Macintosh Portable supply (battery or battery and charger). The total power consumption by the modem in a fully operational state is 500 mW Max / 450 mW Typical; in the sleep state the consumption is 0.3 mW when no ring is detected and 4.5 mW when a ring is detected. The power-up reset time is less than 500 ms.

Under normal operation, +5 VDC is supplied by pin 11; -5 VDC is supplied by pin 10. During the sleep state only +5 VDC is supplied.

Power control interface

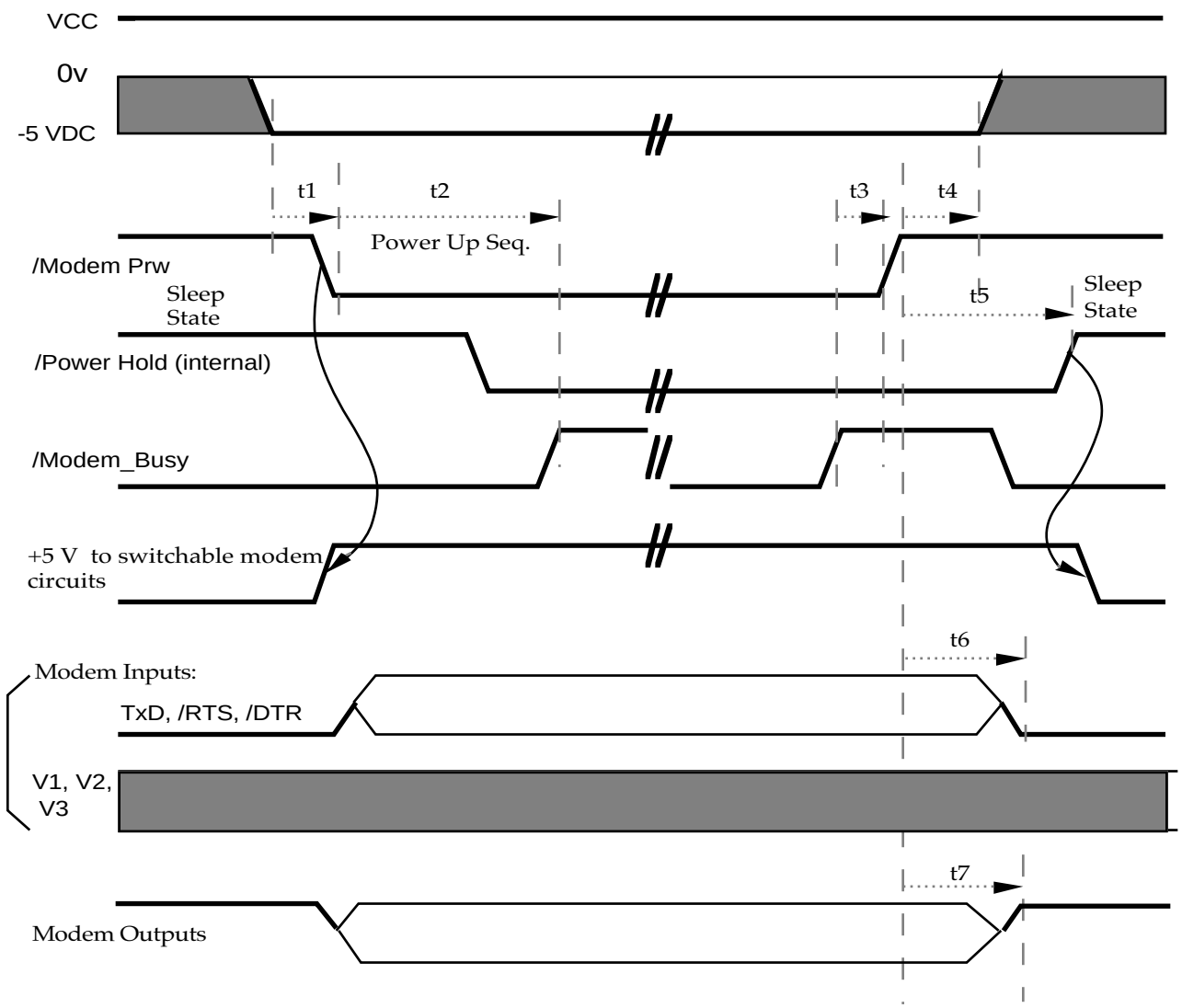
The modem is powered on or off through /Modem_Pwr and /Modem_Busy. The modem asserts /Modem_Busy whenever any of the following is true:

- the modem is executing its power-up sequence
- the modem is off-hook (for any reason)
- the modem is executing a command where "command execution" begins with the <CR> at the end of an AT command sequence or the "/" of the repeat last command sequence ("a/" or "A/")

If the modem is executing any of the self-tests, it is considered to be executing a command and therefore "busy."

The power manager processor controls the /Modem_Pwr signal (see the Power Up/Power Down Timing Diagram, Figure 8-2). If /Modem_Pwr is negated (high), the modem should immediately power-off regardless of what it is doing. The modem enters the sleep state within at most 500 ms following the negation of /Modem_Pwr (t5). Prior to sleeping, the modem forces all its outputs high (except pins 7, 16, and 18 which are driven low), stores its operating parameters and register values for restoration following sleep, and reduces its power consumption to meet the maximum sleep power limitation (see the previous section "Power Supply and Dissipation'). All inputs to the modem, except for the three volume bits and /Modem_Pwr, will be at ground within 50ns (t6) of the negation of /Modem_Pwr. The volume bits will always reflect the current volume setting.

- **Figure 8-2** Power Up/Power Down Timing Diagram



Time:	t1	t2	t3	t4	t5	t6	t7	t8
Min	0	–	0 *	0				
Max	–	500ms	–	–	500ms	500ms	500ms	

*** : t3 > 0 can be respected or not by the CPU**

Most often, the power manager will not negate /Modem_Pwr if the modem has /Modem_Busy asserted. However, there are times when the power manager must turn the modem off even though it is "busy." If this occurs, the modem stops its "busy" activity (e.g., goes on-hook) and performs the necessary activities in preparation for the sleep state. If it is executing a command when it is put into the sleep state, the modem should either complete the command or ignore it and restore the state prior to beginning execution of the command, whichever takes the least amount of time.

Ring detect signal

The /RI_EXT line (pin 9) is asserted during most of the AC cycle of a ring and is used to signal the Macintosh Portable CPU that a ring is present. Both ringing and pulse dialing will trigger the ring detector. The microprocessor in the modem distinguishes between a ring and pulse dialing by detecting the frequency of the signal. If the modem is powered down, the Macintosh Portable can determine whether /RI corresponds to a ring or pulse dial by powering up the modem and reading the appropriate register or looking for the "RINGING" result code.

	Rings	Pulse Dialing
Frequency	15–68 Hz	10 Hz
Duty Cycle	30–95 %	<20 %

The drive capability of the /RI output is:
V_{OH} = 2.8 V I_{OH} = -35 µA
V_{OL} = 0.5 V I_{OL} = 500 µA

Telephone network interface (Data Access Arrangement)

The telephone interface is a balanced, two-wire telephone interface design meeting FCC part 68 rules and DOC rules.

One eight-wire RJ-45 type jack is included and wired as follows:

Pin 4 for TIP signal
Pin 5 for RING signal

Pins 1, 2, 3, 6, 7, and 8 are not used.

The jack on the rear of the modem card is an RJ-11 type. This allows a common RJ-11 plug used on single-line telephone equipment to be be inserted, completing the connection of a phone to the modem.

Standards information for reference

The following compilations of signal characteristics are provided for reference only.

Compatibility and modulation

Standard	Speed (bps)	Modulation	Baud
CCITT V.22 bis	2400	QAM	600
CCITT V.22B	1200	DPSK	600
CCITT V.21	300/110	FSK	300/110
Bell 212A	1200 bps	DPSK	600
Bell 103	300/110 bps	FSK	300/110

Transmit carrier frequencies

V.22 bis/V.22/212A	Transmit Carrier
Originate	1200 Hz
Answer	2400 Hz

Bell 103	Mark	Space
Originate	1270	1070
Answer	2225	2025

V.21	Mark	Space
Originate	980	1180
Answer	1650	1850

Guard tone frequencies and transmit levels (CCITT only)

1800 Hz ± 20 Hz @ 6 ±1 dB below the transmit carrier level.
550 Hz ± 20 Hz @ 3 ±1 dB below the transmit carrier level.

Answer tone frequency

V.22 bis/V.22/V.21	2100 Hz
Bell 103/212A	2225 Hz

Received signal frequency tolerance

Offset frequency ± 7 Hz

Command Definitions

The modem implements the commands listed below. The commands can be either uppercase or lowercase characters. However, the attention characters must be either all uppercase or all lowercase (i.e., "AT" or "at"). The action in response to a command often depends upon the operating state of the modem, either local command (command) or on-line:

- local command state—In the local command state, *commands* are issued by the Macintosh Portable to set up the link parameters. The modem listens while a command is being entered and does not execute the command until the command terminating character, specified in register S3, is received. The only exceptions to this rule are the "A/" or "a/" command and the escape command.
- on-line state—The modem goes on-line after connecting with the remote terminal. In the on-line state, *data* is transmittted or received.

1. **AT**: Attention Code

All commands must begin with the attention code AT. If not set, the speed and parity of the Macintosh Portable interface are determined from this command. The remainder of the command line contains commands for the modem.

2. **A/**: Repeat Last Command

The A/ command instructs the modem to repeat the last command line. A/ is used in place of the AT command. A command termination character (register S3, default of <CR>) is not required for execution of this command.

3. +++: Return to the Local Command State (Escape Command)

The escape command, which consists of [Escape Guard Time (S12)][three escape characters (S2)][Escape Guard Time (S12)], is used to force the modem back to the local command state from the on-line state. The actual escape character is specified by register S2, expressed as the ASCII decimal value of the escape character (default is 43 = "+"). After the escape command is executed, the modem sends an "OK" result code to host. The escape guard time is the required delay time prior to and following the three escape characters. The guard time is determined by the value of the S12 register. The default setting is S12 = 50 = 1 second (one unit = 20 ms).

Example (S12 = 50, S2 = 43):
Wait at least one second after the last character entered.
Enter: +++
Wait at least one second for the modem's response.
Response from modem: OK

4. **A**: Enter Answer Mode

The "A" command forces the modem to go off-hook in the Answer mode and starts sending the answer tone, regardless of the value of register S0. If no carrier signal is received from the phone line within the number of seconds specified by register S7 (default of 30 seconds), the modem goes on-hook, sends the "NO CARRIER" result code to the Macintosh Portable, and returns to the local command state. This command must be the last command on a command line. Pressing any key aborts this command.

5. **Bn**: Communication Protocol Compatibility
The "B" command determines which protocol is used to connect at 300 or 1200 bps.
B0: CCITT V.21 or V.22 compatibility.
B1: Bell 103 or 212A compatibility (default).

The B command is ignored when the modem is communicating at 2400 bps.

[Cn command deleted]

6. **Ds**: Dialing

The dialing command takes the form Ds where s is a string of characters. In the simplest form, s consists of actual dial (or DTMF) characters. This set of dial characters for the modem consists of digits (0 to 9), “A”, “B”, “C”, “D”, “*”, and “#”. However, there are several dial modify commands that can be part of the dial string that allow the modem to perform special dialing functions. These additional commands can precede, be embedded in, or follow the actual number to be dialed.

7. **P**: Pulse Dial

The P command determines the use of pulse dialing The dialing speed is fixed at 10 pulses per second. The Make/Break ratio is determined by the AT&Pn command . Any dial characters following the P command are dialed using pulse dialing until the T command is issued. This command can appear anywhere in the dial string.

8. **T**: Touch-Tone Dial (DTMF)

The T command determines the use of touch-tone, or DTMF, dialing. The dialing speed is fixed at 70 ms for duration of tone and 70 ms for the blank period. Any dial characters following the T command are dialed using DTMF dialing until the P command is issued. This command can appear anywhere in the dial string.

9. **R**: Reverse Mode

The **R** command changes the modem from Originate mode to Answer mode after the dialing process is complete. This command is used only at the end of the dial string.

10. **, (comma)**: Pause

The comma (,) command introduces the delay time before dialing the next dial character or executing the next command in the dial string. The pause time is the value of the S8 register.

11. **W**: 2nd Dial Tone Detect

The W command is used to automatically detect the second dial-tone. If the second dial-tone has been detected by the modem before the S7 register time delay, the modem continues dialing the rest of the dial characters in the dial string. If no dial tone is received the modem goes on-hook, returns the "NO DIAL TONE" result code to the Macintosh Portable, and enters the local command state. This command is valid only when the result code command, X , is in X2 or X4 mode. This command can be embedded anywhere in the dial string.
Example: ATDT9W5551234

12. **@**: Wait for Quiet Answer Before Dialing

The @ command forces the modem to wait out the S7 register time delay for at least one ring followed by 5 seconds of silence (indicating the call has been answered) before continuing execution of the dial string. This command can be embedded anywhere in the dial string.

This command is used to access a system that requires entering additional dial characters after answering the initial call.
Example: ATDT5551234@9876543

13. **!**: Flash

This command causes the modem to go on-hook for 0.5 sec and then back off-hook, as if you had depressed the switch-hook button on the telephone set. This command can be placed anywhere in the dial string.

14. **; (semicolon)**: Return to Local Command State after Dialing

The semicolon (;) command must be put at the end of the dialing command. It forces the modem back to the local command state after dialing a number.

15. **En**: Echo Off/On

This command controls the echoing of characters sent to the modem from the Macintosh Portable back to the Macintosh Portable when the modem is in the local command state.
E0: Disable the echo of characters sent by the Macintosh Portable.
E1: Enable the echo of characters sent by the Macintosh Portable (default).

16. **Fn**: Half/Full Duplex

This command controls the echoing of characters sent to the modem from the Macintosh Portable back to the Macintosh Portable when the modem is in the on-line state.
F0: Enable the local echo when the modem is on-line.
F1: Disable the local echo when the modem is on-line (default).

17. **Hn**: Off/On Hook
H0: Force the modem on-hook (hanging up the phone).
H1: Force the modem off-hook (picking up the phone).

18. **In**: Information on Product Code/Checksum
I0: Requests the product code. The product code contains 3 digits in the form 24X (X= code revision number).
I1: Requests a checksum on the firmware. The first two digits are the checksum of the microcontroller ROM. The last two digits are the checksum of the DSP ROM.

This command is valid only when the modem is idle (not connected).

19. **Mn**: Speaker Off/Auto/On
M0: Disable speaker.
M1: Sets the speaker "ON" until carrier is detected (default).
M2: Sets the speaker always "ON".

20. **O**: Return to On-line State

The O command is used to return to the on-line state after using the escape command to enter the local command state.

21. **Qn**: Quiet/Vocal Mode
Q0: Sends the result codes to the Macintosh Portable (default).
Q1: Disables the sending of result codes to the Macintosh Portable.

22. **Sn?**: Check Contents of Register n

The Sn? command (n= register number) is used for checking the contents of a register and sending the value to the Macintosh Portable. The result is always expressed as a three-digit number, where the leading digits or all digits may be zero.

23. **Sn**=X: Set Register n to Value X

The Sn=X command is used to change the value of register n. To change the value of a register, the Macintosh Portable sends the string "ATSn=X" where n is the register number and X is the value to which the register should be set.

24. **Vn**: Numeric/Verbal Result Codes
V0: The modem sends the result code in numeric form.
V1: The modem sends the result code in verbal (English) form (default).

25. **Xn**: Active Result Codes
X0: Send the 0 to 4 result codes (compatible with Smartmodem 300).
X1: Send the 0 to 5 and 10 result codes (modem does not detect Busy, Ring back, or Dial tone signals).
X2: Send the 0 to 5, 6 and 10 result codes (all CONNECT result codes active and detection of dial tone before dialing occurs; modem does not detect Busy or Ring back signals).
X3: Send the 0 to 5, 7, 10, and 11 result codes (modem detects Busy and Ring back signals; Dial tone is not detected).
X4: Send all result codes (default).

Result codes are shown in Table 8-3:

- **Table 8-3** Result codes

Verbal	Numerical	Description
OK	0	Command completed
CONNECT	1	Connection established *
RING	2	Incoming ring detected
NO CARRIER	3	No connection or carrier drop
ERROR	4	Bad command
CONNECT 1200	5	Connection established at 1200 bps
NO DIAL TONE	6	Dial tone not detected in S7 seconds
BUSY	7	Busy tone detected
CONNECT 2400	10	Connection established at 2400 bps
RINGING	11	Ringback signal detected

Note

* The "CONNECT" result code, for X0, means a connection was established at 300, 1200, or 2400 bps. However, for X1 to X4, this result code means a connection was established at 300 bps.

26. **Yn**: Long Space Disconnect (Remote Disconnect)

This command allows the modem to disconnect if it receives a continuous BREAK (SPACE) signal for longer than 1.6 seconds from a remote modem. If the modem receives the long space, the modem will reply with a continuous BREAK (SPACE) for 4 seconds prior to going on-hook.
Y0: Disable the long space disconnect (default).
Y1: Enable the long space disconnect.

27. **Z**: Reset

The Z command performs a software reset and applies all of the default values to the other commands. This command affects the modem as if it experienced a power-on reset. Also, any commands remaining on the original command line after the Z command are ignored.

28. **&C**: DCD Options
&C0: DCD line (pin 4 of 18 pin interface) follows the actual carrier.
&C1: DCD line is always asserted (default).

29. **&D**: DTR Options
&D0: Modem ignores the \x\to /DTR line pin 12 (default).
&D1: Modem follows \x\to /DTR from the Macintosh Portable.

30. **&Gn**: Guard Tones

This command specifies whether a guard tone should be transmitted and, if so, what frequency should be used. Guard tones are used in some telephone systems to allow proper data transfer over the network. Guard tones are not used in the USA.
&G0: Disables guard tone (default).
&G1: Sets 550 Hz guard tone.
&G2: Sets 1800 Hz guard tone.

31. **&Ln**: Switched/Leased Line

This command affects the modem's use of timeouts during the handshake at the beginning of a connection.
&L0: Selects switched (dial-up) line (default). All timeouts are active according to the definition of the handshake protocols.
&L1: Selects leased line. All timeouts are ignored during the handshake process.

The &L1 setting may be useful for people using the modem on a leased line where no "calls" will be made or received on the line.

32. **&Pn**: Pulse Dial Make/Break Ratio
&P0: Make = 39%, Break = 61%; for use in the United States and Canada (default).
&P1: Make = 33% , Break = 67%; for use in United Kingdom and Hong Kong.

Dial Pulse specifications		&P0	&P1
Break ratio		61%	67%
Break length		61 ms	67 ms
Dial pulse rate		10 pps	10 pps
Dial pulse length		100 ms	100 ms
Interdigit time		789 ms	783 ms

33. **&Tn**: Self Tests

The following diagnostic tests are provided:

Modem type	Test Method
Bell 212A/103; CCITT V.21	Local Analog Loopback w/o Self-Test: &T1
	Local Analog Loopback with Self-Test: &T8
	Local Digital Loopback (no Self-Test mode): &T3
V.22 and V.22 bis	Local Analog Loopback w/o Self-Test: &T1
	Local Analog Loopback with Self-Test: &T8
	Local Digital Loopback (no Self-Test mode): &T3
	Remote Digital Loopback w/o Self-Test: &T6
	Remote Digital Loopback with Self-Test: &T7
	Respond to Remote Digital Loopback request: &T4
	Ignore a Remote Digital Loopback request: &T5

&T0: Terminate the Self-Test
Used to terminate loopback test modes using self-test pattern generation and error checking.

&T1: Local Analog Loopback

Initiates a Local Analog Loopback test. The escape sequence must be entered to terminate this test. This mode tests the local modem and the local data terminal equipment.

&T3: Local Digital Loopback
Initiates a Local Digital Loopback test. The modem echoes characters back to the Macintosh Portable exactly as received.

&T4: Enable the Remote Digital Loopback
Enables the modem to respond to a remote modem attempting to place it in the digital loopback test. If a remote modem places the local modem in the remote digital loopback mode, the local modem will echo characters back to the remote modem exactly as received from the remote modem.

&T5: Disable the Remote Digital Loopback
Disables the modem from responding to a remote modem attempting to place it in the digital loopback mode.

&T6: Remote Digital Loopback
Initiates a Remote Digital Loopback test. In this mode, characters sent to the remote modem are echoed back to the local modem exactly as they were received by the remote modem. This mode tests both local and remote modems and the telephone circuit.

&T7: Remote Digital Loopback with Self-Test
Initiates a Remote Digital Loopback with self-test data pattern generation and error checking.

&T8: Local Analog Loopback with Self-Test
Initiates a Local Analog Loopback with self-test data pattern generation and error checking.

&T9: Constellation Point Output / Data Output
Toggles between constellation point output and data output modes.

The tests using the internal self-test pattern (&T7 and &T8) end after the Macintosh Portable issues an &T0 or H0 command or the register S18 time delay expires. The self-test data pattern is an internally generated alternate binary ones-and-zeros signal. In the self-test modes, an error counter will count the number of errors and send the final result to the Macintosh Portable. The maximum number of errors that can be counted is 255.

34. %**A**: Read A/D Converter

Upon receiving this command, the modem returns the current value of the A/D converter in the analog front end. The result is a three digit decimal number.

Example: AT%A<CR> (command)
nnn (current ADC value, 3 digit decimal)
OK

35. %**Bnnn?**: Read Microcontroller RAM Location

This command causes the modem to read the microcontroller RAM location nnn, where nnn is the RAM address in decimal (range is 0 to 255), and return the value of the location as a three digit decimal number (rang 0 to 255).

Example: AT%Bnnn?<CR> (command)
mmm (Current data at RAM address nnn)
OK

36. %**Bnnn**=yyy: Write Data yyy to Microcontroller RAM

This command causes the modem to write the value yyy (3 digit decimal, 0 to 255) to the microcontroller RAM location nnn (3 digit decimal, 0 to 255).

Example: AT%Bnnn=yyy<CR> (Command)
OK

NOTE: This command must be used with caution. The microcontroller is actively using the RAM into which the value yyy may be written; disturbing this data may cause unpredictable results.

37. %**Cnnn?**: Read DSP RAM Location

This command causes the modem to read the word at the digital signal processing (DSP) RAM location specified by nnn (three digit decimal, 0 to 255) and return the value as two three-digit decimal numbers ranging 0 to 255. The first number represents the high data byte, and the second represents the low data byte.

Example: AT%Cnnn? (Command)
mmm (High DSP RAM data byte)
www (Low DSP RAM data byte)
OK

38. %**Cnnn**=xxx,yyy: Write data xxx,yyy to DSP RAM

Upon receiving this command, the modem writes the data xxx to the high data byte and yyy to the low data byte at DSP RAM location nnn. xxx, yyy, and nnn are all three digit decimal numbers ranging 0 to 255.

Example: AT%Cnnn=xxx,yyy<CR> (Command)
OK

NOTE: This command must be used with caution. The DSP is actively using the RAM into which the value xxx,yyy may be written; disturbing this data may cause unpredictable results.

39. %**Dnnn**: Set AGC Level

This command causes the automatic gain control (AGC) in the analog front end to be set the decimal value nnn (range 0 to 255).

Example: AT%Dnnn (Command)
OK

40. %**E**: Test Microcontroller and DSP RAM

This command causes the modem to test select portions of DSP RAM and all of microcontroller RAM using the "Triple Bit RAM Test Algorithm." The modem returns the result (OK or ERROR) for each of the RAM tests with the result of the DSP RAM test first. The response to this command is issued at 2400 bps after an effective reset of the modem to its factory default state.

Example 1: AT%E (Command)
OK (DSP RAM test passed)
OK (Microcontroller RAM test passed)

Example 2: AT%E (Command)
OK (DSP RAM test passed)
ERROR (Microcontroller RAM test FAILED)

Example 3: AT%E (Command)
ERROR (DSP RAM test FAILED)
OK (Microcontroller RAM test passed)

41. Missing Parameter

A missing parameter is evaluated as zero.
Example: ATF = ATF0

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and Microsoft® Word software. Proof and final pages were created on the Apple LaserWriter® printers. Line art was created using Adobe Illustrator™. POSTSCRIPT®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type and display type are Apple's corporate font, a condensed version of Garamond. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.