APPLE
PROGRAMMER'S
AND DEVELOPER'S
ASSOCIATION

290 SW 43rd. Street
Renton, WA 98055
206-251-6548

Technical Introduction to the Apple IIGS

APDA#: K2BGSI

Technical Introduction to the Apple IIGS

Final Draft
August 20, 1986

Apple Technical Publications

This document is in manuscript form. It does not include
- *final technical corrections*
- *final editorial corrections*
- *final art work*
- *index*

Simultaneously published in the United States and Canada.

Contents

Foreword

The Apple IIGS™ personal computer is an Apple II with many advanced hardware features, new firmware features, and a software **toolbox,** similar to the toolbox in the Macintosh™. To describe the many different aspects of the Apple IIGS, there are several technical books. Together, those books make up the Apple IIGS technical manual suite.

As the first book in the suite, this book, the *Technical Introduction to the Apple IIGS,* has several objectives. They are

- to describe the features of the Apple IIGS.
- to serve as a **delta guide** for purchasers of the Apple IIGS Upgrade for the Apple IIe.
- to explain the general design of the Apple IIGS.
- to introduce hardware designers to the Apple IIGS.
- to describe the different program environments in the Apple IIGS.
- to introduce programmers to the Apple IIGS toolbox.
- to introduce developers to the Apple IIGS Programmer's Workshop.

A **delta guide** is a description of something new in terms of its differences from something the reader already knows about. The name comes from the way mathematicians use the Greek letter *delta* (Δ) to represent a difference.

The Apple IIGS is an Apple II with a difference—or rather several differences. Those differences are particularly important to the person who purchases a Apple IIGS Upgrade, which adds the features of the Apple IIGS to an Apple IIe. By providing technical information about the added features, the *Technical Introduction* serves as a delta guide for the upgraded Apple IIe.

Where the *Apple IIGS Owner's Manual* describes the Apple IIGS from the point of view of the user, the *Technical Introduction* describes the Apple IIGS from the point of view of the application program. In other words, it describes the things the programmer has to consider while designing a program, such as the operating features the program uses and the environment in which the program runs.

Like Gaul, the set of all programmers starting out on the Apple IIGS is divided into three parts:

- programmers who are familiar with one or more computers in the Apple II family,
- programmers who are familiar with the Apple Macintosh, and
- programmers who are not familiar with either line of Apple computers.

The *Technical Introduction* addresses all three kinds of programmers. That means the book often describes features of the Apple IIGS that are also found on some other Apple machines and so are already familiar to some programmers. To make it easy to skip over such descriptions, they are labeled either as Apple II or Macintosh information.

Chapter 1 of the *Technical Introduction* starts by listing the features of the Apple IIGS, with emphasis on the new features that make the Apple IIGS more powerful than earlier models of the Apple II. It also contains a list of the features that provide compatiblity with those earlier models and a list of the features that resemble those of the Macintosh. The latter part of the chapter discusses aspects of the Apple IIGS that will be of interest to developers: the Apple IIGS Toolbox, the programming languages, the Apple IIGS Programmers Workshop, and the technical manuals.

The next three chapters describe the features listed in Chapter 1. Chapter 2 describes the hardware, Chapter 3 describes the I/O features (which involve both hardware and firmware), and Chapter 4 describes the other firmware.

Chapter 5 introduces an important new software feature of the Apple IIGS: the Toolbox, which is a set of built–in program tools. The Apple IIGS Toolbox supports the **desktop environment** and makes it easier for application programs to take advantage of the new hardware features.

The **desktop environment** is a set of program features that make user interactions with an application resemble operations on a desktop. The user selects objects or commands by using the mouse to move a pointer on the screen.

Chapter 6 explains the design of the Apple IIGS, including a summary of the machine's architecture and a description of its memory features.

Chapter 7 describes the different program environments on the Apple IIGS, that is, the different operating features and the way they are used by different types of programs.

Chapter 8 describes other programming issues such as program compatibility with earlier models of Apple II.

Chapter 9 introduces software developers to the Apple IIGS Programmer's Workshop (CPW), which is a complete development system including an editor, compiler, and linker.

The glossary lists technical terms used in this book. Some of the terms are also defined in marginal glosses near where they first appear in the text.

There are two appendixes. Appendix A, "Roadmap to the Apple IIGS Technical Manuals," tells about the other technical manuals and helps you decide which ones you need. Appendix B, "Summary of Program Environments," is a summary of information from Chapter 7, "Program Environments."

Notation and conventions

This manual has a few special ways of indicating a term or a piece of information that is different in some way.

New terms

The first time a specialized term appears in this manual it is printed in **boldface.** All such terms are defined in the glossary at the back of the book. Some of them are also defined in marginal glosses, as shown in the next section.

Some terms that you may already know are used a little differently in this book. You should be aware of these.

Apple II: Any of several computers in the Apple II family, which is made up of the Apple II, the Apple II Plus, the Apple IIe, the Apple IIc, and the Apple IIGS.

Standard Apple II: Any computer in the Apple II family except the Apple IIGS.

8–bit Apple II: Another way of saying standard Apple II. All those computers have 8–bit microprocessors.

64K Apple II: Any standard Apple II that has at least 64K of RAM. That includes the Apple IIc, the Apple IIe, and an Apple II or Apple II Plus with 48K of RAM and the Language Card installed.

128K Apple II: Any standard Apple II with both main and auxiliary 64K banks of RAM. That includes all models of the Apple IIc and some models of the Apple IIe, including those with the Extended 80–Column Text Card installed.

Special messages

Certain types of information are set off in special ways in this book. This is done in three ways: marginal glosses, notes, and important items.

A **marginal gloss** contains either a definition of a term appearing in bold–face in the text or a cross–reference either to another part of this book or to another book.

Note: A note like this usually contains information that is interesting but not necessary for an understanding of the main text. Notes have boldface labels such as **Note** or **Reminder.** Notes that provide background information about the Apple II or Macintosh family are labeled **Apple II** or **Macintosh.**

Important: An item like this—labeled **Important**—contains information that could keep you from causing the computer or its software to malfunction.

Chapter 1

Introduction to the Apple IIGS

The Apple IIGS personal computer is a high-powered addition to the Apple II family. Like Janus, the god of doorways, the Apple IIGS looks in two directions. First, it looks toward the future: with its many high-performance features, such as improved color display, advanced sound system, 16-bit microprocessor, and larger memory, the Apple IIGS makes it possible for future application programs to be more powerful. Second, the Apple IIGS looks toward the past: because it has the features of the Apple IIe and the Apple IIc, it can run most of the programs written for those computers.

The features of the Apple IIGS

The Apple IIGS has more features than any earlier Apple II. So that you can get an overall idea of what the Apple IIGS is, this section lists its features.

Note: The tables that follow are only summaries; to learn more about individual features, please keep reading. Chapters 2, 3, and 4 describe the hardware features, I/O features, and firmware features, respectively.

A more powerful Apple II

The easiest way to describe the Apple IIGS is to list all its features. In addition to the features of the Apple IIe and the Apple IIc, the Apple IIGS has many new features that set it apart from other models of the Apple II. Table 1-1 describes all the major features, both old and new.

Note: Specialized terms that appear in boldface are defined in the glossary.

Table 1-1. Features of the Apple IIGS

Feature	**Specification**	**Description**
More powerful microprocessor	**65C816**	16-bit microprocessor has 24-bit address and 6502 compatibility.
Faster operation	CPU clock speeds of 1 **MHz** and 2.8 MHz	User can select either of two speeds: the standard 1 MHz speed of the Apple II, or fast 2.8 MHz speed.
Larger memory	256K RAM, 128K ROM	Built-in memory includes the features of a 128K Apple II.
Memory expansion	24-bit address bus	Expansion card can expand RAM to as much as 8 megabytes.
Detached keyboard	78 keys	Separate keyboard includes cursor keys and numeric keypad.
Apple DeskTop Bus®	Low-cost serial I/O	Supports detached keyboard, mouse, and additional I/O devices.
RGB video	R, G, B, and sync	Provides both **analog RGB** and **NTSC** video outputs.
40- and 80-column text in color	Text, background, and border colors	Text, background, and border can be any of sixteen colors (only with RGB).
Apple II graphics	**Lo-Res, Hi-Res,** and **Double Hi-Res**	Standard Apple II graphics, including Double Hi-Res as on 128K models.
Super Hi-Res color graphics	True 320 x 200 or 640 x 200 resolution	Improved graphics with up to 16 colors per scan line and up to 256 colors on screen, out of 4096 possible colors.
Desktop user interface	Uses Super Hi-Res color graphics and mouse	Built-in Toolbox supports desktop interface with mouse, menus, and windows.
Improved sound	Ensoniq™ digital sound **IC** with 32 oscillators	Digital sampling synthesizer supports 15 independent voices. (Apple IIGS also retains single-bit sound used in other Apple II's, adds volume control.)
Control panel	Built-in desk accessory	User can set machine parameters for display, operating speed, serial ports, and disk drives.
Enhanced **Monitor**	Monitor in ROM	Handles 16-bit and 24-bit addresses, assembles and disassembles 65816 and 6502 instructions, performs 32-bit arithmetic; includes low-level I/O routines for display and keyboard.
Applesoft	Applesoft in ROM	Applesoft with modifications for lower-case and 80-column operation.
Built-in clock	Time and date	Clock has battery for continuous operation.

Feature	**Specification**	**Description**
Built-in serial ports	Two standard serial ports	Serial ports support modems, printers, and AppleTalk®. (User can still use serial card in slot.)
Built-in AppleTalk	Uses one serial port	No peripheral card required. User can select either serial I/O port to use for AppleTalk.
Built-in disk port	Disk I/O port using custom IC	User can select built-in disk port, disk interface cards in slots, or both, for as many as six drives at one time.
Expansion slots	Seven slots for peripheral cards	Expansion slots like those on Apple IIe. (Apple IIGS does not have auxiliary
slot.)		
Game I/O	External 9-pin jack, internal 16-pin socket	Supports all existing game hardware. (Some new devices use Apple DeskTop Bus instead.)

Apple II Compatibility

Even though the Apple IIGS has many powerful new features, it is important to remember that it is an Apple II. That means that most existing programs and peripherals as well as future programs developed for the Apple IIe and Apple IIc will run on the Apple IIGS.

The Apple IIGS has several features that make it compatible with earlier models of the Apple II. Table 1-2 is a list of those features, along with the other models of Apple II that also have them.

Table 1-2. Apple II features of the Apple IIGS

Apple IIGS feature	**Description**	**Other models**
6502 instruction set	65816 has emulation mode for running 6502 programs	All Apple II
128K RAM	Main and auxiliary 64K banks, with language-card and I/O spaces	IIc, 128K IIe
Applesoft in ROM	Applesoft BASIC interpreter with lower-case and 80-column features	All Apple II
Monitor in ROM	Supports low-level I/O and program development	All Apple II
40- and 80-column text	Black-and-white text displays (text in color only on Apple IIGS with RGB)	IIc, IIe with 128K or 80-column card
Lo-Res color graphics	48 x 40, 16 colors	All Apple II
Hi-Res color graphics	280 x 192, 6 colors	All Apple II
Double Hi-Res color graphics	560 x 192, 16 colors	IIc, 128K IIe
Built-in serial ports	Two RS-232-compatible ports, for modem, printer, other serial devices	IIc (similar)
Built-in disk port	Using **IWM** chip, supports both 5.25-inch and 3.5-inch disk drives	IIc
Expansion slots (7)	Slots for peripheral I/O and expansion cards, in addition to built-in ports	II , II Plus, IIe
Game I/O port	9-pin and 16-pin connectors for game paddles and sketch pads	IIc, IIe

Similarities to the Macintosh

Comparison of the hardware features of the Apple IIGS and the Macintosh will reveal more differences than similarities. Among the differences are: The Apple IIGS has a 65C816 microprocessor, while the Macintosh has a 68000; the Apple IIGS has a color display, the Macintosh is black-and-white; the Apple IIGS has slots, the Macintosh hasn't. On the other hand, while the two machines' operating systems are different, they both support hierarchical disk directories. And some of the hardware features are the same, such as the detached keyboard and the mouse.

While the Apple IIGS itself doesn't work like the Macintosh, applications on the Apple IIGS will bear a strong resemblance to Macintosh applications. The main reason is the use of the same **desktop user interface** on both machines. The built-in Apple IIGS Toolbox, like the Macintosh Toolbox, makes it easy for applications to support the desktop interface. Table 1-3 summarizes the major points of similarity, as well as some of the differences, between the two machines. The resemblances are described further in Chapter 5, "The Apple IIGS Toolbox."

In applications that use the **desktop user interface,** commands appear as options in pull-down menus, and material being worked on appears in areas of the screen called windows. The user selects commands or other material by using the mouse to move a pointer around on the screen.

Table 1-3. Apple IIGS compared with Macintosh

Feature	**Apple IIGS version**	**Macintosh version**
Desktop user interface	Pull-down menus; data in overlapping windows	Pull-down menus; data in overlapping windows
Desktop support for applications	Built-in Toolbox	Built-in Toolbox
Desktop display	Super Hi-Res color graphics	Bitmapped black-and-white graphics
Display resolution	640 x 200	512 x 342
Command selection	Apple mouse, keyboard optional	Apple mouse, keyboard optional
Keyboard	Detached, with keypad and cursor keys	Detached, with keypad and cursor keys on Macintosh Plus
Built-in serial ports	Two ports, using the Zilog SCC chip and RS-422 drivers	Two RS-232 ports, using the Zilog SCC chip
Built-in disk port	5.25-inch and 3.5-inch drives, using the Apple IWM chip	3.5-inch drives only
Operating system	ProDOS® (hierarchical files)	Hierarchical File System
External hard disk	Hard Disk 20SC with SCSI interface card	Hard Disk 20 (Hard Disk 20SC on Macintosh Plus)

For program developers

The Apple IIGS has several features that are important for developers. First of all, there is the Apple IIGS Toolbox, a collection of built-in program routines that can be called by applications. Then there is the program development environment, the Apple IIGS Programmer's Workshop (CPW), which includes the language compilers and their environment. With the built-in toolbox, the language compilers, the workshop programs, and the technical manuals that describe them, developers have everything they need to develop applications for the Apple IIGS.

The Apple IIGS Toolbox

Like the Macintosh, the Apple IIGS has a built-in toolbox whose routines can be called by applications. The toolbox routines include mouse operations with menus and windows to support the desktop user interface.

Not all of the tools are resident in ROM; some of them are loaded from disk and reside in RAM. The calling mechanism is the same regardless of where in memory a tool resides. A tool can even be in ROM in an early version of the system and in RAM in a later version; an application developed on the early version will run on the later version without modification.

The Apple IIGS Toolbox includes many functions like the ones in the Macintosh Toolbox, but they are not all the same. There are important differences between the machines, and those differences affect the nature and operation of the tools.

For a summary of the Apple IIGS Toolbox and more about the differences between the tools in the Apple IIGS and the Macintosh, please read Chapter 5, "The Apple IIGS Toolbox." For a complete description of the toolbox, see the *Apple IIGS Toolbox Reference,* Volumes 1 and 2.

Apple IIGS Programmer's Workshop

To provide a consistent working environment, there is the Apple IIGS Programmer's Workshop (CPW). The development environment consists of two kinds of programs: the compiler and assembler, which have their own reference manuals, and the workshop programs, which are all described in the *Apple IIGS Programmer's Workshop Reference.*

The Apple IIGS Programmer's Workshop is a set of programs that Apple provides to make it easier to develop applications for the Apple IIGS. The programs in the programmer's workshop are

- Shell
- Editor
- Linker
- Debugger
- Utilities

For more information about CPW, please see Chapter 9, "Apple IIGS Development Enviroument," and the manual *Apple IIGS Programmer's Workshop Reference.*

Apple IIGS programming languages

The languages available on the Apple IIGS include 65816 assembly language and C. Thanks to the standard object-file format on the Apple IIGS, the same linker and loader can handle program segments created in either of the available programming languages. Because the languages are available separately, there is a separate manual for each one.

The high-level language in CPW is C. Programs written in C can easily include sections written in assembly language and in Pascal. CPW C comes with a standard C library and a Apple IIGS interface library, which contains the tool calls.

The CPW Assembler is a full-featured macro assembler that supports the full 65C816 instruction set. (While the 65C816 instructions include those for the 6502 and the 65C02, the assembler is not an appropriate development tool for Apple IIs that use those microprocessors because CPW does not support Apple II binary load files.)

Note: The Apple IIGS has standard Applesoft BASIC in ROM for compatibility with other Apple II's.

For more information about programming on the Apple IIGS, please see Chapter 9, "Apple IIGS Development Environment," and then the individual manuals *Apple IIGS Programmer's Workshop Reference*, *Apple IIGS Workshop C Reference*, and *Apple IIgs Workshop Assembler Reference.*

The Apple IIGS technical manuals

To fully describe the Apple IIGS, Apple has produced a suite of technical manuals. There are manuals that describe the Apple IIGS computer itself and other manuals that describe the development tools. Table 1-4 lists the manuals by category. Depending on the way you intend to use the Apple IIGS, you may need to refer only to a few of the manuals, or you may need to refer to most of them.

For more informatiou that will help you decide which manuals you need, please see Appendix A, "Roadmap to the Apple IIGS Technical Manuals."

Table 1-4. Apple IIGS technical manuals

Category	Titles
Introductory manuals	*Technical Introduction to the Apple IIGS* *Programmer's Introduction to the Apple IIGS*
Machine reference manuals	*Apple IIGS Hardware Reference* *Apple IIGS Firmware Reference*
Toolbox manuals	*Apple IIGS Toolbox Reference,* Volumes 1 and 2
Workshop manual	*Apple IIGS Programmer's Workshop Reference*
Programming–language manuals	*Apple IIGS Workshop C Reference* *Apple IIGS Workshop Assembler Reference*
Operating-system manuals	*ProDOS 8 Technical Reference* *Apple IIGS ProDOS 16 Reference*
All-Apple manuals	*Human Interface Guidelines* *Apple Numerics Manual*

Chapter 2

Hardware Features

This chapter and the following two chapters describe the features of the Apple IIGS, with emphasis on the new features. This chapter covers the hardware features, Chapter 3 covers the I/O features, which combine elements of hardware and firmware, and Chapter 4 covers the rest of the firmware features.

Apple IIGS technology

The Apple IIGS is the most advanced Apple II to date. It uses several large-scale integrated circuits that are custom designed for it. These ICs are surface mounted on a four-layer printed circuit board mounted in the bottom of the case. By using large scale custom ICs, the designers of the Apple IIGS were able to increase the machine's capabilities with a minimum increase in manufacturing cost.

Table 2-1 lists the large ICs in the Apple IIGS, including the custom ICs. Some of those ICs are mentioned later in this chapter.

For detailed descriptions of the Apple IIGS ICs,
consult the *Apple IIGS Hardware Reference.*

Table 2-1. Large-scale ICs in the Apple IIGS

Name (abbreviation)	**Function**
Mega II	Provides the basic Apple II addressing and timing
Slotmaker	Provides additional address and control signals for the expansion slots
Fast Processor Interface (FPI)	Provides addressing and timing for fast memory; handles synchronization of processor and Mega II
Video Graphics Controller (VGC)	Provides video addressing and signal generation for Super Hi-Res display
Integrated Woz Machine (IWM)	Controller for 5.25-inch and 3.5-inch disk drives
Sound General Logic Unit (Sound GLU)	Provides interface between the system bus and the Digital Oscillator Chip (DOC)
Digital Oscillator Chip (DOC)	Digital sampling sound generator (made by Ensoniq)
Keyboard General Logic Unit (KeyGLU)	Provides interface between the system bus and the keyboard microprocessor
Keyboard Microprocessor (50740A)	Supports the Apple DeskTop Bus (interface to the detached keyboard, the mouse, and similar devices)

Microprocessor features

The microprocessor in the Apple IIGS is a 65C816 operating in conjunction with the custom FPI (Fast Processor Interface) chip. The 65C816 is a sixteen-bit **CMOS** design based on the venerable 6502. Table 2-2 lists its main features.

CMOS is an abbreviation for *Complementary Metal Oxide Silicon*, which is one of several methods of semiconductor integrated-circuit fabrication. CMOS devices are characterized by their low power consumption.

Table 2-2. Features of the 65C816 Microprocessor

16-bit accumulator

16-bit X and Y index registers

Relocatable zero page

Relocatable stack

24-bit internal address bus

8-bit data address bank register

8-bit program address bank register

11 new addressing modes

36 new instructions, for a total of 91 (all 256 **operation codes**)

Fast block-move instructions

Ability to emulate 6502 and 65C02 8-bit microprocessors

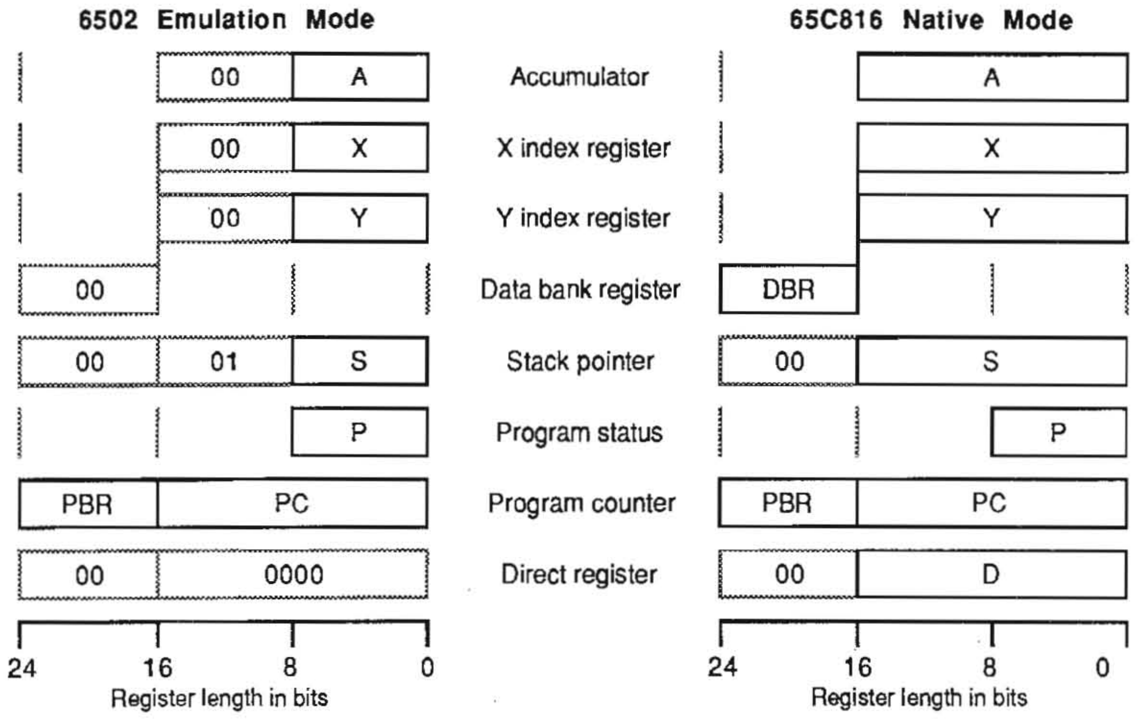
Sixteen-bit processor

In the Apple IIGS, the 65C816 normally operates in either of two modes: 6502 emulation mode and 65C816 native mode. Figure 2-1 shows the sizes of the registers in emulation mode and in native mode. In emulation mode, the accumulator and index registers are 8 bits wide, and existing Apple II programs run the same as they do on any other Apple II model. In native mode, the accumulator and index registers are sixteen bits wide. The 65C816 also has several new and more powerful addressing modes that take advantage of its 24-bit addressing. The new addressing modes operate in either native mode or emulation mode, although the shorter registers in emulation mode make some of them ineffective.

> **Note:** Native mode can also work with 8-bit data registers, with an additional accumulator, the B register. Apple does not recommend 8-bit native mode, but some internal routines use it, and developers are free to use it if they choose.

In Figure 2-1, the boxes represent registers, and the sizes of the boxes correspond to the number of bits in the registers, as indicated by the scales at the bottom of the figure. Letters in the boxes are the names of the registers; numbers (00, 01) in boxes indicate fixed values for those parts of the associated registers. For example, the stack pointer in native mode behaves like a 24-bit register with the upper eight bits permanently set to zero.

Figure 2-1. 65C816 registers



Two operating speeds

The Apple IIGS normally runs its 65C816 microprocessor at a clock rate of 2.8 MHz. For programs in RAM, the effective speed is about 2.5 MHz because the hardware allocates a few clock cycles for refreshing the RAM and cannot execute RAM instructions during the refresh cycles. Programs in ROM are not affected by RAM refresh, so they run at the full 2.8 MHz.

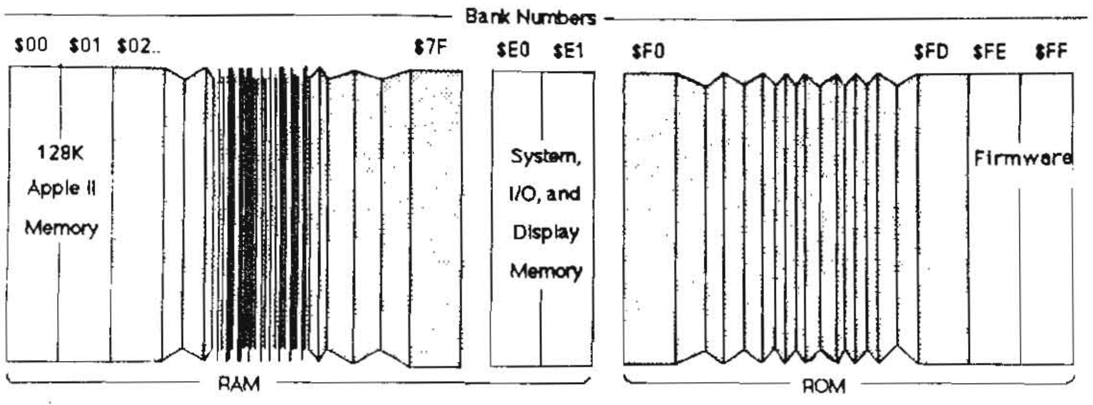
Almost all programs can run at the 2.5 MHz speed on the Apple IIGS, even programs originally written for an eight-bit Apple II. The Apple IIGS can also run at the normal Apple II clock rate, 1 MHz. There are three conditions that can cause the Apple IIGS to run at the 1 MHz speed:

- The user has selected normal speed on the Control Panel.
- A program is executing an instruction that uses 1 MHz memory (see the section "Memory on the Apple IIGS " in Chapter 6 for a description of 1 MHz memory).
- A timing-dependent routine is executing; for example, one in a disk interface card.

Memory features

Thanks to the 24-bit addressing of the 65C816, the Apple IIGS has a memory space totaling 16 megabytes. Of this total, up to 8 megabytes of memory is available for RAM expansion, and one megabyte is available for ROM expansion. Figure 2-2 is a simplified version of the Apple IIGS memory map.

Figure 2-2. Simplified Apple IIGS memory map



(marginal gloss) This book uses **hexadecimal numbers** for memory addresses. The dollar sign before a number signifies that the number is hexadecimal.

The internal memory of the Apple IIGS has two main features: it can emulate the main and auxiliary memory banks of a 128K Apple II, and it can be expanded up to as much as 8.25 megabytes. The next two sections describe these features.

For additional information about memory on the Apple IIGS, read the section on memory in Chapter 6.

The letter *K* is the abbreviation for *kilo-*, meaning *thousand*. In this book, *K* stands for ***kilobyte*** (1024 bytes) except when dealing with memory ICs, when it stands for ***kilobit***—1024 bits.

Apple II main and auxiliary memory

Apple II: This section describes the way memory is used in all models, including the Apple IIe and IIc. If you are already familiar with those machines, you might want to skip ahead to the next section.

The 6502 microprocessor used in the original Apple II can address up to 64K bytes of memory. The Apple IIc and the 128K versions of the Apple IIe have 128K of memory, which they address in two 64K banks. To distinguish the two banks, the original 64K of memory is referred to as *main memory* and the additional 64K as *auxiliary memory*. In the Apple IIGS, banks $00 and $01 work like main and auxiliary memory when running programs written for the Apple IIe and Apple IIc.

In the original Apple II and the Apple II plus, different parts of the 64K memory space are allocated for different purposes. Built-in ROM occupies the highest addresses, from $D000 to $FFFF. Addresses between $C000 and $CFFF are allocated to built-in I/O and to the peripheral slots for I/O devices and ROM on peripheral cards. Applications use memory in the 48K of space below $C000, except for the video display buffers, which are called ***pages***. There are two text display pages and two Hi-Res graphics pages; Table 2-3 shows their locations.

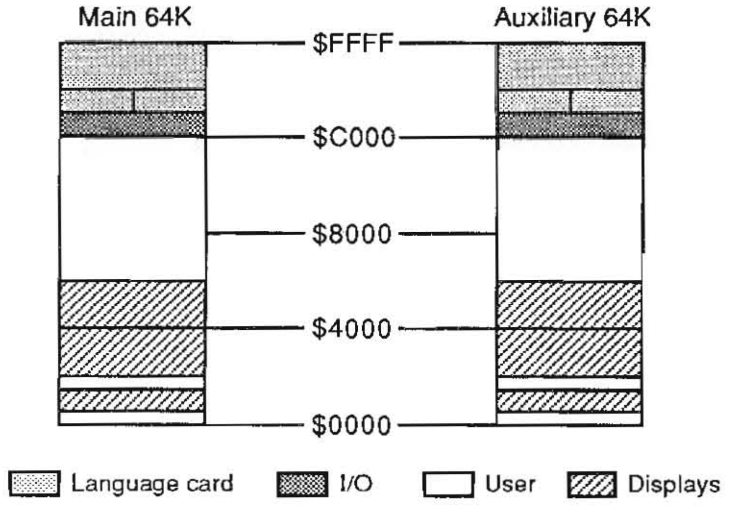
Table 2-3. Standard Apple II display pages

Display page	**Memory locations**
Text Page 1	$0400 - $07FF
Text Page 2	$0800 - $0BFF
Hi-Res Page 1	$2000 - $3FFF
Hi-Res Page 2	$4000 - $5FFF

When Apple introduced UCSD Pascal for the Apple II, the "lower forty-eight" kilobytes of memory was insufficient, so Apple added an expansion card with 16K of RAM. The RAM expansion card was part of the Pascal language package for the Apple II, so it was called the Language Card. To make 16K of RAM addressable without disturbing the memory-mapped I/O in the $Cxxx space, Apple designed the card with two 4K banks at $Dxxx. In the Apple IIe and the Apple IIc, the entire 64K of main RAM is installed on the main circuit board, but the peculiar addressing of the upper banks is retained for the sake of compatibility. Apple still refers to RAM memory between $D000 and $FFFF that has two banks in the $Dxxx space as language-card memory, even when it is on the main board.

(marginal gloss) The letter *x* in an address stands for the range of all possible values for that digit. For example, $Dxxx means all the addresses from $D000 through $DFFF.

Figure 2-3 . 128K Apple II memory map



The technique for addressing the auxiliary 64K memory space also involves switching banks, independently of the language-card bank switching. In fact, the auxiliary memory has its own language-card space, complete with two banks at $Dxxx. (The I/O space, $Cxxx, is the same in both main and auxiliary memory.)

Figure 2-3 is the memory map for an Apple IIc or 128K Apple IIe, showing the two 64K memory banks and the language-card banks above $C000.

If you are interested in learning more about the workings of the Apple II, you should look at the *Apple IIe Technical Reference Manual.*

Memory expansion

The minimum memory in the Apple IIGS is 256K. Apple II programs use 128K of that, mapped as main and auxiliary memory; the system firmware uses parts of the other 128K. Programs written for the Apple IIGS—that is, programs that run the 65C816 microprocessor in native mode, thereby gaining the ability to address more than 128K of memory—can use up to about 176K of the 256K. The rest is reserved for displays and for use by the system firmware.

The Apple IIGS also has a special card slot dedicated to memory expansion. All the RAM on a memory expansion card is available for Apple IIGS application programs that call the **Memory Manager.** Expansion memory is contiguous: its address space extends without a break through all the RAM on the card. Unlike the Apple IIe, expansion RAM on the Apple IIGS is not limited to use as a RAM disk; program code can run in any part of RAM.

The **Memory Manager** is part of the Toolbox. Its job is to allocate memory so that applications and desk accessories can run without clobbering each other.

Note: The memory expansion slot on the Apple IIGS is not like either the expansion slots or the auxiliary slot on the Apple IIe. Memory expansion cards designed to run in either of those slots will not work in the memory expansion slot. (A memory expansion card designed to run in an Apple II expansion slot will run in an expansion slot in the Apple IIGS.)

There can be several different sizes of memory expansion card for the Apple IIGS. Using presently-available 256K (**kilobit**) RAM chips, a memory expansion card can have up to a megabyte of additional RAM. When one–megabit RAM chips become available in quantity, a memory expansion card can have up to four megabytes of RAM. (The Apple IIGS will accept expansion RAM up to eight megabytes.) The additional RAM maps into contiguous 64K banks starting with bank $02, as shown earlier in Figure 2-2.

K means **kilobyte** except when it means **kilobit,** 1024 bits. Similarly, a **megabit** is 1024 kilobits, and a **megabyte** is 1024 kilobytes.

In addition to expansion RAM, the memory expansion cards can also have up to a megabyte of ROM. The additional ROM occupies memory from bank $FD downward to bank $F0. Portions of the top two banks of expansion ROM are allocated for system firmware expansion. The remaining expansion ROM is supported as ROM disk—permanent storage for applications, which the system handles like disk files. For additional information about memory, see Chapter 6.

Display features

To start with, the Apple IIGS has the standard Apple II video modes, both graphics and text, and the text display is enhanced with a choice of colors for borders, text, and background. In addition, the Apple IIGS has built–in RGB video and two new Super Hi–Res graphics modes.

RGB and Composite Video

The Apple IIGS has both **RGB** and composite (**NTSC**) video outputs. Either type of video monitor can be used with the Apple IIGS, although an RGB monitor is required for 80–column text in color.

RGB is an abbreviation for **red–green–blue**, a way of displaying color video by transmitting the three primary colors as three separate signals. With **TTL RGB**, only a few colors are possible; with **analog RGB**, the color signals can take on any values between their upper and lower limits, for a wide range of colors.

NTSC is the abbreviation for *National Television Standards Committee* and refers to a method for transmitting color video information for home television receivers. That method is also called *composite* because it combines all the video information, including color, into a single signal.

Note: A monochrome monitor will work on the Apple IIGS. All the user has to do is connect it to the composite video output jack and use the control panel to set the display type to monochrome.

The RGB video from the Apple IIGS is **analog RGB**. With an appropriate RGB monitor, the Super Hi–Res mode can display sharp graphics with any of 4096 colors. For the sake of compatibility with programs that generate graphics for composite monitors, the Hi–Res and Double Hi–Res displays on the Apple IIGS look like composite video even on an RGB monitor.

Historical note: At one time, Apple provided an RGB adaptor card and an RGB monitor, the AppleColor 100, for the Apple II. Using that system, Hi–Res and Double Hi–Res color displays were restricted to a horizontal resolution of only 140, a restriction that does not apply to the Apple IIGS. Note that an AppleColor 100 Monitor requires separate digital signals, so it will not work on the Apple IIGS.

Text with color

The standard video modes on the Apple IIGS include three enhancements: colored text, colored background, and colored border. For displaying 40–column or 80–column text on an RGB monitor, the user can select any of sixteen standard colors for text and any other of the sixteen colors for background. (The Control Panel won't let the user set the text and background colors the same.) Any of the sixteen colors can be used for the border, that is, the visible part of the display outside the area used for text and graphics.

Note: Colored text works only with an RGB monitor. The composite video output automatically switches to monochrome for text displays, making the text, background, and border colors appear as black, white, or shades of gray. This feature reduces color fringing and improves the legibility of text displayed on composite color monitors.

By the way: The unused border around the video display is wide enough that information on the edge of the display won't be lost when viewed on video monitors with their picture size controls set too big.

Apple II graphics

Apple II: This section describes graphics features found on many other models of the Apple II. If you are already familiar with the Apple II, you might want to skip ahead to the section "Super Hi-Res Graphics."

The Apple IIGS includes the same graphics displays found on the Apple IIc and 128K Apple IIe: Lo–Res, Hi–Res, Double Lo–Res, and Double Hi–Res. Table 2–4 shows the specifications for these displays.

Table 2–4. Apple II graphics displays

Display mode	Resolution	Number of colors	Restrictions
Lo–Res	40 x 48	16	(none)
Hi–Res	280 x 192	6	Some colors cannot appear side–by–side in small areas of the display.
Double Lo–Res*	80 x 48	16	(none)
Double Hi–Res*	560 x 192	16	(none)

*Not supported by firmware.

Like all other Apple II's, the Apple IIGS displays Lo–Res and Hi–Res color graphics. Applesoft BASIC, in ROM, includes simple routines for setting colors and drawing dots and lines. The Apple IIGS also has the double graphics modes, but, like other Apple IIs, it doesn't have graphics firmware for those modes.

Note: For the standard graphics modes—Lo–Res, Hi–Res, and Double Hi–Res—the Apple II uses a simple trick to generate color on a composite monitor. The individual dots in the graphics are spaced just right to stimulate the circuits that the monitor uses to extract color information from a composite signal. (In Lo–Res, the large dots in the display are made up of smaller dots that blend together on the screen.) Different combinations of dots make different colors.

Super Hi–Res graphics

In addition to the standard video modes found on the Apple IIc and Apple IIe, the Apple IIGS also has two new Super Hi–Res graphics modes. The new display modes take advantage of the analog RGB video output to produce high–quality, high–resolution color graphics. Table 2–5 lists the specifications of the two new graphics display modes.

Table 2–5. Super Hi–Res graphics modes

	Resolution:		Bits per	Colors	Colors	Colors
Mode	Horiz.	Vert	pixel	per line	on screen	possible
320	320	200	4 bits	16	256	4,096
640	640	200	2 bits	16*	256*	4,096

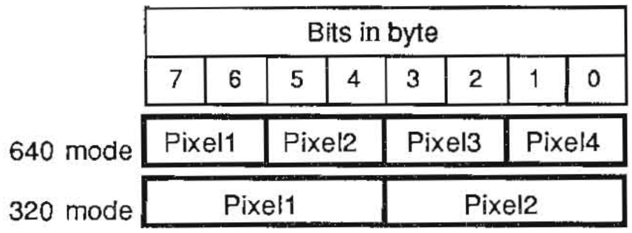
*Different pixels use different parts of the palette.

Pixel is short for ***picture element.*** A pixel corresponds to the smallest dot you can draw on the screen.

In the new Super Hi–Res graphics modes, colored dots have the same horizontal resolution as black–and–white dots. (That's different from the standard Hi–Res and Double Hi–Res graphics modes, where colored dots are effectively wider than black–and–white dots.) Each dot on the Super Hi–Res screen corresponds to a pixel, and pixels are indivisible: the screen does not display individual bits.

Each pixel has either a 2–bit (640 mode) or a 4–bit (320 mode) value associated with it, as shown in Figure 2–4. The pixel values select colors from programmable color tables called *palettes*. A palette consists of sixteen entries, and each entry is a 12–bit value specifying one of 4,096 possible colors. In 320 mode, color selection is quite simple: each pixel consists of four bits, so it can select any one of the sixteen colors in a palette.

Figure 2–4. Bits in pixels



In 640 mode, color selection is more complicated. The 640 pixels in each horizontal line occupy 160 adjacent bytes of memory, and each byte holds four pixels that appear side–by–side on the screen. The sixteen colors in the palette are divided into four groups of four colors each. The first pixel in each horizontal line can select any one of four colors from the third group of four in the palette. The second pixel selects from the fourth group of four colors in the palette. The third pixel selects from the first group of four colors, and the fourth pixel selects from the second group, as shown in Figure 2–5. The process repeats for each successive group of four pixels in a horizontal line. Thus, even though a given pixel can be one of only four colors, different pixels in a line can take on any of the sixteen colors in a palette. Using a technique called **dithering,** software for 640 mode can take advantage of this color–selection scheme to display 16–color graphics on the same screen with 80–column text.

Dithering is a technique for alternating the values of adjacent pixels to create the effect of more colors.

Figure 2–5. Color selection in 640 mode

Pixel	Value	Palette
Pixel3	0	Color1
	1	Color2
	2	Color3
	3	Color4
Pixel4	0	Color5
	1	Color6
	2	Color7
	3	Color8
Pixel1	0	Color9
	1	Color10
	2	Color11
	3	Color12
Pixel2	0	Color13
	1	Color14
	2	Color15
	3	Color16

To further increase the number of colors available on the display, there can be as many as sixteen different palettes in use at the same time. Each of the 200 horizontal lines of pixels can use any one of the palettes, giving as many as 256 different colors at once. All the palette information occupies memory adjacent to the display data; a picture and its palette are normally saved together.

Note: In 320 mode, there is a graphics fill option that enables a program to fill any portion of a horizontal line with a new color simply by setting marker values on the boundaries of the fill area. Because individual windows usually don't control the entire width of the screen, this technique is not useful in a window environment. On the other hand, if you are writing a graphics package that uses the entire screen, you might want to consider using it.

Sound capabilities

The Apple IIGS has more powerful sound–generating circuits than any previous Apple computer, although programs that generate sounds with the single–bit sound output of earlier models of the Apple II will still work on the Apple IIGS.

Single–bit sound

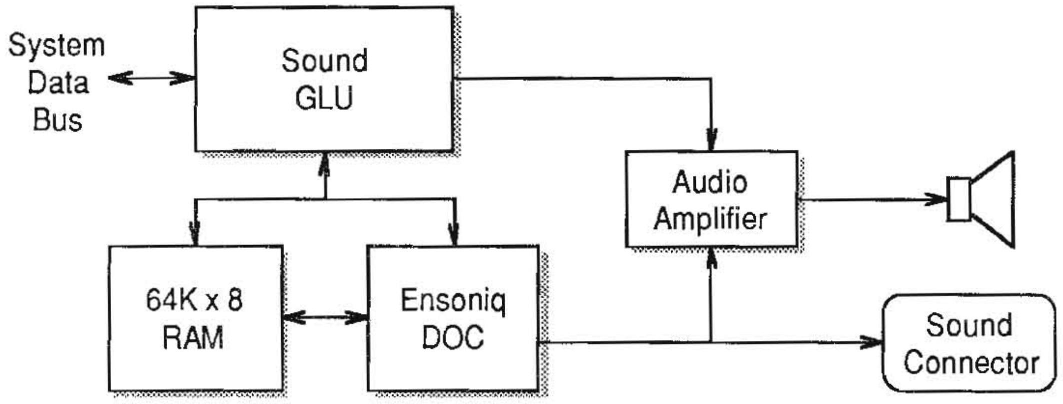
The standard Apple II sound output consists of a single bit, and programs produce sounds by switching that bit on and off. In the Apple IIGS, you can also adjust the volume of the sounds generated this way, by using the Control Panel or by making a call to the sound tools.

Digital synthesizer

In addition to the old single–bit sound output, the Apple IIGS has a new digital sound system that includes a special–purpose synthesizer IC called the Digital Oscillator Chip, or **DOC** for short. The DOC, which is made by Ensoniq and used in their line of music synthesizers, generates sound waveforms from digital samples stored in RAM. Using the DOC, the Apple IIGS can produce multi–part, multi–voice music and other complex sounds without tying up its main processor.

Figure 2–6 is a block diagram of the sound system of the Apple IIGS. The sound system consists of the DOC, an audio amplifier and internal speaker, a connector for an external amplifier and speaker, 64K of independent RAM for storing sound samples for the DOC, and a custom IC, the Sound **GLU** (General Logic Unit). The Sound GLU chip functions as the system interface to the DOC; in addition, it gives the Apple IIGS the ability to control the volume of sound from the old–style single–bit output.

Figure 2–6. Apple IIGS Sound System



The DOC contains thirty–two individual oscillators, each of which generates a signal by stepping through a table of digital samples of a sound. In the Apple IIGS, one oscillator is used as a dedicated clock for the DOC and one is reserved for future use, leaving thirty. Even though each oscillator can produce sound independently, it takes two oscillators to produce a continuous instrumental voice, so in normal use the DOC can produce up to fifteen voices.

The DOC also has a single analog–to–digital converter (ADC). If a properly–conditioned audio signal is connected to the input to the ADC, the DOC can record digital samples of sounds for later playback by the DOC's oscillators. (You can condition the signal by using a low–pass filter with a cutoff no higher than 14kHz or by adding a sample–and–hold circuit that is synchronized to the DOC's clock.)

Refer to the *Corltand Hardware Reference* for details about the sound system and the DOC.

Built–in clock

The Apple IIGS has a built–in real–time clock with battery back–up during power interruptions. The user sets the time and date by means of the Control Panel. ProDOS uses the clock to set date and time in files.

Note to Developers: The Apple IIGS clock does not use the same commands as the various third–party clock peripherals. Applications can call ProDOS and get the time the same way as on an Apple IIe, or they can determine which system they are running on and use the calls appropriate to the clock on that system.

Chapter 3

I/O features

This chapter describes the I/O features, which have both hardware and firmware aspects. As in Chapter 2, the emphasis is on the new features. You can find further descriptions of the I/O features in the manuals *Apple IIGS Hardware Reference* and *Apple IIGS Firmware Reference*.

I/O expansion slots

Except for the Apple IIc, all models of the Apple II have I/O expansion slots. The original Apple II and the Apple II Plus had eight slots, numbered 0 through 7. The Language Card normally occupied slot 0 on those machines. On later models, including the Apple IIGS, the language–card memory is built in and there are only seven slots, numbered 1 through 7.

Slots on the Apple IIGS

The I/O expansion slots are designed to accept circuit cards that contain hardware and firmware to control and communicate with peripheral devices. The slots are not simply I/O ports; a card in a slot has access to the clock and control signals, the data bus, and the low–order sixteen bits of the address bus. The same signals are available on all the slots, except for the color subcarrier, which is only on slot 7. In addition to the common signals, each slot has its own select signals, which are separately decoded for each slot. The slots on the Apple IIGS are almost identical to the slots in an Apple IIe, and can accept most Apple II peripheral cards. (Two of the slot signals, Inhibit and Sync, work differently on the Apple IIGS, and there is a new signal, M2Select; please refer to the *Apple IIGS Hardware Reference* for more information.)

As far as the slots themselves are concerned, any peripheral card can operate in any slot. However, it has become conventional to use certain cards in certain slots: for example, printer interface in slot 1, 80–column display in slot 3, and disk controllers in slots 5 and 6. Even though later models of Apple II have these I/O interfaces built in, compatibility requires them to have the same kind of program access that was originally designed for cards in slots.

> **Peripheral–card compatibility:** Only the low–order sixteen bits of the 24–bit address bus are available on the expansion slots. Peripheral cards that derive their enabling signals by decoding the address bus will not work in the Apple IIGS unless they also use one of the select signals to verify that the address on the bus is in the appropriate 64K bank for I/O (that is, bank $E0).

Apple II slot memory

This section briefly describes the memory spaces allocated to the slots. Except for their location in bank $E0 and the consequent need for shadowing to be on for old–style programs to work, the slot memory locations in the Apple IIGS are the same as on any other model of the Apple II. If you need all the details about slot memory, refer to the manual *Apple IIGS Hardware Reference*.

Shadowing for I/O: In the Apple IIGS, I/O uses memory locations in bank $E0. To make those locations available to 6502–based Apple II programs, which run in banks $00 and $01, the Apple IIGS has a feature called I/O shadowing that makes *load* and *store* instructions to locations in bank $00 also happen in bank $E0. For more information about shadowing, see the section "Memory Shadowing" in Chapter 6.

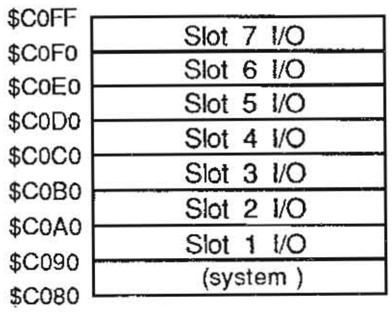
Apple II: The rest of this section describes the way the expansion slots work on all models of the Apple II. If you are already familiar with the Apple II, you might as well skip ahead to the next section.

The microprocessor in an Apple II does all its I/O through memory locations. To make the slots accessible to the processor, parts of the memory space are allocated to the slots. In addition to the memory locations used for actual I/O, there are memory spaces for programmable memory (RAM) and for read–only memory (ROM) on the cards, as described below.

Slot I/O space

Each expansion slot has the exclusive use of 16 memory locations for data input and output. The 16 locations for a given slot have addresses $C08x + s0, where *x* stands for hexadecimal values from 0 to F and *s* stands for the slot number. Figure 3–1 shows the allocation of I/O addresses for the slots: for example, the I/O addresses for slot 3 are $C0B0 – $C0BF. Whenever one of those addresses appears on the address bus, the slot hardware activates the device select signal in that slot. The circuits on the card can use the device select signal and the low–order 4 bits of the address to activate devices on the card.

Figure 3–1. Slot I/O device locations



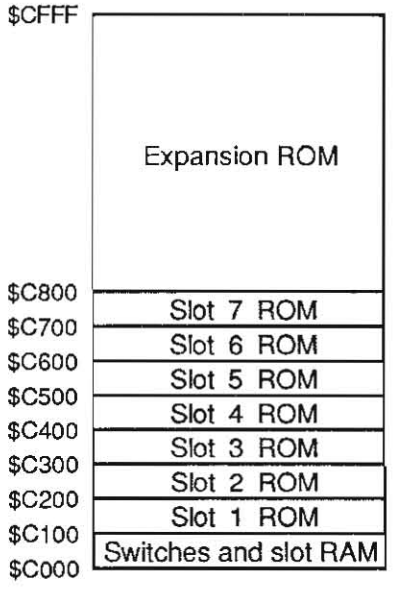
Slot ROM space

Each expansion slot has the exclusive use of one 256–byte page of memory space. Most peripheral cards use this space for ROM or **PROM** for storing the driver routine that controls the operation of the peripheral device.

PROM stands for *Programmable Read–Only Memory*, a type of ROM device designed to be programmed after fabrication, unlike ordinary ROM devices, which are programmed during fabrication.

The 256 ROM locations for a given slot have addresses $Cs00, where *s* stands for the slot number. Figure 3–1 shows the allocation of ROM addresses for the slots: for example, the ROM addresses for slot 3 are $C300 – $C3FF. Whenever one of those addresses appears on the address bus, the slot hardware activates the I/O select signal in that slot. That signal enables the ROM device on the card, and the low–order 8 bits of the address bus determine which of the 256 locations is being addressed.

Figure 3–2. Slot ROM locations



Expansion ROM space

In addition to the small areas of memory allocated to each slot, there is a 2K memory space from $C800 to $CFFF that can be used by a card in any slot. More than one peripheral card can have expansion ROM on it, but only one of them can be active at one time.

Each card that has expansion ROM on it must also have a circuit that uses the I/O select and I/O strobe signals on the slot to enable the ROM. The card must also have a circuit to disable the ROM so that other cards can use the same addresses for their expansion ROM.

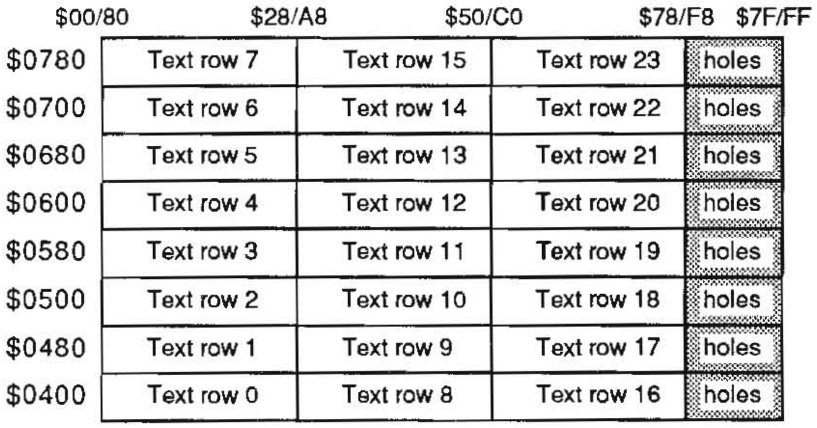
For more details, refer to the chapter "Programming for Peripheral Cards" in the *Apple IIe Technical Reference Manual.*

Slot RAM space

Besides the various locations allocated for devices on peripheral cards, a few locations in main memory are reserved for variables used by the peripheral–card routines. These locations are called the *screen holes*. Each slot gets one byte in each of the eight small blocks of text–page memory, as shown in Figure 3–3. To determine the addresses of the eight RAM locations assigned to a particular slot, add the slot number to the starting addresses of the blocks. For example, the RAM locations for slot 1 are $0479, $04F9, $0579, $05F9, $0679, $06F9, $0779, and $07F9.

Screen Holes: The text display buffer (text Page 1) occupies memory from $400 to $7FF, but there are locations in that range that are neither displayed nor modified by the firmware's display subroutines (for example, COUT1). Those locations are called the screen holes, and are used for temporary storage either by I/O routines running in peripheral–card ROM or by firmware routines addressed as if they were in card ROM. (Application programs never use this area of memory.)

Figure 3–3. Screen hole locations



Finding the slot number

The ROM routines on a peripheral card often need to know which slot the card is in. One way to do this is to execute a JSR (jump to subroutine) instruction to a location with an RTS (return from subroutine) instruction in it, then get the return address from the stack and derive the slot number from that, using the formula given above in the section "Slot ROM space."

The *Apple IIGS Hardware Reference* and the technical reference manuals for the Apple IIe and the Apple IIc describe in detail how a peripheral–card routine goes about determining its slot number.

Serial I/O ports

The Apple IIGS has two built–in serial ports that can substitute for slots 1 and 2. By using the Control Panel desk accessory, the user can select either the built–in port or the card for either slot. A built–in port can operate while there is a peripheral card plugged into the corresponding slot, but the port and the card cannot both run at the same time.

The *Apple IIGS Owner's Guide* gives a complete description of the use of the Control Panel.

The hardware for the serial ports consists of a two–channel Serial Communications Chip (Zilog 8530) and RS–422 driver ICs. The firmware for the ports emulates the functions of the Super Serial Card and the Apple IIc serial–port firmware. The firmware provides input and output buffering as well as background printing, as described below.

The ports are normally configured such that port 1 is a printer port and port 2 is a communications port, but either port can be configured either way by using the Control Panel desk accessory. (Alternatively, the user can connect either one of the ports to AppleTalk: see the section "AppleTalk interface" later in this chapter.)

Apple II serial ports

Apple II: This section describes the way the serial ports work in other models of the Apple II. If you are familiar with the operation of the Apple Super Serial Card or the serial ports on the Apple IIc, you might as well skip ahead to the section "New Serial–Port Features."

This section describes the basic functions of the serial I/O ports. Those functions are the same on the Apple IIGS as on other Apple II's with built–in ports, even though their hardware implementation is different. For complete descriptions of the serial ports, refer to the manual *Apple IIGS Firmware Reference*.

Both serial ports are general–purpose I/O ports, compatible with RS–232 standard devices. Serial port 1 is initalliy set up as an output port for a printer or plotter, and serial port 2 as a communications port for a modem: Table 3–1 shows the settings. The user can change the characteristics of either port by using the Control Panel desk accessory. An application can change port characteristics by means of commands, as summarized in Table 3–3 and described fully in the *Apple IIGS Firmware Reference*.

Table 3–1. Initial settings for serial ports

Characteristic	**Port 1**	**Port 2**
Line length	unlimited	unlimited
Delete line feed after carriage return?	no	no
Add line feed after carriage return?	yes	no
Echo output to display screen?	no	no
Buffering on?	no	no
Data transmission rate	9600 baud	1200 baud
Number of data bits	8	8
Number of stop bits	1	1
Type of parity checking	none	none
DCD–type handshaking enabled?	yes	yes
DSR/DTR handshaking enabled?	yes	yes
XON/XOFF handshaking enabled?	no	no
Command character*	Control–I	Control–A

***Note:** The Control Panel doesn't change the command character. You change the control character by sending the current command character followed by a control character, which becomes the new command character. For more information, see the following section.

DCD, DSR, and **DTR** stand for *data carrier detect, data set ready,* and *data terminal ready,* respectively, which are names of signals on the serial ports. **XON** and **XOFF** are two control characters. I/O driver routines use those signals or those characters for **handshaking,** that is, controlling the transfer of data between the computer and the peripheral device.

Serial–port commands

There are two ways of controlling a serial port. One way, commonly used by Applesoft programs or from the Monitor, is to activate a port or slot by means of the Input and Printer commands, as shown in Table 3–2, and then send command characters in the output stream, as shown in Table 3–3.

The second method of controlling a serial port is by the standardized firmware protocol. Your program makes calls to command routines whose addresses your program has found in standardized locations derived from the slot number. The firmware actually contains two separate interfaces, one for Applesoft BASIC and one, called the Pascal 1.1 interface, for other languages. Table 3–4 and 3–5 summarize the two interfaces to the firmware. For complete descriptions, refer to the *Apple IIGS Firmware Reference*.

Important: The manuals for the Super Serial Card and for the Apple IIc also list hardware registers and screen–hole locations for controlling the ports. If you want your programs to run properly on the Apple IIGS and on future models of the Apple II, do not control the ports by means of the hardware; use calls to the firmware or use the toolbox. See the section "Serial-port Compatibility."

Table 3–2. Input and Printer commands. The letter *s* stands for the port number, either 1 or 2.

Function	Applesoft command	Monitor command
Start input on port *s*	IN#*s*	*s* Control–K
Start output on port *s*	PR#*s*	*s* Control–P

Table 3–3. Summary of I/O commands

Command	Description
*nnn*N	Set line width to *nnn*
*nn*B	Set baud rate to one of fifteen standard values selected by *nn*. Lowest rate is 50, highest is 19200.
C	Send automatic carriage return whenever line width exceeded
*n*D	Set data format—number of data bits and stop bits—to setting specified by *n*. Data bits can be 5, 6, 7, or 8; stops bits, 1 or 2.
F	Disable keyboard to prevent disturbing input data stream
I	Echo output to display screen
K	Disable automatic line feed after carriage return
L	Generate automatic line feed after carriage return
M	Mask out (delete) incoming line–feed characters
*n*P	Set parity as selected by *n*. Parity can be even, odd, mark, space, or none.
Q	Quit (turn off) terminal mode
R	Reset port
S	Send a break character
T	Enter terminal mode
X	Turn on XON/XOFF I/O protocol
Z	Zap (ignore) further commands until Control–Reset

Table 3–4. Address locations for BASIC protocol. The letter *s* stands for the port number, either 1 or 2.

Address	Description
$C*s*00	Initialization routine (also outputs a character)
$C*s*05	Read a character
$C*s*07	Write a character

Table 3–5. Address locations for Pascal 1.1 protocol. The letter *s* stands for the port number, either 1 or 2.

Address	Description
$C*s*0D	Offset to initialization routine (PInit)
$C*s*0E	Offset to read routine (PRead)
$C*s*0F	Offset to write routine (PWrite)
$C*s*10	Offset to status routine (PStatus)
$C*s*12	Offset to control routine for extended interface

Note: To obtain the address of the desired routine, read the offset byte from the address given in the table and add it to the slot address, $C*s*00. To use the extended interface, set up a command list and then JSR to the address of the control routine, as described in the *Apple IIGS Firmware Reference.*

For complete descripitons of the interfaces to the serial I/O firmware, refer to the *Apple IIGS Firmware Reference.*

Terminal emulation

The Apple IIGS firmware supports a terminal emulation mode that works like the one in the Apple IIc. The terminal emulation has a minimum of features, and is intended for use only when a full–featured communications package is not available. The terminal emulation passes characters typed on the keyboard (except command strings) to the serial output, and passes serial input to the display.

The user puts the Apple IIGS into terminal mode through the BASIC interface by typing

IN#*sc*T

where *s* is the port number and *c* is the command character (usually Control–I for the printer port or Control–A for the communications port). The letter *T* is the terminal command, as shown in Table 3–3. To quit terminal mode, the user types the command character followed by the letter *Q*, the Quit command.

When running terminal emulation at high baud rate, you can use the firmware's buffering features (described below) to keep from losing characters during display scrolling.

New serial–port features

The serial ports on the Apple IIGS have several new features in addition to the ones found on the Super Serial Card and the Apple IIc. The new features include:

- I/O buffering
- background printing
- built–in AppleTalk interface

This section describes the new features briefly; for more information, refer to the manual *Apple IIGS Firmware Reference.*

I/O buffering

The serial–port firmware supports input and output buffering. Each port has an input buffer and an output buffer. The default buffer size is 2K, which the firmware requests from the Memory Manager, but an application can request larger buffers (up to 64K) and pass the location and size to the firmware.

There are four ways to turn on buffering:

- from the control panel
- from the keyboard after the Applesoft PR# command
- from an application by a command in the output stream
- from an application by a command to the serial firmware

Output buffering puts characters in a FIFO (first–in, first–out) queue in the output buffer space, then sending them on to the output device whenever it is ready. Input buffering puts characters into a queue in the input buffer and responds to calls to the firmware's Read routine with characters from the queue.

Although the application is not involved in the interrupt process that the firmware uses to support buffering, the application can keep track of buffer activity by making extended–interface calls that return the number of characters in the input queue or the amount of space left in the output queue. (Those calls are InQStatus and OutQStatus; refer to the Apple IIGS Firmware Reference for descriptions.)

Background printing

The firmware can send a block of characters out a serial port while an application is running. This background printing is similar to output buffering except that the firmware accepts a large number of characters all at once instead of getting them one at a time. When the firmware transmits the last character in the output buffer, it calls a recharge routine, supplied by the application, that refills the buffer. As with normal buffering, the application can either use the default 2K buffer or request its own buffer of up to 64K from the Memory Manager.

AppleTalk interface

The user can connect AppleTalk to either one of the serial port connectors and activate it by means of the Control Panel desk accessory. At any given time, only two of three I/O functions—AppleTalk, serial port 1, serial port 2—can be active. (The Control Panel ensures that one serial port is made inactive when AppleTalk is selected.)

AppleTalk is Apple's local–area network for Apple II and Macintosh, using the LaserWriter and ImageWriter II. Like the Macintosh, the Apple IIGS has the AppleTalk interface built in.

So that the Apple IIGS can support AppleTalk, the interrupt service routine is designed to respond to the serial–port hardware fast enough to preclude **data overruns.** In addition, a hardware timer generates a system interrupt four times a second to enable the AppleTalk firmware to carry out network operations.

A **data overrun** occurs when input data comes faster than the computer can accept it.

Serial–port compatibility

Even though the commands used to communicate with the serial–port firmware are the same as those in the firmware on the Super Serial Card (and similar to the ones in the Apple IIc), some existing programs using these ports will not be compatible with the serial ports on the Apple IIGS. The reason is that many programs, especially communications packages, bypass the firmware commands and go directly to the hardware. Programs that control the hardware directly won't be compatible with the Apple IIGS, because it uses the 8530 Serial Communications Chip (SCC), not the 6551 Asynchronous Communications Interface Adapter (ACIA) used in the Super Serial Card and the Apple IIc.

Programs that use the port to control a printer are more likely to use the firmware commands, making them compatible with the Apple IIGS. The same goes for most applications written in Applesoft or Pascal. AppleWorks and MousePaint are examples of programs that control the ports by calls to the firmware and so are compatible with the Apple IIGS.

Even programs that use the firmware can get into trouble if they communicate with the firmware by modifying the contents of the **screen holes**. The serial–port firmware takes the place of ROM in slots 1 and 2, so it uses the screen–hole locations for those slots. Rather than making proper calls to the firmware, some programs control the operation of the firmware by changing the values in those locations. While that may work on a particular model of Apple II, the firmware in another model may not react the same way. For complete information about the serial ports, refer to the manual *Apple IIGS Firmware Reference.*

(marginal gloss) The **screen holes** are locations in the text display page that are used by the ROM on cards in expansion slots, as described in the earlier section "Slot RAM space."

Built–in disk port

The Apple IIGS has a built–in disk port like the one on the Apple IIc. The disk port uses an IC called the IWM (Integrated Woz Machine) and can handle up to six drives, connected in a daisy chain. The drives can include one DuoDisk (which counts as two drives), up to two UniDisk drives, and four UniDisk 3.5 or Apple 3.5 (unified) drives.

Note: Disk II 5.25–inch drives won't work with the built–in port because their connectors won't fit. They work fine with a Disk II controller card installed in an expansion slot.

Apple II: The earliest form of disk storage available for the Apple II consisted of Disk II controller cards and Disk II drives using 5.25–inch floppy disks with 143K storage capacity. Each controller card could handle one or two drives; for more than two disk drives, you needed additional controller cards. The conventional location for the first controller card was slot 6; the second card went in slot 5. For initial loading (**booting**) from disk, the startup routine in the firmware started with slot 7 and tried successively lower–numbered slots until it found one with a disk controller card in it. Most software for the Apple II was designed to use slot 6, drive 1, as its startup drive. On more recent Apple II's that have a built–in disk interface, the slot and drive nomenclature is less meaningful, but it is still the convention because so many programs designed that way are still in use.

(marginal gloss) **Boot** is short for **bootstrap load**, a term suggestive of the difficulty of initial loading of loader programs into early computers that didn't have built–in firmware in ROM.

The disk–port firmware handles drives addressed as internal slots 5 and 6. You can also install a disk interface card in slot 6 and have two additional 5.25–inch drives (although you can't use all the drives at the same time). You can boot the Apple IIGS from drive 1 in either slot. Using the Control Panel desk accessory, you can determine whether the firmware will look for the boot device in slot 5, in slot 6, or scan downward from a specified slot.

The disk–port firmware also controls /RAM5, a block–storage device emulated in RAM and activated as slot 5, drive 2. When /RAM5 is active, the firmware accesses the second 3.5–inch disk drive as slot 2, drive 1.

SmartPort and Protocol Converter

SmartPort is a set of assembly–language routines used to support **block I/O devices.** The SmartPort firmware includes the Protocol Converter software used in the Apple IIc 3.5 ROM revision. SmartPort supports two 5.25–inch drives, two Apple 3.5 drives, up to 127 UniDisk 3.5 drives, and the RAM disk volume /RAM5. (The disk–port *hardware* can handle a maximum of six drives.)

(marginal gloss) A **block I/O device** reads or writes information in organized groups called blocks, typically 512 bytes. A disk drive is a block device.

Applications can make calls to the SmartPort to perform the following functions.

- obtaining status information about a device
- resetting a device
- formatting the medium in a device
- reading from a device
- writing to a device
- sending control information to a device

Calls to SmartPort use the same technique as the Pascal 1.1 protocol summarized in Table 3–5, except the address values are in the slot 6 locations. For complete information about SmartPort, refer to the manual *Apple IIGS Firmware Reference*.

Game I/O connectors

The game I/O connectors can be used for attaching one or two pairs of hand controllers or game paddles, one or two joysticks, a graphics tablet, or a similar I/O device designed for use with Apple II computers.

> **Note:** Similar I/O devices designed for the Apple IIGS can be connected to the DeskTop Bus, which is described in the next section.

Like the Apple IIe, the Apple IIGS has two game I/O connectors: a 9–pin miniature D–type connector on the back panel, and a 16–pin DIP socket on the main circuit board, inside the case. The 9–pin connector has four analog inputs (used for hand controllers or in pairs for joysticks), three button inputs, power, and ground. The 16–pin socket has the same signals as the 9–pin connector, plus a strobe and four single–bit outputs.

Apple DeskTop Bus

The Apple DeskTop Bus is a simple I/O interface with two different but related functions. Its primary function is to provide intelligent support for the keyboard and the DeskTop Bus mouse. It also provides a convenient way to connect additional input devices, such as hand controls, graphics tablets, numeric keypads, and other keyboards.

The DeskTop Bus is a serial interface (not a standard serial I/O port) that is controlled by its own built–in microprocessor, the Apple DeskTop Bus (ADB) microcontroller. DeskTop Bus devices use inexpensive four–conductor cables and four–pin miniature **DIN** connectors. Additional devices connect in parallel with devices already installed; some devices, such as the detached keyboard, include a jack for connecting other devices. The different types of devices have different identifiers; if there are two devices of the same type, the ADB microcontroller assigns them different identifiers.

A **DIN** connector is a type of connector with multiple pins inside a round outer shield. The initials DIN stand for Deutsche Industrie Normal, an European standards organization.

Detached Keyboard

The Apple IIGS keyboard is the new Apple standard detached keyboard. The new keyboard layout includes several enhancements, most notably a numeric keypad. It also conforms to European standards in the shape and position of the Return and Shift keys.

The Apple DeskTop Bus microcontroller (ADB microcontroller) supports the detached keyboard, providing basic scanning and encoding along with special features such as a type–ahead buffer. The ADB microcontroller supports eight different keyboard layouts, making it easier to localize the Apple IIGS for other countries. The ADB microcontroller also supports the **Dvorak** keyboard layout, which the user can select by means of the Control Panel desk accessory.

(marginal gloss) The **Dvorak** typewriter keyboard, also called the *New American Standard Keyboard,* has its keys arranged such that it is more efficient to use than the more common Qwerty (for the first line of keys) keyboard arrangement.

With the Apple IIGS Upgrade installed in an Apple IIe, the ADB microcontroller supports the internal keyboard, providing the same features that are available with the detached keyboard.

Mouse

The DeskTop Bus provides an improved interface for the Apple Mouse. Although the actual mouse hardware is unlike that on either the Apple IIe mouse card or the Apple IIc, the calling sequences are the same, as required for program compatibility.

The Apple Mouse contains a microcontroller that keeps track of the movement of the mouse up to plus–or–minus 63 increments (±$3F) and reports mouse information to the DeskTop Bus, which passes it on the the mouse routines in the firmware. Like the AppleMouse card for the Apple IIe (and unlike the mouse interface on the Apple IIc), the ADB controller reduces the burden that operation of the mouse places on the main processor, as described in the next section.

DeskTop Bus firmware

The DeskTop Bus firmware provides communications and control for the detached keyboard (along with the built–in keyboard when the Apple IIGS Upgrade is installed in an Apple IIe) and the DeskTop Bus mouse. It also acts as a simple communications interface for other input devices such as joysticks and graphics tablets.

For applications using the DeskTop Bus, there is a DeskTop Bus tool set: See Chapter 5 for more information.

The firmware supports mouse operations in somewhat the same way as the AppleMouse card for the Apple IIe. Like the AppleMouse card, the Apple DeskTop Bus supports interrupt–mode operation of the mouse, waiting until VBL occurs before interrupting the system. It also provides a true passive mode: that is, a mode in which the mouse interface doesn't interrupt the application, but waits for the application to poll it. Using passive mode, applications can operate the mouse while running software routines that mustn't be interrupted, such as critical timing loops.

VBL is short for *vertical blanking;* it is an interrupt signal generated by the video timing circuit each time it finishes a vertical scan. The vertical scan happens 60 times a second, so VBL is a convenient way to control the frequency of other events, such as mouse interrupts.

For complete information about the operation of Apple DeskTop Bus, refer to the manual *Apple IIGS Firmware Reference*. To find out how to connect a device to the bus, refer to the *Apple IIGS Hardware Reference*.

Chapter 4

Firmware Features

The Apple IIGS has a total of 128K bytes of ROM for firmware: permanently resident programs. The firmware includes the following features:

- driver programs for built–in I/O ports
- resident desk accessories
- Monitor
- Monitor I/O routines
- resident Toolbox
- Applesoft BASIC interpreter

This chapter describes only the resident desk accessories, the Monitor, and the Monitor I/O routines. The built–in I/O ports are described in the previous chapter. The Toolbox is described in Chapter 5 and in the *Apple IIGS Toolbox Reference,* Volume 1 and Volume 2. (See Chapter 5, "The Apple IIGS Toolbox.") Applesoft BASIC has its own manuals: *Applesoft Tutorial, Applesoft Reference Manual,* and *BASIC PRogramming with ProDOS.*

Resident desk accessories

Desk accessories are programs, usually small, that the user can invoke to perform some immediate task when some larger program is running. When the desk accessory is finished, the interrupted program can continue. Most desk accessories are loaded from disk and reside in RAM, but there are two that are permanently resident in ROM: the Control Panel and the Alternate Display Mode.

Control Panel

The Control Panel is a permanently resident desk accessory that the user can invoke while another program is running. The Control Panel enables the user to specify the operating parameters for the following functions:

- I/O ports: printer or modem, line length, baud rate, and so on
- display: 40 or 80 columns; colors for text, background, and border
- pitch and volume of sound to use for bell
- operating speed: 1 MHz or 2.5 MHz
- slot allocation: internal ports or peripheral cards

- startup slot
- language (character set) for keyboard and display
- built–in clock: time and date

The Desk Manager, which controls the desk accessories, is described in Chapter 5, "The Apple IIGS Toolbox." The *Apple IIGS Owner's Guide* describes the operation of the Control Panel by the user. The *Apple IIGS Firmware Reference* describes the operation of the Control Panel by an application.

Alternate Display Mode

The Alternate Display Mode is a small firmware routine that can be activated from the Control Panel. It makes the Apple IIGS compatible with standard Apple II programs that create animated displays by rapid alternation or flipping of the two Lo–Res graphics pages.

Standard Apple II programs running on the Apple IIGS normally can't display text Page 2 (also known as Lo–Res graphics Page 2) because the hardware does not shadow it. If such programs use page flipping for Lo–Res animation, the display won't look right unless they can display text Page 2. To run the programs on the Apple IIGS, the user must first turn on Alternate Display Mode, which periodically transfers data from text Page 2 in bank $00 to the same area of bank $E0, where it can be displayed.

The Monitor

The Monitor is a built–in program that provides machine–language access to the registers and memory. The Monitor includes firmware I/O routines to accept commands typed at the keyboard and to display text on the screen. These I/O routines provide low–level input and output functions that application programs can also use: see the next major section, "Monitor I/O firmware." Even though the Mini–assembler and Disassembler are considered parts of the Monitor, this section describes them separately, after the other features of the Monitor.

Using the Monitor

Apple II: This section describes features that are common to the Monitor programs on all models of the Apple II. If you are already familiar with the Monitor, you might as well skip ahead to the section "New Monitor Features."

The Monitor is a built-in utility that enables a programmer to examine code and data in memory and to execute portions of the code. The Monitor program occupies memory in ROM bank $FF, starting at location $FF69 (–151). That part of ROM is mapped into banks $00 and $01 by the language–card switches when Applesoft BASIC or other standard Apple II programs are running. One way to invoke the Monitor is to have Applesoft running and type

```
CALL -151
```

When the Monitor is running, the prompt character is an asterisk (*). When you are finished using the Monitor, you return to Applesoft by pressing Control–Reset or Control–C.

The Monitor does not support the desktop user interface. To give a command to the Monitor, you type a line at the keyboard and press Return. Commands contain three kinds of information: addresses, data values, and command characters. Addresses are in hexadecimal; data values can be in hexadecimal or in the form of ASCII characters.

Monitor commands

> **Apple II:** This section describes features that are common to the Monitor programs on all models of the Apple II. If you are already familiar with the Monitor, you might as well skip ahead to the section "New Monitor Features."

Like the Monitor programs in all other models of the Apple II, the Apple IIGS Monitor allows programmers to operate on programs in memory at the lowest level. The Monitor includes instructions to

- display the contents of a memory location
- display a range of memory locations
- store values starting at a location
- display the contents of the registers
- change the contents of the registers
- move a block of memory
- compare (verify) two blocks of memory
- direct output to port or slot *n*
- accept input on slot or port *n*
- execute program code starting at a location
- disassemble code starting at a location

New Monitor features

Among the new features of the Apple IIGS Monitor are

- new commands
- improved display
- extended memory addressing

The Apple IIGS Monitor also includes enhanced versions of the Apple II Mini–assembler and Disassembler, which are described later.

New Commands

The Apple IIGS Monitor has many new commands. Among them are commands to

- save and restore registers and mode settings
- search memory for a pattern up to 256 bytes long
- fill part of memory with a one-byte value
- make a call to the Tool Locator
- store a new value into a specific register
- enter ASCII characters from keyboard into memory
- change the setting of the real-time clock
- convert hexadecimal to decimal or vice-versa
- perform 32-bit addition, subtraction, multiplication, and division
- switch between native and emulation modes

For descriptions of the Monitor commands, refer to the chapter on the Monitor in the *Apple IIGS Firmware Reference.*

Improved Display

Many of the Monitor commands display the contents of part of memory on the screen. The format of those displays has been improved, so they now include both hexadecimal and ASCII values.

Extended Memory Addressing

The Apple IIGS Monitor supports all the features of the new 65C816 microprocessor, including 16-bit registers and 24-bit addresses. The command syntax now includes two hexadecimal digits of bank address (delimited by a slash) so the Monitor can address any bank.

Mini-Assembler and Disassembler

All models of the Apple II have some version of the Disassembler, and all but the early models of the Apple IIe have a Mini-assembler. The Apple IIGS has both. They are enhanced to support the 65C816 microprocessor's new instructions and long addresses, and they support both native and emulation mode.

The Mini-assembler and Disassembler are special features of the Monitor. The Mini-assembler provides a means of developing and debugging an program or routine in a very simple form of assembly language.

When you invoke the Mini-assembler, the prompt character changes to an exclamation point (!) and the Monitor accepts 65C816 instructions in the form *address : opcode operands*

The address field and the colon are optional; you omit them to enter consecutive instructions. The Mini–assembler does not maintain a symbol table, but it does recognize all the standard instruction mnemonics, and it calculates relative addresses. It recognizes a preceding number sign (#) as signifying an immediate operand. You use the letters X and Y, set off by commas, for indexing, and you type indirect addresses inside parentheses.

To stop the Mini–assembler, you type a null line by pressing the Return key.

Unlike the Mini–assembler, which takes over the user interface and accepts inputs from the user, the Disassembler is just the Monitor's List command. It lists the contents of memory, one screenful at a time, converting op codes into mnemonics and relative addresses into absolute addresses.

Both the Mini–assembler and the Disassembler can handle all 91 of the 65C816's instructions and all 24 addressing modes (a total of 256 op codes). In addition, the disassembler properly expands operating–system calls to **ProDOS 8** and **ProDOS 16,** showing command numbers and parameter–list pointers on separate lines.

ProDOS 8 and **ProDOS 16** are the disk operating systems that run on the Apple IIGS. See Chapter 8 for descriptions of ProDOS 8 and ProDOS 16.

Monitor I/O firmware

Apple II: This section describes the Monitor I/O routines, which are functionally the same on the Apple IIGS as on the Apple IIe and Apple IIc. If you are already familiar with the Monitor I/O routines, you might as well skip ahead to the section "Interrupt support."

The Monitor accepts inputs from the keyboard and displays information on the screen. To do these tasks, it has its own I/O routines. Every Apple II contains some version of the Monitor, so it also contains these built–in I/O routines. Because they are always available, many application programs use them for keyboard input and text display output.

Standard I/O links

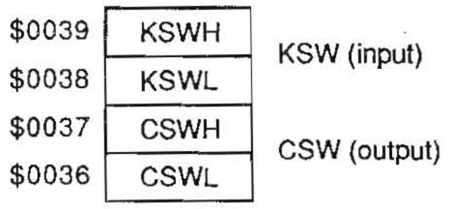
The Monitor I/O routines include standard input and output routines that are used by the operating system, by device drivers, and by applications. The standard I/O routines pass control on to internal I/O routines by way of two locations in RAM called the I/O links. The I/O links contain the addresses of whatever I/O routines are in control at the time.

In an Apple II running without an operating system, the I/O links normally contain the addresses of the standard internal I/O routines. An operating system typically replaces the link addresses with the addresses of its own I/O routines, and in turn calls the internal I/O routines.

There are two sets of internal I/O routines: one set that exists in all Apple II's, even the earliest, and another set that exists only on Apple II's that support 80–column displays. The routines in the earlier set are KeyIn and COut1; the 80–column routines are C3KeyIn and C3COut1. (KeyIn is pronounced *key in* and COut1 is pronounced *C out one.* C3KeyIn and C3COut1 are pronounced *C three key in* and *C three C out 1.)*

The I/O links are two–byte addresses at locations $0036 and $0038 in bank $00: see Figure 4-1. The link at location $0036 is the output link; it is named CSW, for *character (output) switch.* It holds the address of the subroutine that handles single–character output. When you issue a PR#n command from Applesoft or an n Control–P from the Monitor, the firmware changes the address in this link to the first address in the ROM space allocated to slot or port number n. Subsequent calls to the output link are thus transferred to the firmware associated with that slot or port. When you issue a PR#0 or a 0 Control–P, the firmware replaces the slot ROM address at CSW with the address of the internal output routine.

Figure 4-1. Standard I/O links



The link at location $0038 is the input link; it is named KSW, for *keyboard (input) switch.* Like the output link, it normally holds the starting address of a standard routine—in this case, the routine for single–character input. When you issue an IN#n command from Applesoft or an n Control–K from the Monitor, the firmware changes the address in this link to the first address in the ROM space allocated to slot or port number n. Subsequent calls to the input link are thus transferred to the firmware associated with that slot or port. When you issue an IN#0 or a 0 Control–K, the firmware replaces the slot ROM address at KSW with the address of the internal input routine.

Input routines

The Monitor firmware includes two different subroutines for reading from the keyboard: RdKey (pronounced *read key)* and GetLn (pronounced *get line).* The RdKey routine provides input of a single character by calling the current character input routine, that is, the routine whose address is stored in the input link at KSW. That routine is normally either KeyIn or C3KeyIn, which accepts one character from the keyboard. The KeyIn routine displays a cursor at the current cursor position, waits until someone presses a key, then puts the ASCII value of that key into the accumulator and passes control back to the calling program.

The GetLn routine provides input for entire lines by making repeated calls to the input routine until it gets a carriage return. GetLn starts by displaying a prompt: a character that indicates that the program is waiting for input. Different programs can have different prompt characters simply by storing the desired character at a specified location in RAM. As the user types keys, the GetLn routine stores the ASCII values into successive locations in the input buffer in memory locations $0200–$02FF. The GetLn routine also supports some simple screen editing and control features.

Output routine

The standard output routine is named COut (pronounced*C out*, for *character output)*. It calls the current character output routine, that is, the routine whose address is stored in the output link CSW. The character output routine is normally either COut1 or C3COut1, which sends one character to the display, advances the cursor position, and scrolls the display if necessary. Both character output routines restrict their use of the display to an active area called the text window, which is determined by four values stored in RAM: left margin, width, top line, and bottom line.

For more information about the standard input and output routines, refer to the manual *Apple IIGS Firmware Reference.*

Other routines

The Monitor firmware also contains other useful routines for dealing with the keyboard and display. Like the standard I/O routines described above, they carry out low–level functions appropriate for the operation of the Monitor. The firmware routines include functions such as

- clearing all or specific parts of the screen
- clearing the screen and putting the cursor in the upper–left corner
- drawing colored points and lines in Lo–Res graphics
- getting the color of a specified location on the Lo–Res screen
- printing out the value in the accumulator, in hexadecimal

For more information about the firmware I/O routines, refer to the manual *Apple IIGS Firmware Reference*.

Interrupt support

The firmware includes interrupt support for the full range of interrupts possible on the Apple IIGS. As in the Apple IIc and the enhanced Apple IIe, the firmware on the Apple IIGS makes interrupt–driven programs possible. Interrupts work well with ProDOS (any version) and Pascal (revision 1.2 or higher); DOS 3.3 doesn't support interrupts.

The goal of the interrupt handler is to support interrupts in any memory configuration. It saves the machine's state at the time of the interrupt, and puts the machine into a standard memory configuration before passing control to your program's interrupt handler.

An **interrupt vector** is the address of an interrupt service routine, stored in a fixed location. When an interrupt occurs, the system uses the interrupt vector to transfer control to the service routine.

Important: The **interrupt vectors** are stored in system ROM (bank $FF, locations $FFEE–$FFFF), and so is a short interrupt service routine (bank $FF, locations $C071–$C07F). For interrupts to work with programs running in banks $00 and $01, I/O shadowing and language–card mapping must be on. Table 4–1 is a summary of the types of interrupts the firmware recognizes. For more information about interrupts, please see the manual *Apple IIGS Firmware Reference*.

IRQ is short for *interrupt request*, which is a signal input to the microprocessor requesting an interrupt. Depending on the state of a flag in the processor's status register, it can either react to an IRQ or ignore it.

Table 4–1. List of Apple IIGS Interrupts

Type of interrupt	**Cause of interrupt**
Program BRK instruction	A break instruction in a program
Peripheral card **IRQ**	Request from a peripheral card
VBL	Vertical–blanking time occurred
Video scan line	Scan–line time occurred
Mouse	Button, movement, or VBL
AppleTalk Network	Address recognition or error
Timer for AppleTalk	Occurs every 0.26667 seconds, to trigger event processing by AppleTalk
Keyboard	Key was pressed
Serial input on port 1	Transmitter empty, data received, or error
Serial input on port 2	Transmitter empty, data received, or error
Ensoniq DOC	An oscillator completed a waveform table
Clock chip	Occurs every second
Apple DeskTop Bus	A Desktop–Bus device requires service
Cold–start reset	Power up, or Control–Apple–Reset keys pressed
Warm–start reset	Peripheral–card reset, or Control–Reset keys pressed

Chapter 5

The Apple IIGS Toolbox

One of the important differences between the Apple IIGS and earlier models of the Apple II is that, like the Macintosh, the Apple IIGS has a built-in Toolbox with routines that can be called by applications. The Toolbox serves two purposes: It makes developing new applications easier, and it supports the desktop user interface.

What's the Toolbox?

The Apple IIGS Toolbox is a collection of useful routines that can be called by application programs. The Toolbox routines are a permanent part of the system; they are available to application programs without the need to link libraries to applications.

The Toolbox routines have many uses. There are routines that support the new hardware features of the Apple IIGS, such as Super Hi-Res graphics and the digital oscillator chip (DOC). Other routines support the desktop user interface, which uses mouse operations in menus and windows.

The Toolbox routines are arranged in logical groups called *tool sets, managers,* or simply *tools.* Each individual routine that can be called by an application is a *tool call.* For example, the routines that support the Super Hi-Res graphics display are in a tool set named *QuickDraw II*, and *PaintPoly* is a typical call in that tool set.

Not all of the tools are resident in ROM; some of them are loaded from disk and reside in RAM. The calling mechanism is the same regardless of where in memory a tool resides. A tool can even be in RAM in one version of the Toolbox and in ROM in another version; the application will run the same in either case.

Developers are not restricted to the tool sets provided by Apple; they can create tool sets of their own. The Tool Locator provides a way to switch back and forth between the Apple IIGS tool sets and the application's own tools. For information about creating a tool set, please read the manual *Apple IIGS Toolbox Reference, Volume 1.*

Apple IIGS Toolbox compared with Macintosh

Most of the routines in the Apple IIGS Toolbox are similar to routines in the Macintosh Toolbox. In fact, the Apple IIGS designers started with the most important Macintosh routines and tried to copy them as closely as possible, considering the differences between the machines. Much of the work a typical event-driven application does to support the user interface can be accomplished using the Apple IIGS Toolbox.

Similarities

People familiar with the Macintosh Toolbox will find that many of the routines in the Apple IIGS Toolbox are similar to their Macintosh counterparts. Table 5-1 is a list of those Apple IIGS tool sets and the similar tool sets in the Macintosh.

Table 5-1. Macintosh counterparts for Apple IIGS tool sets

Apple IIGS tool set	**Macintosh tool set**
QuickDraw II	QuickDraw
SANE	Floating-Point Package
Desk Manager	Desk Manager
Event Manager (high–level calls)	Toolbox Event Manager
Event Manager (low–level calls)	Operating System Event Manager
Menu Manager	Menu Manager
Window Manager	Window Manager
Control Manager	Control Manager
LineEdit	TextEdit
Dialog Manager	Dialog Manager
Scrap Manager	Scrap Manager
Print Manager	Printing Manager

Several other tool sets in the Apple IIGS Toolbox have functions similar to tool sets in the Macintosh, but actually work quite differently. For example, the Tool Locator in Apple IIGS has the same function as the Trap Dispatcher in the Macintosh, but it's actually quite different. Similarly, the Memory Manager in Apple IIGS has the same job as the one in the Macintosh, but deals with a memory space quite unlike that of the Macintosh. Other examples include the Apple IIGS System Loader, which is associated with **ProDOS 16**, and the Text Tools, used with the text display, a Apple IIGS feature that has no equivalent on the Macintosh.

ProDOS 16 is the disk operating system for the Apple IIGS. See Chapter 8 for a description.

Differences

While many of the routines in the Apple IIGS Toolbox are similar to their counterparts in the Macintosh Toolbox, they are certainly not identical. Table 5-2 lists the main reasons for the differences.

Table 5-2. Differences between the Apple IIGS and the Macintosh

Feature	**Apple IIGS**	**Macintosh**
Display	Color graphics and text	Black–and–white graphics and text
Microprocessor	65C816, a descendant of the 6502	68000
Memory organization	64K memory banks	Continuous memory
Resource manager	Not present	Part of Toolbox
TaskMaster	Part of Window Manager	Not present
Sound tools	Sound Manager	Free-Form Sound Player

The **Resource Manager** is a Macintosh tool for editing data in programs without recompiling them.

The Super Hi-Res graphics display on the Apple IIGS is supported by the QuickDraw II tool set, which provides many functions similar to those in the QuickDraw tool set on the Macintosh—similar, but not identical.

One major difference is that the Super Hi-Res display has color, which the Macintosh display doesn't have. Another difference is that the Super Hi-Res display is coarser: its highest resolution is 640 x 200, compared with 512 x 342 for the display on the Macintosh. The **aspect ratios** of the pixels are also different: pixels in the Macintosh display are square, but pixels in the Super Hi-Res display are tall (aspect ratio 5:6 in 320 mode, 5:12 in 640 mode). If your Macintosh application includes the display dimensions as constants, you'll have to make appropriate changes to use the application on the Apple IIGS.

The **aspect ratio** of an image is the ratio of its width to its height. The standard video display has a 4:3 aspect ratio.

The microprocessors used in the two machines are entirely different. The 65C816 used in the Apple IIGS has different instructions and addressing modes from those of the 68000 used in the Macintosh. The 65C816 has 16–bit data registers, while the 68000 has 32–bit registers.

Memory is organized differently on the two machines. Memory in the Macintosh is continuous, but memory in the Apple IIGS, though contiguous, is not continuous: it is divided into 64K banks, with parts of some banks dedicated to special tasks (such as display buffers and I/O devices).

On the Apple IIGS, memory banks $00 and $01 are broken up by several features needed for running programs written for earlier versions of the Apple II: the display pages, the I/O space, and the language-card space. Also, there are differences in the way the 65C816 microprocessor handles different banks. The Memory Manager on the Apple IIGS has to accomodate these restrictions.

The Apple IIGS Toolbox doesn't have everything in it that the Macintosh Toolbox has. One tool set not found on the Apple IIGS is the Resource Manager. You can still put your program's constants and data structures in a separate segment, but they won't be quite as easy for you to change. You'll need to be aware of this difference when setting up the segment with the items you might want to change, such as icons and menu titles.

On the other hand, the Apple IIGS has some tools that the Macintosh doesn't. For example, the Window Manager on the Apple IIGS has a special call, TaskMaster, that makes it easier to use the window environment (see the section "The Window Manager," below).

Another area where the Apple IIGS differs from the Macintosh is that of sound tools. While the Macintosh has the Free-Form Sound Player, the Apple IIGS has the Sound Manager, a low-level tool for controlling the digital sound chip (the Ensoniq DOC).

Suggestions for Programmers

Applications on the Apple IIGS have strong resemblances to applications on the Macintosh. They can have a similar desktop user interface, with menus and windows which the user manipulates by using a mouse. Programs on both machines can be event-driven, so their structures can be similar, with a main event loop and conditional branches to the parts of the program that deal with each kind of event.

To keep from being dependent on memory configuration, applications on the Apple IIGS use program segmentation and relocatable code. Unlike programs on the Macintosh, Apple IIGS programs are not normally position independent, but they can be relocated by the System Loader.

Programs written in high level languages like C or Pascal on the two machines can be very similar. Programs or segments written in assembly language can also take advantage of the similarities between the Apple IIGS and the Macintosh, but must be rewritten because the machines use different microprocessors. The microprocessors have different architectures and addressing modes, so they require different assemblers.

> **Note:** Apple's assembler for the Apple IIGS is not like the EdAsm assembler for the Apple II. The Apple IIGS assembler not only has the 65C816's additional instructions, it also uses different macros. For more information about the assembler, see Chapter 9.

For more information about programming on the Apple IIGS compared with the Macintosh, refer to the *Programmer's Introduction to the Apple IIGS*.

Making Calls to the Toolbox

Programs make calls to individual routines in the Apple IIGS Toolbox by means of call names. The calling mechanism depends on the language of the program that is making the call. For programs in assembly language, a macro library defines the names, and programs make calls in the following fashion:

1. Push space for the result (if any) onto the stack.
2. Push the input parameters onto the stack.
3. Invoke the call macro.
4. Pull the result (if any) from the stack.

For calls in high-level languages such as C or Pascal, there are libraries that define the names of the tool calls, along with appropriate coding conventions for passing parameters on the stack, similar to the one defined above for assembly-language calls.

The tool sets

Here are brief descriptions of the tool sets on the Apple IIGS. For complete descriptions, please refer to the manuals *Apple IIGS Toolbox Reference,* Volume 1 and Volume 2. Tool sets with related functions are grouped together: for example, all the tools that support the desktop interface are together in the section "The desktop tools."

The big five

These five tools—Tool Locator, Memory Manager, QuickDraw II, Event Manager, and Miscellaneous tools—make up the foundation of the Toolbox. Your program may not call on them directly, but other parts of the Toolbox and of the operating system are heavily dependent on them.

The Tool Locator

The Tool Locator provides the mechanism for dispatching tool calls. Thanks to the Tool Locator, tool sets can reside either in ROM or in RAM. That makes it possible for future versions of the Toolbox to substitute enhanced tools in RAM for tools presently in ROM with no changes to application programs.

Developers need to use the Tool Locator only if they are adding their own tool sets to the Toolbox; the *Apple IIGS Toolbox Reference,* Volume 1 tells how to do that.

The Memory Manager

The Memory Manager controls use of memory by application programs. Keeping memory use under control of the Memory Manager makes it possible to have co-resident applications such as desk accessories. The System Loader calls the Memory Manager to request memory space for loading a program. The program has the option of making its own calls to the Memory Manager to request (allocate) additional memory, release (deallocate) memory, or find out how much memory is currently available.

QuickDraw II

The standard display for the desktop environment on the Apple IIGS is the new color Super Hi-Res graphics. To support the graphics display, Apple IIGS firmware includes a set of graphics routines named QuickDraw II.

The graphics routines in QuickDraw II are based on a subset of the Macintosh QuickDraw routines. They include calls for changing the graphics environment and for drawing simple objects called primitive objects. The primitive objects QuickDraw II handles are:

- lines
- rectangles
- regions
- polygons
- ovals
- rounded rectangles
- arcs of circles
- pixel images
- text characters and strings

QuickDraw II is important not only to graphics applications, but to all applications that use the desktop interface, because it includes the text–drawing calls applications use for putting text into display windows on the desktop.

In addition to the drawing routines, QuickDraw II also has routines for performing calculations on different graphics objects; for example, to determine whether a specified point is inside a particular rectangle.

Besides all that, QuickDraw II also includes calls for defining the global graphics environment (for example, setting color tables) and for defining portable graphics environments, called GrafPorts, so that an application can keep track of several different graphics activities on different parts of the screen (or even in memory that isn't being displayed).

The Event Manager

An event-driven application carries out its operations in response to mouse and keyboard actions by the user. The application program is organized around a main loop that contains a call to the Event Manager followed by a series of conditional statements. These conditional statements determine the program's operations on the basis of the information returned by the Event Manager. For example, pressing the mouse button generates an event, which the Event Manager reports the next time around the loop. The Event Manager also reports events within the application that may require a response. For example, changing one window may cause another window to become visible and need to be redrawn.

The Event Manager on the Apple IIGS was designed to be as much like the event manager on the Macintosh as possible. Although it is a single tool set, it has two kinds of calls, high–level and the low–level, that resemble calls to the Macintosh Toolbox Event Manager and Operating System Event Manager. The Apple IIGS Event Manager detects low–level events, such as presses of the mouse button, and stores them in an event queue. High–level calls retrieve events from the event queue and report events that aren't kept in the queue, such as window events.

Miscellaneous tools

Tool calls in the miscellaneous tool set include routines to perform such tasks as

- accessing battery backed-up RAM
- reading and setting the built-in clock
- accessing peripheral cards
- changing the firmware interrupt vectors
- installing and deleting tasks in the heartbeat interrupt queue
- enabling or disabling some interrupt sources
- accessing the mouse directly

The desktop tools

These tools—Menu Manager, Window Manager, Control Manager, LineEdit, Dialog Manager, and Desk Manager—support the standard desktop interface.

The Menu Manager

An application program sets up menus and defines the menu bar by calling the Menu Manager. When the user gives a command, either from the menu using the mouse or by typing a command key, the application calls the Menu Manager to find out which command it is.

The Window Manager

Information displayed by an application program appears in windows. The application makes calls to the Window Manager to create windows, activate them, move them, change their sizes, and close them. The Window Manager keeps track of overlapping windows and posts events so the application can redraw windows that are newly uncovered. Also, when the application detects the event that happened when the user pressed the mouse button, the application calls the Window Manager to find out whether the cursor was in the menu bar or a desk accessory or, if it was in the window, which part of the window it was in.

One of the calls in the Window Manager is `TaskMaster`, which is a kind of extended get-event call. A `TaskMaster` call can handle many of the events that are likely to happen in a window environment, such mouse clicks in the control regions, without passing control back to the application. By using `TaskMaster` calls, a programmer can get an application up and running quickly and still take advantage of the features of the desktop user interface.

The Control Manager

A control is an object on the screen that the user clicks with the mouse to cause an action or change a setting. Controls include objects such as buttons, check boxes, and scroll bars. The application creates and responds to controls by means of calls to the Control Manager. When the application has found out from the Window Manager that the user pressed the mouse button in a window that contains controls, it then calls the Control Manager to find carry out appropriate actions, such as

- displaying or hiding a control
- monitoring the user's operation of a control
- reading or changing the setting of a control
- changing the size, location, or appearance of a control

LineEdit

Application programs accept text typed by the user and perform standard editing functions on the text by means of calls to LineEdit. Its functions are

- inserting and deleting text
- using the mouse to select text
- cutting and pasting text

LineEdit provides basic text-display formatting such as word wraparound. It handles only a line at a time, unlike the text editor in the Macintosh Toolbox, which is a multi-line editor.

The Dialog Manager

The Dialog Manager is a tool for handling dialog boxes and alerts in a way that is consistent with the Apple User Interface Guidelines.

When an application needs more information from the user about a command, it displays a dialog box. To alert the user in case of an error or a potentially dangerous situation, the application can display a box with a message, cause a sound from the speaker, or both. To create and display dialog boxes, to alert the user by a sound, and to find out the user's responses to the boxes and the sounds, the application calls the Dialog Manager.

The Desk Manager

The Desk Manager handles desk accessories, which are small co-resident application programs such as calculators, calendars, and the like. The user can invoke a desk accessory while an application is running, use the desk accessory for some task, then continue the application as if nothing had happened.

There are two kinds of desk accessories on the Apple IIGS: *classic* desk accessories that can run either in the Apple IIGS desktop environment or with non–Apple IIGS applications (like AppleWorks), and *new* desk accessories that run only in the Apple IIGS desktop environment. The Desk Manager checks to see which environment it is in and makes sure that a desk accessory can run in that environment before calling it.

Two classic desk accessories are built in: the Control Panel that is used to change the machine configuration and set the time and date, and the Alternate Display Mode that is needed for applications that use both Lo-Res graphics pages.

Mathematical tools

The Toolbox has two different ways of handling numeric operations: the SANE numerics, which provide comprehensive floating–point arithmetic, and the integer math tools, which are used by the other tool sets to perform integer arithmetic.

Floating-point numerics (SANE)

The Standard Apple Numerics Environment (SANE) is a scrupulously–conforming, extended–precision implementation of IEEE standard floating–point arithmetic. The Apple IIGS SANE tool set was derived from the 6502 Assembly Language SANE software, and has the same functions as the Macintosh SANE packages. Features of the numeric tool set include

- IEEE types single (32-bit), double (64-bit), and extended (80-bit)
- 64-bit type for exact fixed-point computations, such as in accounting
- basic floating-point operations (+ – * ÷ √ rem)
- comparisons
- conversions between binary and decimal or floating-point and integer
- scanning and formatting for ASCII numeric strings
- logs, exponentials, and trigonometric functions
- compound interest and annuity functions for financial computations
- random number generator
- functions for managing the floating-point environment
- other functions required or recommended by the IEEE standard

Integer math tools

The integer math tool set includes several routines for working on data of types *integer, long integer, fixed,* and *frac* (that is, fractional part). The functions of this tool set include multiplication, division, square root, some trigonometric functions, rounding, and conversions between data types.

The Print Manager

There is one tool set for dealing with printing: the Print Manager. Refer to the *Apple IIGS Tools Reference,* Volume 2, for information.

Specialized tools

The tool sets described in this section take care of specialized tasks.

The Sound Manager

The Sound Manager controls both the single–bit sound hardware and the digital oscillator chip (DOC). It includes two sets of routines: standard tool calls (called by way of the Tool Locator) and low-level calls (called by way of a jump table) designed for faster access.

By making tool calls to the Sound Manager, an application can

- send sound data to and from the sound RAM
- control the volume of the sound
- start and stop the sound from a particular sound generator in the DOC
- get the status of any or all generators in the DOC
- set up the sound interrupt handler
- get the address of the jump table for accessing the low-level routines

Using the low-level sound routines, an application can

- read or write any register in the DOC
- read or write any location in the sound RAM

The DeskTop Bus tools

The Apple DeskTop Bus (ADB) Tool Set provides a communications and control interface between your application and the ADB microcontroller that operates the DeskTop Bus. Besides the bus commands, the ADB Tool Set includes calls used by diagnostic routines and the Control Panel.

The ADB Tool Set includes specific commands for the keyboard and the mouse. For other devices, applications need driver routines that set up the devices and handle their operation. The setup routines identify the different devices on the bus and may even change bus addresses and data handlers for them.

The ADB Tool Set includes calls for polling all the devices on the bus. For repeated use of a single device, there is a polling call that always starts with the last device that was active. The application can use whichever polling method is appropriate to control the priority of devices on the bus.

The Scheduler

Much of the system code in the Apple IIGS is not **reentrant.** The Scheduler makes it possible to delay the execution of tasks that require non-reentrant system code whenever that code is already in use. Non-reentrant resources indicate that they are in use by modifying a flag called the Busy word. The Scheduler maintains a queue of processes waiting to use non-reentrant resources. By keeping track of the Busy word, the Scheduler determines when to activate the next process in the queue.

To be **reentrant,** a routine must be able to accept a call while one or more previous calls to it are pending without invalidating any previous calls.

The Text Tool Set

Like the other computers in the Apple II familty, the Apple IIGS has a video display mode for text only. To use the text–display firmware as earlier Apple II programs do, programs have to be running in emulation mode in bank $00. The Text Tool Set, along with the enhanced video output routines in the firmware, makes it possible for applications on the Apple IIGS to use the text display without switching environments and moving to bank $00.

Standard File Operations Tool Set

The Standard File Operations Tool Set provides the standard user interface for specifying a file to be opened or saved by an application. When the user selects Open or Save in the File menu, the application calls the appropriate standard file operation, which opens a dialog box, displays the files in the current volume, and handles user selection of files or options, such as selecting a different drive or ejecting a disk.

The Scrap Manager

The Scrap Manager includes routines and data types that make it possible to cut and paste text or graphics between two applications, between an application and a desk accessory, or between two desk accessories. From the user's point of view, the data being cut or pasted resides in the Clipboard.

The Scrap Manager keeps the data being cut and pasted resides in a block of memory called the *desk scrap*. The Scrap Manager can store it on disk if there isn't enough room for it in memory. The type of data being transferred is different for different applications. The Scrap Manager provides for different data types and provides some control over the amount of information that is retained when the scrap is transferred.

Chapter 6

Architecture of the Apple IIGS

The basic idea behind the Apple IIGS architecture is to make a more powerful Apple II—one that can run programs designed for earlier models of the Apple II and also support more sophisticated programs. The Apple IIGS achieves this contradictory–sounding goal by a combination of hardware and firmware—including a new microprocessor, expanded memory, improved video displays, and a new sound generator—that still has the ability to operate as an Apple II.

The microprocessor used in the Apple IIGS is the 65C816, a new 16–bit design based on the 6502 microprocessor used in other Apple II's. The 65C816 has two major features:

- It can operate either as a 16–bit processor or as an 8–bit 6502.
- It can address to up to 16 megabytes of memory.

The ability of the 65C816 to execute 6502 instructions makes it possible for the Apple IIGS to run programs designed to run on 6502-based models of the Apple II. The 65C816's large address space makes it possible for the Apple IIGS to have more memory than 6502–based Apple II's.

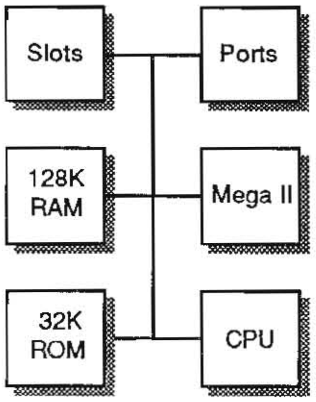
The design process

This section describes the design of the Apple IIGS as a process of expansion, starting with the Apple II. Understanding a little about the way the Apple IIGS evolved will help you understand the relationships between its new features and its old features.

Starting point: the Apple II

To understand how the Apple IIGS incorporates the features of the Apple II, first consider the standard Apple II. Figure 6–1 is a simplified block diagram showing how an Apple II might be designed around Apple's Mega II integrated circuit. The Mega II is a custom large–scale integrated circuit that incorporates most of the timing and control circuits of the standard Apple II. It addresses 128K of RAM organized as 64K main and auxiliary banks. The Mega II also provides the standard Apple II video display modes, both text (40–column and 80–column) and graphics (Lo–Res, Hi–Res, and Double Hi–Res). The slots indicated in Figure 6–1 are like the ones on the Apple IIe; the ports are like the ones on the Apple IIc.

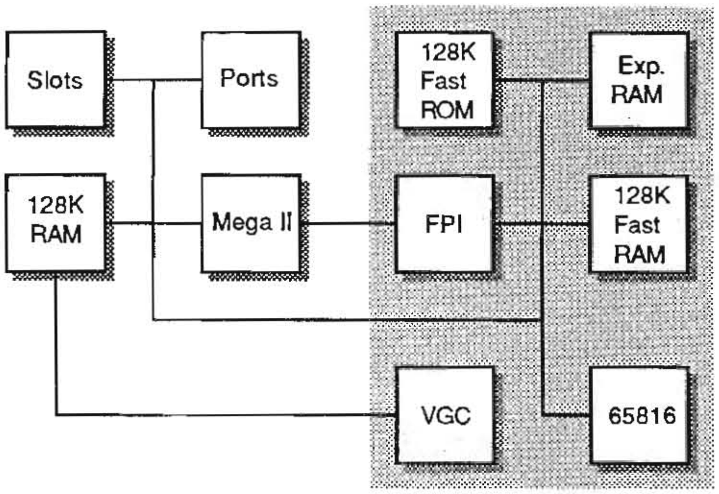
Figure 6–1. Hypothetical Apple II using the Mega II



Adding a faster processor

Now suppose that we replace the CPU with a new, faster microprocessor and add faster RAM and ROM and a new video display generator. Figure 6–2 is a simplified diagram of the result. Shading identifies the parts that provide the new features; generally speaking, the parts on the unshaded side provide the standard Apple II features. The CPU is now the 65C816 on the shaded side; it operates in 6502 emulation mode when executing standard Apple II programs.

Figure 6–2. New hardware added to the Apple II



The new CPU runs faster than the normal Apple II processor—2.8 MHz, compared with the normal 1 MHz. To manage the disparate speeds, the new system has a custom integrated circuit, the Fast Processor Interface (FPI), that supports the faster memory for the new CPU and controls CPU access to the slower Mega II side. Besides controlling the fast RAM and ROM, the FPI also controls expansion RAM, up to eight megabytes of additional fast RAM.

The shaded side of Figure 6–2 also includes the Video Graphics Controller (VGC). This integrated circuit provides a new video display, the Super Hi–Res graphics display. The new graphics display produces clear high–resolution color graphics on an RGB color monitor.

Figure 6–2 is misleading in one important respect: it implies that programs designed for the standard Apple II run in the part of RAM controlled by the Mega II, which is not the case; such programs actually run in the 128K of fast RAM on the shaded side of the diagram. The next section explains that aspect of memory on the Apple IIGS.

Memory on the Apple IIGS

The description of the Apple IIGS as merely an Apple II with a faster processor falls far short of the whole story. As Figure 6–2 shows, adding the faster processor requires adding faster memory. Besides that, one of the reasons for the new processor is not just that it runs faster, but that it can address more memory, making possible a significant increase in the amount of memory on the Apple IIGS. The following sections tell how the larger, faster memory is implemented.

Faster memory

The Apple IIGS is capable of executing instructions almost three times as fast as a standard Apple II. That speed can be used in two different ways: to obtain faster execution of standard Apple II programs, and to enable new programs to take full advantage of the 65C816 processor.

It's important to realize that application programs—even programs designed for the standard Apple II—do not run in the 128K of RAM controlled by the Mega II. That part of RAM always runs at the standard 1 MHz speed, because it contains the I/O slots and the display pages. The I/O slots must be able to run Disk II controller cards and other peripheral–card firmware with timing loops designed to run at the standard 1 MHz speed. The display pages have to be synchronized with the video hardware, which also runs at 1 MHz. The I/O and display features are allocated to memory in high–numbered banks to keep the low–numbered banks available for fast RAM for running application programs.

The Fast Processor Interface (FPI) handles addressing and memory refresh for all of the RAM except the 128K controlled by the Mega II. The FPI also handles ROM addressing. Instruction execution in those areas of memory runs at the rate of 2.8 MHz. Whenever the CPU needs to read or write in the Mega II RAM (banks $E0 and $E1), the FPI synchronizes the CPU timing to match the Mega II's 1 MHz clock.

The user always has the option of using the 1 MHz speed for an application (CPU speed is an option in the Control Panel). Note that the program is still executing in the fast part of RAM, but the FPI is operating at the standard speed.

Memory shadowing

For Apple II programs to run in memory banks $00 and $01, those banks must have the same features as the memory in a 128K Apple IIe or an Apple IIc. That means they must include the language–card mapping in the area above $D000, the I/O spaces starting at $C000, and the display buffers for the standard Apple II displays. Here is a puzzle: To make the low–numbered memory banks available as fast memory, the Apple IIGS designers put the hardware for the I/O and the displays into memory banks $E0 and $E1. Programs designed for the Apple II run in banks $00 and $01 (as main and auxiliary memory), and don't address any other banks. How can such programs operate I/O and displays?

Note: All I/O in an Apple II is memory mapped. Certain memory locations are attached to I/O devices, and I/O operations are just memory read and write instructions.

The designers of the Apple IIGS devised a technique so that programs running in the fast part of memory (banks $00 to $7F) can operate the I/O and display features implemented in the slow part of memory (banks $E0 and $E1). The technique is called memory shadowing, and here's how it works. When shadowing is selected for a specific area, the Apple IIGS hardware executes any instruction that writes into that area of bank $00 or $01 by writing both there and into the same address in bank $E0 or $E1. Because the memory in banks $E0 and $E1 is synchronized to the video hardware, the instruction must execute at the slow speed.

Display shadowing works a little differently from I/O shadowing. For I/O shadowing, both reading and writing are slowed down. For display shadowing, the slowdown affects only instructions that *write* in the shadowed areas; the CPU still *reads* from the display areas of banks $00 and $01 at the faster speed.

So that existing application programs will run on the Apple IIGS, the operating system turns shadowing on whenever it loads an old–style application.

Memory maps

The memory maps in Figures 6–3 and 6–4 show the RAM and ROM areas indicated in Figures 6–1 and 6–2. The 128K of fast RAM on the shaded side of Figure 6–2 corresponds to memory banks $00 and $01; the (fast) RAM on the memory expansion card begins in bank $02 and can extend as high as bank $7F.

The slow RAM controlled by the Mega II corresponds to memory banks $E0 and $E1. Those banks contain the video display pages and the memory locations allocated to the I/O expansion slots. In addition, the built–in firmware also uses RAM in banks $E0 and $E1.

To give application programs full access to the low–numbered banks, the Apple IIGS designers allocated system memory in the high–numbered banks. The system ROM is in banks $FE and $FF. System ROM includes Applesoft, the Monitor, built–in port firmware, and the ROM portion of the Toolbox. Banks $F0 through $FD are allocated to ROM on a memory expansion card, which is used for additional system firmware and for applications stored as ROM Disk files.

Memory for standard Apple II programs

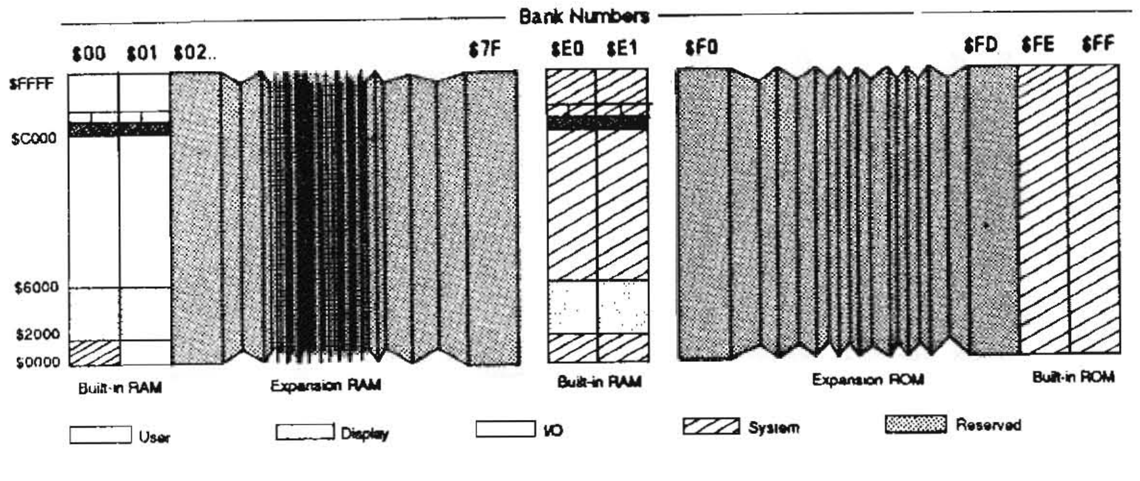
The feature of the Apple IIGS that makes it possible for it to run standard Apple II programs is the implementation of the standard 128K Apple II memory map in the 65C816's expanded memory space. This is done by configuring two of the 64K memory banks to look like the RAM in a 128K Apple IIe: banks $00 and $01, as shown in Figure 6-3.

To make two memory banks on the Apple IIGS work like the main and auxiliary memory in an Apple IIe, those banks must have memory shadowing in effect for I/O spaces and for the standard Apple II text and graphics display pages. (The Super Hi-Res graphics display is not a standard Apple II display and is not normally used with Apple II programs.)

When the user boots up an Apple II program on the Apple IIGS, the firmware sets up memory banks $00 and $01 as main and auxiliary memory, with language-card spaces, display buffers, and the I/O space at hex $Cxxx. The firmware also sets the direct page (zero page) and stack locations to $0000 and $0100 in bank $00.

Programs written for 8-bit Apple II's don't use RAM outside the main and auxiliary banks. To make additional memory useful with such programs, ProDOS 8 uses the additional memory as a mass-storage volume named /RAM5.

Figure 6-3. Memory map for standard Apple II programs

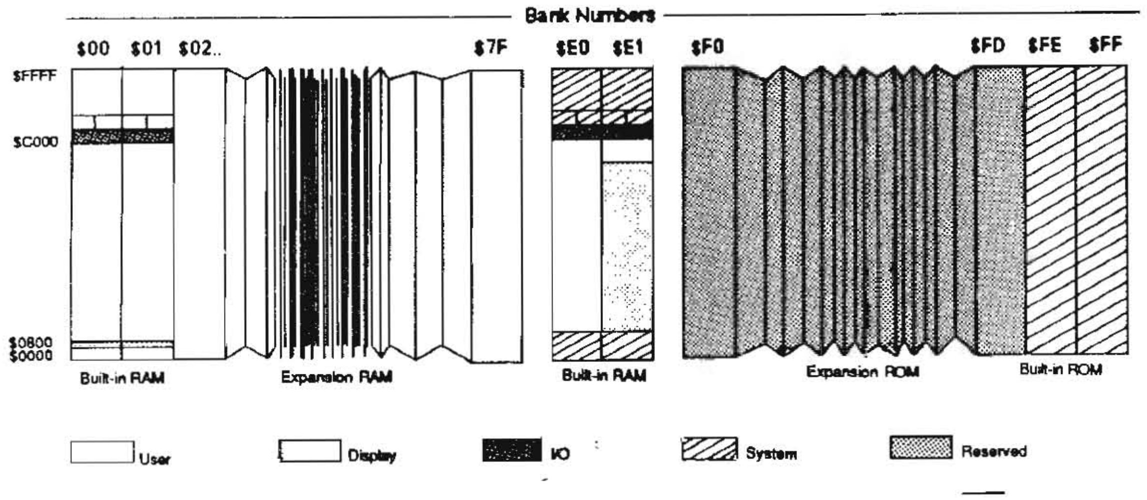


Memory for new programs

New application programs written to use the full capabilities of the Apple IIGS don't have the restrictions of programs written for the standard Apple II. New programs can occupy memory in banks $00 and $01, parts of banks $E0 and $E1, and all of the expansion RAM in banks $02 through $7F. The applications can call the Memory Manager to obtain additional memory in those areas.

Figure 6–4 shows the areas of memory available to applications written specifically for the Apple IIGS. Notice that banks $00 and $01 still have shadowing in I/O space and text Page 1. Those areas must be shadowed for proper operation of interrupts and peripheral cards. Also notice that the expansion RAM (banks $02 through $7F, if present) is available as contiguous memory space.

Figure 6–4. Memory map for new Apple IIGS programs



A reminder: To ensure compatibility with desk accessories and other co–resident routines, Apple IIGS applications that need additional memory must request it from the Memory Manager. (The System Loader calls the Memory Manager to obtain memory space needed for loading initial program segments.)

Chapter 7

Program Environments

The *program environment* is the combination of all of the aspects of the machine that affect the operation of the program. Many of the things that make up the program environment are fixed: for example, the fact that memory is addressed as bytes, or the fact that all I/O is memory mapped. This chapter describes those aspects of the program environment that can be changed from one application to another.

Environment options

Programs running on the Apple IIGS will usually be of two basic types: programs that can also run on 8–bit Apple II's, and programs that can run only on the Apple IIGS. While the environments for those two program types are the ones used most often, they are not the only ones possible, and there is no single master switch for changing from one to the other. The program environment has many aspects, and programs can change any of them independently of the others.

> **Note:** There are two operating systems for the Apple IIGS, corresponding to the two types of programs: ProDOS 8 for 8–bit programs, and ProDOS 16 for 16–bit programs. Chapter 8 includes brief descriptions of the operating systems.

The aspects of the environment that a program can change are

- the microprocessor mode, register sizes, and values in bank registers
- the locations and sizes of the stack and direct page
- the execution speed
- operation of the language card and I/O spaces
- the display memory spaces, including choice of displays and shadowing

The following sections describe those aspects of the program environment.

Microprocessor options

Several of the conditions that are different in the different environments are attributes of the microprocessor. Those include the microprocessor mode, the register sizes, the bank register values, and the locations and sizes of the stack and direct page.

Microprocessor modes

The 65C816 microprocessor can operate in two different modes: *native mode,* with all of its new features, and *6502 emulation mode,* for running programs written for 8–bit Apple IIs.

For more information about the operating modes of the 65C816, refer to the manual *Apple IIGS Hardware Reference.* You might also want to read a reference book about the 65C816 itself.

The 65C816 has three flags named e, m, and x that programs use to control its operating modes. You put the 65C816 into 6502 emulation mode by setting the e flag to 1. When you do that, the 65C816 automatically makes the accumulator and index registers 8 bits wide. It also makes the stack only 256 bytes long, like the stack in the 6502. In emulation mode, the direct page and the stack are automatically at locations $0000 and $0100 in bank $00.

Setting the e flag to 0 puts the 65C816 into native mode. In native mode, a program can make the stack and direct page larger than 256 bytes and can put them anywhere in memory bank $00.

Register sizes

In the 65C816 processor's native mode, the widths of the accumulator and index registers are controlled by the m and x flags. In the Apple IIGS, both the m and x flags are normally set to zero, making the registers sixteen bits wide. Applications running in native mode can change either of those flags to make the accumulator or the index registers only eight bits wide, but there is normally no reason for an application to do so, even though some system routines work that way.

When running applications written for it, the Apple IIGS normally operates with 16–bit accumulator and index registers. When running 8–bit Apple II programs, the system switches the processor to emulation mode, which automatically forces the register widths to eight bits. (In emulation mode, the m and x flags have no effect.)

Bank register values

Applications written specifically for the Apple IIGS can use any banks in memory by setting the program bank register and data bank register appropriately. When running 8–bit Apple II programs, the system firmware sets both the program bank and the data bank to bank $00.

Stack and direct page

For programs written for standard Apple II's, the stack and direct page must be in their proper 6502 locations, and the stack must be 256 bytes long. For programs written specifically for the Apple IIGS, the size of the stack and the locations of the stack and direct page within bank $00 are at the discretion of the application.

When running the 65C816 in native mode, you can locate the stack anywhere between $0800 and $BFFF in bank $00. If you switch to emulation mode, the processor automatically sets the upper half of the stack pointer to $01. When you then switch back to native mode, the upper half of the stack pointer remains set to $01, and your original stack pointer is lost.

When you switch to emulation mode, you have to save your native–mode stack pointer temporarily, then set the stack pointer to the emulation–mode stack and push the native–mode stack pointer onto the emulation–mode stack. After doing that, you switch the processor to emulation mode. To switch back from emulation mode to native mode, you reverse the process: First switch to full native mode, then pull the native–mode pointer off the emulation–mode stack and transfer the 16–bit value to the stack pointer.

Note: Never use the main and auxiliary switches in native mode; doing so prevents the firmware tools from working properly. When setting up the change from native to emulation mode, you have to use the emulation–mode stack in main memory, that is, bank $00.

Important: You must always have interrupts disabled while you are manipulating the stack pointer.

Execution speeds

The microprocessor in the Apple IIGS can operate at either of two clock speeds: the standard Apple II speed, 1 MHz, and the faster speed of 2.8 MHz. For programs running in RAM, a few clock cycles are used for refreshing RAM, reducing the fast speed to an effective value of about 2.5 MHz. System firmware, running in ROM, runs at the full 2.8 MHz.

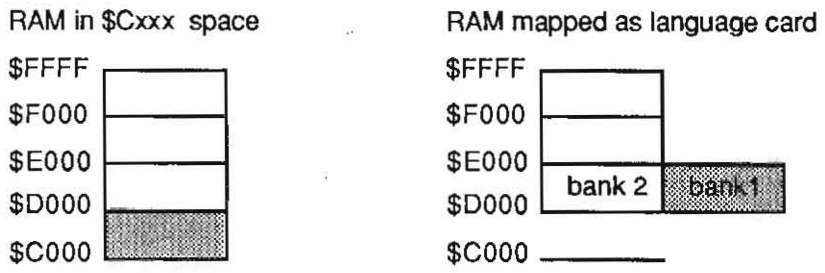
There are three different ways of changing the operating speed. First, the user can use the Control Panel to set the speed. Second, if a slot has a Disk II controller card in it, the firmware switches to the 1 MHz speed whenever that slot is active, so that the disk controller will work correctly. Third, programs can change the clock speed by changing the high bit of the Configuration register, a **control register** in location $C036.

Control registers are located in the I/O space ($Cxxx) in bank $E0; they are accessible from bank $00 if I/O shadowing is on. For more information about the control registers, refer to the *Apple IIGS Hardware Reference.*

Language–card and I/O spaces

Shadowing of the I/O and language card spaces is controlled by the IOLC bit in the Shadow register, a control register located at $C035 in bank $E0: See Table 7–1. The IOLC bit is normally set to zero, enabling I/O in the $Cxxx space and mapping the 4K of RAM that would ordinarily occupy that space into a second bank of RAM in the $Dxxx space, as shown in Figure 7–1. That configuration of the high 16K of RAM is called the language card, after the first Apple II product that provided RAM memory in those locations.

Figure 7–1. Memory map of language–card RAM



Implications for interrupts

Part of the interrupt routines are in ROM in the I/O space at $C07x. For that ROM code to operate, I/O must remain enabled in the $Cxxx part of bank $00 and the high 16K of RAM must stay mapped as a language card; that is, the IOLC bit of the Shadow register must be zero. If a program changes the IOLC bit so it can use RAM in the $Cxxx space, the interrupt routines won't work. IOLC shadowing must be left on even by programs running in native mode, which don't otherwise use the language–card mapping.

Standard Apple II display memory

An application running on the Apple IIGS can use any of the display modes available on 128K Apple II's or the new Super Hi–Res display. Of course, a typical application will use only one or two display modes, so it can disable the rest.

Applications written for 8–bit Apple II's run in banks $00 and $01, but the hardware for video displays uses memory in banks $E0 and $E1. For those applications, the firmware sets shadowing on for those display spaces, so that when the application writes into a display page in bank $00 or $01, the hardware also writes to the same location in bank $E0 or $E1.

The program–selection routine in the Apple IIGS automatically sets the display shadowing appropriately for the operating system that it is loading: On for DOS 3.3, UCSD Pascal, and ProDOS 8, and off for ProDOS 16. When the startup routine sets display shadowing on, it sets shadowing for all standard display pages. An application can turn off shadowing of individual display pages by setting individual bits in the Shadow register, as shown in Table 7–1.

Table 7–1. The Shadow register (location $C035)

Bit	**Function (1 = Inhibit)**
7	(reserved — read undefined, must write zero)
6	IOLC (I/O and language card) operation
5	(reserved — read undefined, must write zero)
4	Auxiliary Hi–Res Pages 1 and 2
3	Super Hi–Res graphics space
2	Hi–Res graphics Page 2
1	Hi–Res graphics Page 1
0	Text Pages 1 and 1X

Super Hi–Res display memory

The Super Hi–Res display is a new graphics display that has several advantages over the standard Apple II displays. While its two modes have resolutions that are only slightly higher than the resolutions of standard Hi–Res and Double Hi–Res, there is no interference between adjacent colors, so Super Hi–Res displays looks much clearer than Hi–Res or Double Hi–Res. It is also easier to program, because it maps entire bytes onto the screen, instead of just seven bits, and its memory map is linear and continuous. Even though Super Hi–Res does cost a little more, occupying 32K of RAM, you'll probably want to use it anyway, because it is supported by the desktop tools.

Shadowing for Super Hi–Res display

The Super Hi–Res display uses locations $4000 through $BFFF in bank $E1 and is normally not shadowed. An application can turn shadowing on and off for the Super Hi–Res display by means of the Shadow register. When shadowing is on for the Super Hi–Res display, applications can write to that display space in bank $01 (auxiliary memory).

A reminder: Applications that use the QuickDraw II routines in the Apple IIGS Toolbox for their displays should have display shadowing off. The QuickDraw II routines write directly to the Super Hi–Res display space in bank $E1, so no shadowing is needed.

Linear memory map

To make life easier for the graphics programmer, there is an option to make the addresses in the Super Hi–Res display memory map onto the display in a simple linear fashion. Bit 6 of the New Video register controls the linear mapping option (1 to enable, 0 to inhibit). Of course, applications that use QuickDraw II don't have to set the video control bits; QuickDraw II takes care of that itself.

Note: The linear–mapping option is not compatible with standard Hi–Res and Double Hi–Res graphics.

Apple II: Memory mapping in the standard display modes is of a byzantine complexity. First, adjacent rows of dots or characters on the screen are not stored in adjacent areas of memory, making it necessary for display routines to calculate the starting location for each row. Also, Hi–Res graphics has the added handicap of 7–bit bytes; that is, only seven bits of each byte are displayed. This makes it harder for display routines to calculate the address of a byte that corresponds to a position on the screen. Double Hi–Res is further burdened with the necessity to alternate between memory banks when addressing adjacent locations on the screen. By using linear mapping, Super Hi–Res display routines avoid these problems.

Table 7–2. New Video register (location $C029)

Bit	Function
7	Enables Super Hi–Res graphics display
6	Enables linear mapping for Super Hi–Res graphics
5	Inhibits color in standard Apple II displays
1–4	(reserved)
0	Enables bank latch (used by system)

Mixing environments

Despite the profound differences between the different program environments on the Apple IIGS, many operating features are similar. It is therefore possible to enhance existing Apple II programs so that they can take advantage of Apple IIGS features such as desk accessories and program tools.

Specifically, the Toolbox routines are accessible not only from application programs written specifically for the Apple IIGS, but also from programs running in 6502 emulation mode. It is therefore possible to modify existing 6502 programs, adding Toolbox calls so the programs can use the new features and the desktop user interface. (The tool sets themselves run in native mode; applications running in emulation mode must switch to native mode to make tool calls, and switch back to emulation mode afterward.)

It is even possible to make a hybrid program that runs on either a 6502–based Apple II or on a Apple IIGS, by having the program check to see that it is running on a Apple IIGS before it makes any Toolbox calls.

A similar kind of compatibility is available with desk accessories, which are accessible from standard Apple II programs running with ProDOS 8 or from Apple IIGS programs running with ProDOS 16. There are two kinds of desk accessories: *classic* desk accessories, which can run in any Apple IIGS environment, and *new style* desk accessories, which can run only under ProDOS 16. Each desk accessory has a flag that determines which versions of ProDOS it can run with. In the software hierarchy, the Desk Manager is below ProDOS. When the user invokes a desk accessory, the Desk Manager detects which version of ProDOS it is running under and checks to see that the requested desk accessory can run with that version.

Environment summary

The simplest distinction between program environments on the Apple IIGS is between the one used for running programs written for 8–bit Apple II's and the one used for programs written specifically for the Apple IIGS. Table 7–1 is a list of the conditions making up these two program environments. While it is possible for applications to set up other combinations, these two program environments are the only ones the firmware and tools support.

Table 7–1. Apple IIGS program environments

Feature	**8–bit Apple II programs**	**Apple IIGS programs**
CPU mode	Emulation (e=1)	Native (e=0)
Accumulator size	8 bits (e=1)*	16 bits (m=0)
Index register size	8 bits (e=1)*	16 bits (x=0)
Execution speed	1MHz or 2.8 MHz	2.8 MHz
Direct–page address	$0000 in bank $00	Any page in bank $00
Stack address	$0100 in bank $00	Any page from $0800 to $BF00 in bank $00
Stack size	256 bytes	Any size up to $B7FF
Language–card spaces in banks $00 and $01	Yes	Yes
Shadowing of I/O spaces in banks $00 and $01	Yes	Yes
Shadowing of text Pages 1 and 1X	Yes	Yes
Shadowing of Hi–Res graphics pages	Yes	No
Default display	Text	Super Hi–Res
Mapping of Super Hi–Res memory addresses	Normal, for Apple II standard displays	Linear, for super Hi–Res display
RAM available to application	Banks $00 & $01 (plus expansion RAM & parts of banks $E0 & $E1, if modified to run on the Apple IIGS)	Banks $00 & $01, expansion RAM, & parts of banks $E0 & $E1
Use of expansion RAM by application	As RAM Disk (or via Memory Manager, if modified to run on the Apple IIGS)	As RAM Disk or via Memory Manager
Operating system	ProDOS 8, DOS 3.3, or UCSD Pascal	ProDOS 16

*In emulation mode (e=1), the m and x flags are always effectively equal to 1.

Chapter 8

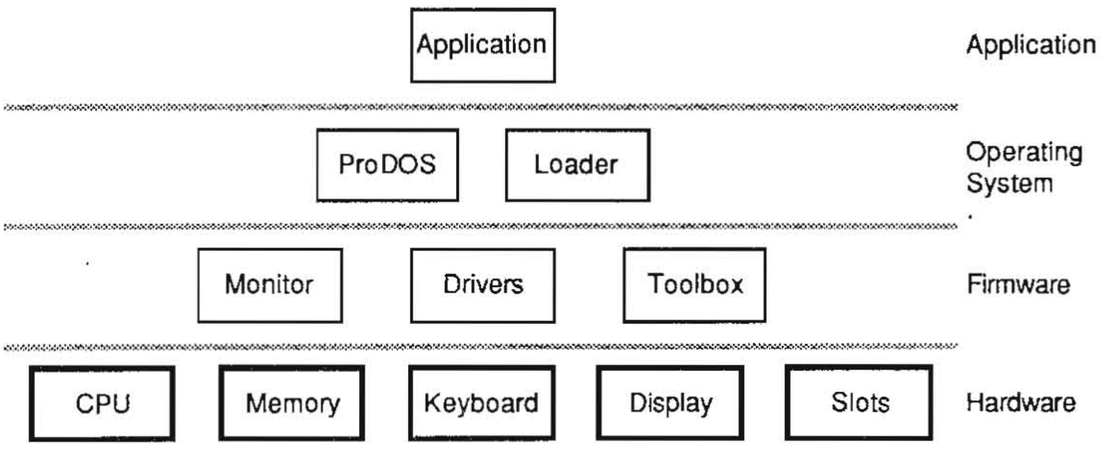
Programs and the Apple IIGS

Since its inception, the Apple II has had built–in firmware to support application programs, and the Apple IIGS continues and extends that tradition. In the past, some applications programmers have bypassed the firmware, taking direct control of the system hardware. This chapter describes some of the ways this is done and some of the problems that arise.

Levels of program operation

You can think of the different levels of program operation on an Apple II as a hierarchy, with a hardware layer at the bottom, firmware and operating–system layers in the middle, and the application at the top. Figure 8–1 illustrates this idea. (The hierarchy in Figure 8–1 is a hierarchy of command levels—generally speaking, higher–level components call on lower–level ones.)

Figure 8–1. Levels of program operation



Program control of the hardware

From the beginning, the Apple II has been an open machine. Not only has it been possible to extend the hardware by means of peripheral cards in expansion slots, but programs have been able to take control of the hardware independently of the built–in firmware.

Whenever the firmware seemed too slow, the application programmer has taken the option of controlling the hardware himself. As later models of Apple II have incorporated more firmware, the need for applications to do it all for themselves has diminished. The Apple IIGS has built–in program support far beyond that available on earlier models of the Apple II. Even so, it is still possible for a program to bypass the firmware and control the hardware directly.

As Figure 8–1 shows, all of the levels except the lowest one are software—even firmware is only software that is permanently resident. As far as the hardware is concerned, one program is much like another, regardless of its origin.

Every part of the Apple IIGS, including the 65C816 microprocessor, control registers in the custom ICs, the display buffers, and the I/O devices, is accessible to the application program. Many of the computer's functions are controlled by soft switches, which are memory locations permanently assigned to some hardware function. The soft switches are described in the *Apple* IIGS *Hardware Reference* .

The phrase "programming on the bare metal" expresses the attitude of programmers who control the hardware themselves. That method has the advantage that everything is done the way the programmer wants it. The obvious disadvantage is that the programmer has to do a lot more work, but a more important one is the increased likelihood that the resulting program will be incompatible either with other programs or with future versions of the computer.

In order to run older programs that were written with this approach, the Apple IIGS continues the Apple II tradition of hardware accessibility at the lowest level. That makes it possible to program the Apple IIGS "on the bare metal." It does not make it advisable.

Using the Apple II firmware

The next level up from the bare metal is the built–in firmware. In the earliest Apple II, this was little more than primitive I/O routines for handling input from the keyboard and formatting text output to the display screen (in 40 columns only, of course). The latest model Apple IIe and Apple IIc include more powerful firmware to handle the 80–column display, the mouse, serial I/O, and disk drives.

Because there have been many changes from model to model, it has generally been easier to maintain compatibility with application programs that make use of the firmware interface, as compared with programs that control the hardware themselves. There is now a strong argument in favor of using the firmware, even when the programmer is dissatisfied with its performance, just to minimize incompatibilities.

A similar argument applies to disk operations. In the past, some applications have set up their own disk file formats and included their own versions of DOS. Apple's new ProDOS for the Apple IIGS is fast and powerful; the cost of going your own way is now quite high compared with the advantages of staying compatible.

Using the Apple IIGS Toolbox

The concept of a program toolbox is new to the Apple II family: The Apple IIGS is the first Apple II to have one. If you are an experienced Apple II developer, even if you have striven to maintain maximum compatibility by using only the firmware interfaces that Apple has provided, you may find the toolbox to be a new way of programming. From that point of view, the Macintosh developer may have an easier time of it. While the toolbox is not the same as the one on the Macintosh, it is similar in concept, and many of its functions are the same.

Chapter 5, "The Apple IIGS Toolbox," is an introduction to the tools. For more information about writing programs in this new way, you should read *Apple IIGS Toolbox Reference,* Volumes 1 and 2, and *Programmer's Introduction to the Apple II GS,* which describes the process of putting a program together.

The advantages of using the Apple IIGS Toolbox are many. Not only do the tools do a lot of the work that the application would otherwise have to do, but the machine itself is set up to use the tools.

Apple IIGS operating systems

There are three kinds of operating systems that can run on the Apple IIGS:

- earlier systems such as DOS 3.3, ProDOS 1.0, and UCSD Pascal, which run the same way on the Apple IIGS as on other models of the Apple II
- the latest version of ProDOS, ProDOS 8, which runs on all current Apple IIs and supports many of the new features of the Apple IIGS
- the new ProDOS for the Apple IIGS, ProDOS 16, which supports all of the new features but runs only on the Apple IIGS

The new ProDOS for the Apple IIGS takes advantage of the 16–bit instructions and large, continuous memory space on the Apple IIGS, making it unable to run on 64K and 128K machines. To make it easy to distinguish between the two kinds of ProDOS, the ProDOS that runs on 8–bit Apple II's is called *ProDOS 8* and the ProDOS for the Apple IIGS is called *ProDOS 16.*

ProDOS 16 is functionally similar to 8–bit versions of ProDOS, but it does not work the same way, so programs that run under an 8–bit ProDOS will not run under ProDOS 16 without suitable modifications. The latest version of 8–bit ProDOS, ProDOS 8, supports 8–bit programs running on the Apple IIGS. The System Loader automatically loads the appropriate version of ProDOS, depending on the type of startup file it finds on the boot disk. Table 8–1 is a summary of the differences between ProDOS 8 and ProDOS 16.

Table 8–1. ProDOS 8 and ProDOS 16 compared

Feature	**ProDOS 8**	**ProDOS 16**
Microprocessor mode	6502 emulation	65C816 native mode
Minimum memory	64K	256K
Maximum memory	128K	8.25 megabytes
Memory management	Bit map in global page	Memory Manager
RAM Disk	Connected	Disconnected
Memory pointer size	2 bytes	4 bytes
System call instruction	JSR into bank $00	JSL into bank $E1
System file suffix	.SYS	.SYS16
System file type	$FF	$B3

Just remember that ProDOS 8 is for 8–bit Apple II applications running on the Apple IIGS, and ProDOS 16 is for Apple IIGS applications.

Note: Even though ProDOS 8 and ProDOS 16 are different, they both use the same disk formats and file structures. Either one can read a file written by the other, except that ProDOS 8 won't start up from the startup files (type $B3) used for ProDOS 16, and ProDOS 16 won't start up from the system files (type $FF) or binary files (type $06) used for ProDOS 8.

The System Loader

The Apple IIGS Toolbox includes the System Loader, a system program that makes full use of the large memory and the standardized load modules on the Apple IIGS. The System Loader, working in conjunction with ProDOS 16 and the Memory Manager, loads and relocates program segments. Programs can be compiled and linked as individual segments, some of which can be loaded dynamically, as needed.

Apple IIGS ProDOS 16 Reference includes information about the System Loader.

Load segments can be either static or dynamic. Static segments remain in memory all during program execution. The System Loader loads all of a program's static segments when it first loads the program.

The System Loader doesn't load dynamic segments until they are called for during program execution. The program can request specific segments by calling the System Loader, or the loader can use the segment jump table, which is a special segment set up by the linker to deal with references across segment boundaries.

Apple II compatibility

One of the most important features of the Apple IIGS is its ability to run standard Apple II programs. The Apple IIGS incorporates all the features of the Apple IIe and most of the features of the Apple IIc, including the ability to support either 5.25–inch or 3.5–inch disk drives connected to its disk port.

Running existing programs

Users can boot standard Apple II program disks on the Apple IIGS and run most programs without modification. Such programs will not use any of the new features of the Apple IIGS except its ability to run 2.5 times as fast. The programs will be running in 6502 emulation mode and the memory space available to them will be configured just like the 128K of RAM in an Apple IIc.

Users can invoke the Control Panel desk accessory to change the I/O slot assignments to use with their Apple II programs. They can also change the text display colors and the operating speed. For example, they'll probably want to run their business programs at the fast speed, but they may want to slow down to normal speed for games.

Enhancing existing programs

Even for programs running in emulation mode, all of the new features of the Apple IIGS are available. The only trouble is that programs written for earlier Apple II's don't include routines that make use of the new features. As a developer, you can modify your programs and add such routines while maintaining compatibility with older models of Apple II. Modified programs can check to see what kind of Apple II they are running on and take advantage of the new features if they are running on a Apple IIGS.

> **Note:** To find out what kind of Apple II they are running on, programs can read the ID bytes at locations $FBB3, $FBC0, and $FBBF in ROM. Assembly–language programs can execute a JSR (jump to subroutine) to location $FE1F in ROM, then branch on the state of the carry bit: It will be one for any 8–bit Apple II and zero for the Apple IIGS. For more information, refer to the *Apple IIGS Firmware Reference.*

Of course, if you're going to modify an existing Apple II program, some of the new features make more sense than others. For example, changing the program to add routines that use the new 16–bit instructions would require a lot of work—work that would probably be better spent on writing a new version of the program. On the other hand, modifying a program so it could use the built–in tools might be worthwhile. The decision should be based on whether the resulting program could still fit in memory on an Apple IIe or Apple IIc. If it couldn't, it would be better to make a new version of the program just for the Apple IIGS.

Another way to make an application run on either an 8–bit Apple II or on a Apple IIGS is to make a new version that runs only on the Apple IIGS and put both versions on a single disk. The appropriate version would run, depending on what kind of machine the disk was booted on. The cold–start routine on the Apple IIGS looks for a system file with the suffix .SYS16 and loads ProDOS 16 if it is present; an 8–bit Apple II boots with a .SYS file and gets ProDOS 8. Refer to the *Apple IIGS ProDOS 16 Reference* for more information.

Chapter 9

Apple IIGS Development Environment

The development environment is the software that you use for developing programs on the Apple IIGS. The development environment includes two kinds of programs: first, the language compilers and assemblers, and second, programs that all developers use, regardless of which language they are using. Each compiler or assembler has its own manual. The programs that are used with any of the programming languages are described in the *Apple IIGS Programmer's Workshop.*

Several features of the Apple IIGS help you with program development. First of all, there is a standard format for object files, regardless of their source. Then there are the linker and the System Loader that, together with standard load files, make it possible to create modular programs with relocatable segments and to combine segments written in different source languages. The languages available on the Apple IIGS include assembly language and C. To provide a consistent programming environment, there is the Apple IIGS Programmer's Workshop (CPW). The workshop includes the operating shell for controlling the language compilers, along with the program editor, the debugger, the linker, and utility programs.

Program modularity

The basis of the Apple IIGS development environment is the standard file formats. The standard formats make it possible to use many different programming languages on the Apple IIGS. Along with the System Loader, they also make possible program segmentation, with relocatable segments that can be loaded dynamically during program execution.

Creating a program is a multi-step process. First, the program is written in the form of one or more source files. Compilers and assemblers process the source files and produce object files. The linker then takes the program object files, along with any appropriate library object files, and produces one or more load files. It is the load files that get loaded into memory when the program is executed.

Object files and load files

Assemblers and compilers produce object files. The linker combines object segments from one or more object files and produces a load file. Separate segments in object files can be combined into a single segment by the linker. That makes it possible to write the program as separate parts and recompile only the affected part whenever you make a change.

In addition to the program object files, there can be library files containing general-purpose segments used by several programs. The linker can search the library and extract the segments needed by the program.

Each load file consists of one or more segments, which can be static or dynamic. Static segments must remain in memory while the program is running, but dynamic segments can be loaded and unloaded individually as they are needed.

Program segments can also be capable of being loaded anywhere in memory, that is, they can be relocatable. The actual relocation is carried out at run time by the System Loader. Each load segment contains both the program code and a relocation dictionary, which the System Loader uses to recalculate addresses when it loads relocatable segments. The load file format was designed to make dynamic loading as fast as possible.

Programming languages

The Apple IIGS development environment does not restrict developers to a single programming language. You can use any programming language for which there is a compiler that produces object files in the Apple IIGS object module format. The languages available from Apple include assembly language and C.

Assembler

The CPW Assembler executes under the control of the CPW Shell. The assembler supports the Apple IIGS standard object file format and relocatable segments.

A **macro assembler** can combine multiple assembly-language instructions into single pseudo-instructions—*macros*—that make it easier to write assembly-language programs.

Conditional assembly is the ability to define macros or other pieces of code such that they assemble differently under different conditions.

The CPW Assembler is a full-featured macro assembler. It supports the instruction sets and addressing modes of the 65816 microprocessor. The assembler includes

- an extensive set of assembler directives
- macros and conditional assembly
- support for segments, which can be either code or data
- partial assembly, so that changes do not require reassembly of the entire program
- support for library files that the linker searches in case of unresolved references

Note: The CPW Assembler is not a version of Apple's EdAsm™ Assembler for the Apple II.

C compiler

The high-level language in the Apple IIGS Programmer's Workshop is C. Programs written in C can easily include sections written in assembly language and in Pascal.

CPW C is similar to Macintosh Workshop C. The Apple IIGS Interface Library provides an interface to the Apple IIGS Toolbox that is functionally similar to the Macintosh Interface Libraries.

There are a few differences from Macintosh C, such as

- The size of *int* variables is 16 bits.
- The format of the *pascal* declaration is different.
- Function results are returned in a global variable, rather than the stack.
- Register variables are not available.

The *Apple IIGS Workshop C Reference* includes definitions of the C language and of the standard C library and the Apple IIGS Interface Library. It describes the differences between Apple IIGS C and a standard C: the Berkeley 4.2 BSD VAX implementation of the Portable C Compiler.

Other compilers

There can be a Apple IIGS compiler for almost any programming language; all the compiler has to do is produce object files compatible with the Apple IIGS object file format. Languages for which compilers could be written include Pascal, BASIC, Fortran, Logo, Cobol and Lisp.

Apple IIGS Programmer's Workshop

The Apple IIGS Programmer's Workshop (CPW) is a set of programs that Apple provides to make it easier to develop applications for the Apple IIGS. The programs in the programmer's workshop are

- shell
- editor
- linker
- debugger
- utilities

These programs are all described in the manual *Apple IIGS Programmer's Workshop.*

Shell

The Shell provides the user interface that enables you to execute other CPW programs and to perform various housekeeping functions such as copying files. You type in commands in the old-fashioned way.

The shell also acts as an extension to ProDOS 16, providing additional support functions for programs such as compilers, assemblers, and linkers running under the shell. Those functions include

- parameter-passing between programs and the shell
- reading and setting the language type of a source file
- getting file names by using wildcards
- passing control to other system programs
- moving, copying, and deleting files and subdirectories
- renaming files
- changing prefixes
- listing files and directories
- changing the ProDOS file type of a file

The shell supports programmable command or Exec files that can be used to execute any number of shell commands. The Exec files can include parameter passing and conditional execution statements. The shell also supports redirection of input and output and pipelining of CPW programs.

Editor

The CPW Editor is a text editor for use with the CPW Assembler and compilers. To use the editor, you invoke it from the shell. If you select a pre-existing file for editing, the editor is automatically set to the language of that file. Other wise, the editor is set to the last language used or the last language selected with a shell command. You can also use the editor to create Exec files.

Linker

The CPW Linker reads object files created by the CPW Assembler or by the CPW C compiler and generates load files. For relocatable code, the linker resolves external references and creates relocation dictionaries. Because the assembler and compiler create object files that conform to the same format, the linker can link together object modules created by any combination of CPW languages.

Normally, you call the linker by a command to the shell that lets you specify a limited number of linker options. You specify parameters for segmentation and printing in the source code itself.

For advanced programmers who need more flexibility than the link command provides, the linker has a command language called *LinkEd*. You can use LinkEd commands to perform such functions as

- extracting segments from object files
- opening and closing output files
- creating static or dynamic segments
- searching libraries
- controlling printing by the linker

Debugger

The CPW Debugger enables you to trace program execution one instruction at a time or run full speed and stop at a breakpoint. Each time the program stops, the debugger displays a disassembly of the code, the contents of a specified area of RAM, and the contents of the microprocessor's registers, stack, and direct page.

The debugger can switch between its own display and the display of the program under test.

Utilities

The programmer's workshop includes several programs that perform functions that cannot be handled by the built-in shell commands. These programs are called utilities, and they include

- CRUNCH: compresses object modules after partial assemblies or compilations
- INIT: initializes a disk
- MACGEN: generates a macro file
- MAKELIB: generates a library file
- DUMPOBJ: lists all routines in an object module or load module

Some of the utility programs require no input from the user other than the name; those programs are treated like any other shell command, and are referred to as *external commands*.

Appendix A

Roadmap to the Apple IIGS Technical Manuals

The Apple IIGS has many advanced features, making it more complex than earlier models of the Apple II. To describe it fully, Apple has produced a suite of technical manuals. Depending on the way you intend to use the Apple IIGS, you may need to refer to a select few of the manuals, or you may need to refer to most of them.

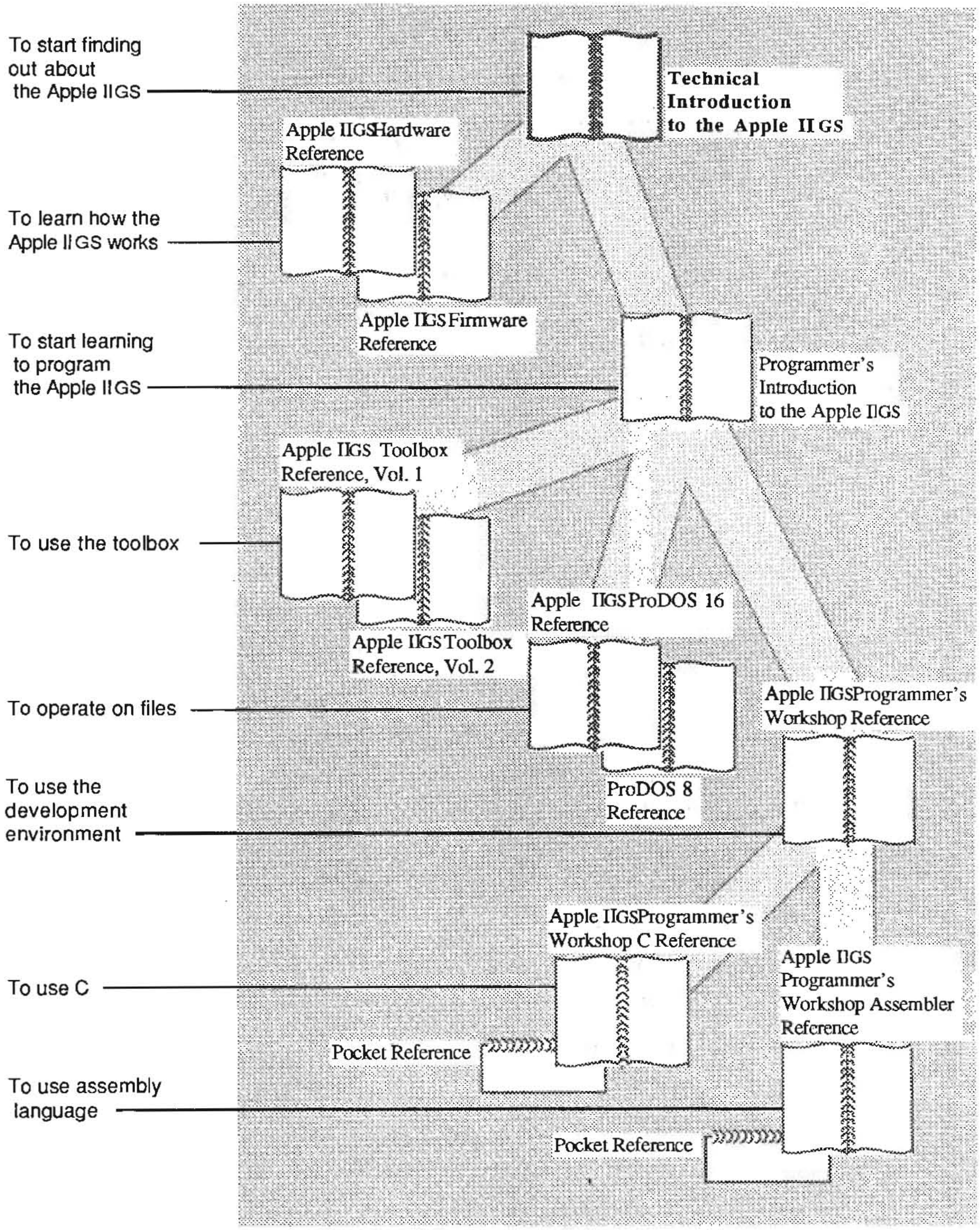
The technical manuals are listed in Table A-1. Figure A-1 is a diagram showing the relationships among the different manuals.

Table A-1. The Apple IIGS Technical Manuals

Title	**Subject**
Technical Introduction to the Apple IIGS	What the Apple IIGS is
Apple IIGS Hardware Reference	Machine internals—hardware
Apple IIGS Firmware Reference	Machine internals—firmware
Programmer's Introduction to the Apple IIGS	Concepts and a sample program
Apple IIGS Toolbox Reference: Volume 1	How the tools work and some Toolbox sepcifications
Apple IIGS Toolbox Reference: Volume 2	More Toolbox specifications
Apple IIGS Programmer's Workshop Reference	The development environment
Apple IIGS Workshop Assembler Reference*	Using the CPW assembler
Apple IIGS Workshop C Reference*	Using C on the Apple IIGS
ProDOS 8 Reference	ProDOS for Apple II programs
Apple IIGS ProDOS 16 Reference	ProDOS and Loader for Apple IIGS
Human Interface Guidelines	Guidelines for the desktop interface
Apple Numerics Manual	Numerics for all Apple computers

*There is a Pocket Reference for each of these.

Figure A-1. Roadmap to the technical manuals


To start finding out about the Apple IIGS
Technical Introduction to the Apple IIGS
Apple IIGS Hardware Reference
To learn how the Apple IIGS works
Apple IIGS Firmware Reference
To start learning to program the Apple IIGS
Programmer's Introduction to the Apple IIGS
Apple IIGS Toolbox Reference, Vol. 1
To use the toolbox
Apple IIGS ProDOS 16 Reference
Apple IIGS Toolbox Reference, Vol. 2
To operate on files
Apple IIGS Programmer's Workshop Reference
To use the development environment
ProDOS 8 Reference
Apple IIGS Programmer's Workshop C Reference
Apple IIGS Programmer's Workshop Assembler Reference
To use C
Pocket Reference
To use assembly language
Pocket Reference

Introductory manuals

These books are introductory manuals for developers, computer enthusiasts, and other Apple IIGS owners who need technical information. As introductory manuals, their purpose is to help the technical reader understand the features of the Apple IIGS, particularly the features that are different from other Apple computers. Having read the introductory manuals, the reader will refer to specific reference manuals for details about a particular aspect of the Apple IIGS.

The technical introduction

The *Technical Introduction to the Apple IIGS* is the first book in the suite of technical manuals about the Apple IIGS. It describes all aspects of the Apple IIGS, including its features and general design, the program environments, the Toolbox, and the development environment.

Where the *Apple IIGS Owner's Guide* is an introduction from the point of view of the user, the *Technical Introduction* describes the Apple IIGS from the point of view of the program. In other words, it describes the things the programmer has to consider while designing a program, such as the operating features the program uses and the environment in which the program runs.

The programmer's introduction

When you start writing programs that use the Apple IIGS user interface (with windows, menus, and the mouse), the *Programmer's Introduction to the Apple IIGS* provides the concepts and guidelines you need. It is not a complete course in programming, only a starting point for programmers writing applications for the Apple IIGS. It introduces the routines in the Apple IIGS Toolbox and the program environment they run under. It includes a sample **event-driven program** that demonstrates how a program uses the Toolbox and the operating system.

An **event-driven program** waits in a loop until it detects an event such as a click of the mouse button.

Machine reference manuals

There are two reference manuals for the machine itself: the *Apple IIGS Hardware Reference* and the *Apple IIGS Firmware Reference*. These books contain detailed specifications for people who want to know exactly what's inside the machine.

The hardware reference manual

The *Apple IIGS Hardware Reference* is required reading for hardware developers, and it will also be of interest to anyone else who wants to know how the machine works. Information for developers includes the mechanical and electrical specifications of all connectors, both internal and external. Information of general interest includes descriptions of the internal hardware, which provide a better understanding of the machine's features.

The firmware reference manual

The *Apple IIGS Firmware Reference* describes the programs and subroutines that are stored in the machine's read-only memory (ROM), with two significant exceptions: Applesoft BASIC and the Toolbox, which have their own manuals. The *Firmware Reference* includes information about interrupt routines and low-level I/O subroutines for the serial ports, the disk port, and for the DeskTop Bus, which controls the keyboard and the mouse. The *Firmware Reference* also describes the Monitor, a low-level programming and debugging aid for assembly-language programs.

The Toolbox manuals

Like the Macintosh, the Apple IIGS has a built-in Toolbox. The *Apple IIGS Toolbox Reference,* Volume 1, introduces concepts and terminology and tells how to use some of the tools. It also tells how to write and install your own tool set. The *Apple IIGS Toolbox Reference,* Volume 2, contains information about the rest of the tools.

Of course, you don't have to use the Toolbox at all. If you only want to write simple programs that don't use the mouse, or windows, or menus, or other parts of the **desktop user interface**, then you can get along without the Toolbox. However, if you are developing an application that uses the desktop interface, or if you want to use the Super Hi-Res graphics display, you'll find the Toolbox to be indispensable.

In applications that use the **desktop user interface,** commands appear as options in pull-down menus, and material being worked on appears in rectangular areas of the screen called windows. The user selects commands or other material by using the mouse to move a pointer around on the screen.

The Programmer's Workshop manual

The development environment on the Apple IIGS is the Apple IIGS Programmer's Workshop (CPW). CPW is a set of programs that enable developers to create and debug application programs on the Apple IIGS. The *Apple IIGS Programmer's Workshop Reference* includes information about the parts of the workshop that all developers will use, regardless which programming language they use: the shell, the editor, the linker, the debugger, and the utilities. The manual also tells how to write other programs, such as custom utilities and compilers, to run under the CPW Shell.

The CPW reference manual describes the way you use the workshop to create an application and includes a sample program to show how this is done.

Programming–language manuals

Apple is currently providing a 65C816 assembler and a C compiler. Other compilers can be used with the workshop, provided that they follow the standards defined in the *Apple IIGS Programmer's Workshop Reference.*

There is a separate reference manual for each programming language on the Apple IIGS. Each manual includes the specifications of the language and of the Apple IIGS libraries for the language, and describes how to write a program in that language. The manuals for the languages Apple provides are the *Apple IIGS Workshop Assembler Reference* and the *Apple IIGS Workshop C Reference.*

Operating–system manuals

There are two operating systems that run on the Apple IIGS: ProDOS 16 and ProDOS 8. Each operating system is described in its own manual: *ProDOS 8 Reference* and *Apple IIGS ProDOS 16 Reference.* ProDOS 16 uses the full power of the Apple IIGS and is not compatible with earlier Apple IIs. The ProDOS 16 manual includes information about the System Loader, which works closely with ProDOS 16. If you are writing programs for the Apple IIGS, whether as an application programmer or a system programmer, you are almost certain to need the *ProDOS 16 Reference.*

ProDOS 8, previously just called *ProDOS,* is compatible with the models of Apple II that use 8-bit CPUs. As a developer of Apple IIGS programs, you need to use ProDOS 8 only if you are developing programs to run on 8-bit Apple II's as well as on the Apple IIGS.

All-Apple manuals

In addition to the Apple IIGS manuals mentioned above, there are two manuals that apply to all Apple computers: *Human Interface Guidelines* and *Apple Numerics Manual.* If you develop programs for any Apple computer, you should know about those manuals.

The *Human Interface Guidlines* manual describes Apple's standards for the desktop interface of programs that run on Apple computers. If you are writing an application for the Apple IIGS, you should be familiar with the contents of this manual.

The *Apple Numerics Manual* is the reference for the Standard Apple Numeric Environment (SANE), a full implementation of the IEEE standard floating-point arithmetic. The functions of the Apple IIGS SANE tool set match those of the Macintosh SANE package and of the 6502 Assembly Language SANE software. If your application requires accurate arithmetic, you'll probably want to use the SANE routines in the Apple IIGS. The *Apple IIGS Tools Reference* tells how to use the SANE routines in your programs. The *Apple Numerics Manual* is the comprehensive reference for the SANE numerics routines. A description of the version of the SANE routines for the 65816 is available through the Apple Programmer's and Developer's Association, administered by the A.P.P.L.E. cooperative in Renton, Washington.

Note: The address of the Apple Programmer's and Developer's Association is 290 SW 43rd Street, Renton, WA 98055, and the telephone number is (206) 251-6548.

Appendix B

Summary of Program Environments

The simplest distinction between program environments on the Apple IIGS is between the one used for running programs written for 8–bit Apple II's and the one used for programs written specifically for the Apple IIGS. Table B-1 is a list of the conditions making up these two program environments. (This table is a duplicate of Table 7–1. For more information about the program environment, refer to Chapter 7.)

Table B-1. Apple IIGS program environments

Feature	**8–bit Apple II programs**	**Apple IIGS programs**
CPU mode	Emulation (e=1)	Native (e=0)
Accumulator size	8 bits (e=1)*	16 bits (m=0)
Index register size	8 bits (e=1)*	16 bits (x=0)
Execution speed	1MHz or 2.5 MHz	2.5 MHz
Direct–page address	$0000 in bank $00	Any page in bank $00
Stack address	$0100 in bank $00	Any page from $0800 to $BF00 in bank $00
Stack size	256 bytes	Any size up to $B7FF
Language–card spaces in banks $00 and $01	Yes	Yes
Shadowing of I/O spaces in banks $00 and $01	Yes	Yes
Shadowing of text Pages 1 and 1X	Yes	Yes
Shadowing of Hi-Res graphics pages	Yes	No
Default display	Text	Super Hi-Res
Mapping of Super Hi-Res memory addresses	Normal, for Apple II standard displays	Linear, for super Hi-Res display
RAM available to application	Banks $00 & $01 (plus expansion RAM & parts of banks $E0 & $E1, if modified to run on the Apple IIGS)	Banks $00 & $01, expansion RAM, & parts of banks $E0 & $E1
Use of expansion RAM by application	As RAM Disk (or via Memory Manager, if modified to run on the Apple IIGS)	As RAM Disk or via Memory Manager
Operating system	ProDOS 8, DOS 3.3, or UCSD Pascal	ProDOS 16

*In emulation mode (e=1), the m and x flags are always effectively equal to 1.

Glossary

This glossary defines technical terms used in this book. Boldfaced terms within a definition are defined elsewhere in the glossary.

accumulator: The register in a computer's central processor or microprocessor where most computations are performed.

ACIA: Acronym for *Asynchronous Communications Interface Adapter,* a type of communications IC used in some Apple computers. See **SCC.**

acronym: A word formed from the initial letters of a name or phrase, such as ROM (from *read–only memory).*

ADC: See **analog–to–digital converter.**

address: A number that specifies the location of a single **byte** of memory. Addresses can be given as decimal integers or as hexadecimal integers. A 64K system has addresses ranging from 0 to 65535 (in decimal) or from $0000 to $FFFF (in hexadecimal). The letter *X* in an address stands for all possible values for that digit. For example, $Dxxx means all the addresses from $D000 through $DFFF.

American Simplified Keyboard: See **Dvorak keyboard.**

American Standard Code for Information Interchange: See **ASCII.**

analog: (adj) Varying smoothly and continuously over a range, rather than changing in discrete jumps. For example, a conventional 12–hour clock face is an analog device that shows the time of day by the continuously changing position of the clock's hands. Compare **digital.**

analog RGB: A type of color video monitor that accepts separate analog signals for the red, green, and blue color primaries. The intensity of each primary can vary continuously, making possible many shades and tints of color.

analog signal: A signal that varies continuously over time, rather than being sent and received in discrete intervals. Compare **digital signal.**

analog–to–digital converter (ADC): A device that converts quantities from analog to digital form. For example, computer hand controls convert the position of the control dial (an analog quantity) into a discrete number (a digital quantity) that changes stepwise even when the dial is turned smoothly.

Apple key: A modifier key on the Apple IIGS keyboard, marked with both an Apple icon and a spinner, the icon used on the equivalent key on some Macintosh keyboards. See **Open Apple.**

Applesoft BASIC: The Apple II dialect of the BASIC programming language. An interpreter for creating and executing Applesoft BASIC programs is built into the firmware of computers in the Apple II family.

AppleTalk: Apple's local-area network for Apple II and Macintosh and the LaserWriter and ImageWriter II. Like the Macintosh, the Apple IIGS has the AppleTalk interface built in.

AppleTalk connector: A piece of equipment, consisting of a connection box, a short cable, and an 8–pin miniature **DIN** connector, that enables a Apple IIGS to be part of an AppleTalk network.

Apple II: A family of computers, including the original Apple II, the Apple II Plus, the Apple IIe, the Apple IIc, and the Apple IIGS.

Apple IIc: A transportable personal computer in the Apple II family, with a disk drive and 80–column display capability built in.

Apple IIe: A personal computer in the Apple II family with seven expansion slots and an auxiliary memory slot that allow the user to enhance the computer's capabilities with peripheral and auxiliary cards.

Apple IIe 80–Column Text Card: A peripheral card that plugs into the Apple IIe's auxiliary memory slot and enables the computer to display text as either 40 or 80 characters per line.

Apple IIe Extended 80–Column Text Card: A peripheral card that plugs into the Apple IIe's auxiliary memory slot and enables the computer to display text as either 40 or 80 characters per line while extending the computer's memory capacity by 64K.

Apple II Pascal: A software system for the Apple II family that lets you create and execute programs written in the Pascal programming language. Apple II Pascal was adapted by Apple Computer from the University of California, San Diego, Pascal Operating System (UCSD Pascal).

Apple II Plus: A personal computer in the Apple II family with expansion slots that allow the user to enhance the computer's capabilities with peripheral and auxiliary cards.

application program: A program that enables a person to carry on some work, such as word processing, data base management, graphics, or telecommunication. Compare **system program.**

ASCII: Acronym for *American Standard Code for Information Interchange*, pronounced *ASK–ee*. A code in which the numbers from 0 to 127 stand for text characters. ASCII code is used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

aspect ratio: The ratio of an image's width to its height. For example, a standard video display has an aspect ratio of 4:3.

assembler: A language translator that converts a program written in **assembly language** into an equivalent program in **machine language.** The opposite of a **disassembler.**

assembly language: A low–level programming language in which individual machine–language instructions are written in a symbolic form that's easier to understand than machine language itself. Each assembly–language instruction produces one machine–language instruction. See also **machine language.**

asynchronous: Not synchronized by a mutual timing signal or clock. Compare **synchronous.**

Asynchronous Communications Interface Adapter: See **ACIA.**

auxiliary slot: The special expansion slot inside the Apple IIe used for the Apple IIe 80–Column Text Card or Extended 80–Column Text Card, and also for the **RGB monitor** card. The slot is labeled AUX. CONNECTOR on the circuit board.

back panel: The rear surface of the computer, which includes the power switch, the power connector, and connectors for peripheral devices.

baud: A unit of data transmission speed: the number of discrete signal state changes per second. Often, but not always, equivalent to *bits per second.* Compare **bit rate.**

binary file: A file whose data is to be interpreted in binary form. Machine–language programs and pictures are stored in binary files.

bit: A contraction of *binary digit*. The smallest unit of information that a computer can hold. The value of a bit (1 or 0) represents a simple two–way choice, such as yes or no, on or off, positive or negative, something or nothing.

bit image: A collection of bits in memory that have a rectilinear graphical representation. The display on the screen is a visible bit image.

bitmap: A set of bits that represents the positions and states of a corresponding set of items; for example, dots in an image. See **bit image.**

bit rate: The speed at which bits are transmitted, usually expressed as *bits per second,* or *bps*. Compare **baud.**

block I/O device: A type of device that reads or writes information in organized groups called blocks, which are typically 512 bytes long. A disk drive is a block device.

boot: Another way to say **start up.** A computer boots by loading a program into memory from an external storage medium such as a disk. *Boot* is short for *bootstrap load,* a term suggestive of the difficulty of initial loading of loader programs into early computers that didn't have built-in firmware in ROM.

bootstrap: See **boot.**

buffer: A holding area in the computer's memory where information can be stored by one program or device and then read at a different rate by another; for example, a print buffer.

bug: An error in a program that causes it not to work as intended. The expression reportedly comes from the early days of computing when an itinerant moth shorted a connection and caused a breakdown in a room–size computer.

bus: A group of wires or circuits that transmit related information from one part of a computer system to another. In a network, a line of cable with connectors linking devices together. A bus network has a beginning and an end. (It's not in a closed circle or T shape.)

buttons: The pushbutton–like images in dialog boxes where you click to designate, confirm, or cancel an action. See also **mouse button.**

byte: A unit of measure of computer data or **memory,** consisting of a fixed number of **bits.** On Apple II systems, one byte consists of eight bits, and a byte can have any value between 0 and 255. The value can represent an instruction, letter, number, punctuation mark, or other character. See also **kilobyte, megabyte.**

call: (v) To request the execution of a subroutine, function, or procedure. (n) A request from the keyboard or from a procedure to execute a named procedure. See **procedure.**

carriage return: An ASCII character (decimal 13) that ordinarily causes a printer or display device to place the next character on the left margin.

carry flag: A status bit in the microprocessor, used as an additional high–order bit with the accumulator bits in addition, subtraction, rotation, and shift operations.

cathode–ray tube: A display device.

central processing unit (CPU): The part of the computer that performs the actual computations in machine language. See **microprocessor.**

character: Any symbol that has a widely understood meaning and thus can convey information. Some characters—such as letters, numbers, and punctuation—can be displayed on the monitor screen and printed on a printer.

chip: See **integrated circuit.**

circuit board: A board containing embedded circuits and an attached collection of integrated circuits (chips).

clock chip: A special chip in which parameter RAM and the current setting for the date and time are stored. This chip is powered by a battery when the system is off, thus preserving the information.

close: To turn a **window** back into the icon that represents it.

CMOS: Abbreviation for *complementary metal oxide silicon,* one of several methods of making integrated circuits out of silicon. CMOS devices are characterized by their low power consumption. CMOS techniques are derived from **MOS** techniques.

code: (1) A number or symbol used to represent some piece of information. (2) The statements or instructions that make up a program.

cold start: The process of starting up the Apple II when the power is first turned on (or as if the power had just been turned on) by loading the **operating system** into main memory, and then loading and running a program. Compare **boot, warm start.**

column: A vertical arrangement of graphics points or character positions on the display.

command: An instruction that causes the computer to perform some action. A command can be typed from a keyboard, selected from a menu with a hand–operated device (such as a mouse), or embedded in a program.

compiler: A language translator that converts a program written in a high–level programming language (source code) into an equivalent program in some lower–level language such as machine language (object code) for later execution.

component: A part; in particular, a part of a computer system.

composite video: A video signal that includes both display information and the synchronization (and other) signals needed to display it. See **NTSC, RGB monitor.**

computer: An electronic device that performs predefined (programmed) computations at high speed and with great accuracy. A machine that is used to store, transfer, and transform information.

computer language: See **programming language.**

conditional assembly: A feature of an assembler that allows the programmer to define macros or other pieces of code such that the assembler assembles them differently under different conditions.

conditional branch: A type of branch instruction whose execution depends on the truth of a condition or the value of an expression.

configuration: (1) The total combination and arrangement of hardware components—CPU, video display device, keyboard, and peripheral devices—that make up a computer system. (2) The software settings that allow various hardware components of a computer system to communicate with each other.

Control key: A specific modifier key on Apple II–family keyboards that produces control characters when used in combination with other keys.

Control Panel: A **desk accessory** that lets you change certain system parameters, such as speaker volume, display colors, and configuration of slots and ports.

control registers: Special registers that programs can read and write, similar to **soft switches.** The control registers are specific locations in the I/O space ($Cxxx) in bank $E0; they are accessible from bank $00 if I/O shadowing is on.

Control–Reset: A combination keystroke on Apple II–family computers that usually causes an Applesoft BASIC program or command to stop immediately.

controller card: A peripheral card that connects a device such as a printer or disk drive to a computer's main logic board and controls the operation of the device.

CPU: See **central processing unit.**

cursor: A symbol displayed on the screen marking where the user's next action will take effect or where the next character typed from the keyboard will appear.

DAC: See **digital–to–analog converter.**

data: information transferred to or from or stored in a computer or other mechanical communications or storage device.

data bits: The bits in a communication transfer that contain information. Compare **start bit, stop bit.**

data format: The form in which **data** is stored, manipulated, or transferred. For example, when data is transmitted and received serially, it typically has a data format of one start bit, five to eight data bits, an optional parity bit, and one or two stop bits.

Data Carrier Detect (DCD): A signal from a DCE (such as a modem) to a DTE (such as an Apple IIc) indicating that a communication connection has been established. See **Data Communication Equipment, Data Terminal Equipment.**

Data Communication Equipment (DCE): As defined by the RS-232-C standard, any device that transmits or receives information. Usually this device is a modem.

Data Set Ready (DSR): A signal from a DCE to a DTE indicating that the DCE has established a connection. See **Data Communication Equipment, Data Terminal Equipment.**

Data Terminal Equipment (DTE): As defined by the RS-232-C standard, any device that generates or absorbs information, thus acting as an endpoint of a communication connection. A computer might serve as a DTE.

Data Terminal Ready (DTR): A signal from a DTE to a DCE indicating a readiness to transmit or receive data. See **Data Communication Equipment, Data Terminal Equipment.**

DCD: See **Data Carrier Detect.**

DCE: See **Data Communication Equipment.**

debug: A colloquial term that means to locate and correct an error or the cause of a problem or malfunction in a computer program. See also **bug.**

default: A preset response to a question or prompt. The default is automatically used by the computer if the user doesn't supply a different response. Default values prevent a program from stalling or crashing if no value is supplied by the user.

delete: To remove something, such as a character or word from a file, or a file from a disk.

Delete key: A key on the upper–right corner of the Apple IIe, Apple IIc, and Apple IIGS keyboards that erases the character immediately preceding (to the left of) the cursor. Similar to the Macintosh Backspace key.

delta guide: A description of something new in terms of its differences from something the reader already knows about. The name comes from the way mathematicians use the Greek letter delta (Δ) to represent a difference.

desk accessories: "Mini–applications" that are available from the computer's menu regardless of which application you're using—for example, the Control Panel, Calculator, Note Pad, Alarm Clock, Scrapbook, and so on.

desktop: The visual interface between the computer and the user—the menu bar and the gray area on the screen. You can have a number of documents on the desktop at the same time.

desktop environment: A set of program features that make user interactions with an application resemble operations on a desktop. Commands appear as options in pull–down **menus,** and material being worked on appears in areas of the screen called **windows.** The user selects commands or other material by using the **mouse** to move a pointer around on the screen.

desktop user interface: See **desktop environment.**

device driver: A program that manages the transfer of information between the computer and a peripheral device.

digit: (1) One of the characters 0 through 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary or 0 through 9 and A through F in hexadecimal.

digital: (adj) Represented in a discrete (noncontinuous) form, such as numerical digits or integers. For example, contemporary digital clocks show the time as a digital display (such as 2:57) instead of using the positions of a pair of hands on a clock face. Compare **analog.**

digital oscillator chip: an integrated circuit that contains thirty-two digital oscillators, each of which can generate a sound from stored digital waveform data.

digital signal: A signal that is sent and received in discrete intervals. A signal that does not vary continuously over time. Compare **analog signal.**

digital–to–analog converter: A device that converts quantities from digital to analog form.

DIN: Abbreviation for *Deutsche Industrie Normal,* a European standards organization.

DIN connector: A type of connector with multiple pins inside a round outer shield.

direct page: A page (256 bytes) of memory in the Apple IIGS that works like the **zero page** in a 6502 system but can reside anywhere in bank $00, rather than always starting at location $0000. Co–resident programs or routines can have their own direct pages at different locations.

directory: A file that contains a list of the names and locations of other files stored on a disk. These other files may themselves be directories (called *subdirectories*). A directory is sometimes called a *catalog*.

disassembler: A language translator that converts a machine–language program into an equivalent program in assembly language, which is easier for programmers to understand. The opposite of an **assembler.**

disk–based: See **disk–resident.**

disk controller card: A peripheral card that provides the connection between one or two disk drives and the computer. (This connection, or interface, is built into the Apple IIc, the Apple IIGS, and all Macintosh–family computers.)

disk operating system: An **operating system** whose principal function is to manage files and communications with one or more disk drives. **DOS** and **ProDOS** are two disk operating systems for the Apple II.

disk–resident: A program that does not remain in memory. The computer retrieves all or part of the program from the disk, as needed. Sometimes called *disk–based.* Compare **memory–resident.**

Disk II drive: An older type of disk drive made and sold by Apple Computer for use with the Apple II, II Plus, and IIe. It uses 5.25–inch floppy disks.

display: (1) A general term to describe what you see on the screen of your display device when you're using a computer. (2) Short for a display device.

display device: A device that displays information, such as a television set or video monitor.

dithering: A technique for alternating the values of adjacent pixels to create the effect of intermediate values. Dithering can give the effect of shades of gray on a black–and–white display, or more colors on a color display.

DOC: See **digital oscillator chip.**

DOS: See **Disk Operating System.**

DOS 3.3: An **operating system** for the Apple II family of computers. DOS stands for *Disk Operating System;* 3.3 is the version number.

DSR: See **Data Set Ready.**

DTE: See **Data Terminal Equipment.**

DTR: See **Data Terminal Ready.**

Dvorak keyboard: An alternate keyboard layout, also known as the *American Simplified Keyboard,* which increases typing speed because the keys most often used are in the positions easiest to reach. Compare **QWERTY keyboard.**

e flag: One of three flag bits in the 65C816 processor that programs use to control the processor's operating modes. The setting of the e flag determines whether the processor is in native mode or emulation mode. See **m flag, x flag.**

edit: To change or modify. For example, to insert, remove, replace, or move text in a document.

editor: A program that helps you create and edit information of a particular form; for example, a text editor or a graphics editor.

effective address: In machine–language programming, the address of the memory location on which a particular instruction operates, which may be arrived at by indexed addressing or some other addressing method.

8–bit Apple II: Another way of saying standard Apple II, that is, any Apple II with an 8–bit microprocessor (6502 or 65C02).

80–column text card: A peripheral card that allows the Apple II, Apple II Plus, and Apple IIe to display text in 80 columns (in addition to the standard 40 columns).

emulate: To operate in a way identical to a different system. For example, the 65C816 microprocessor in the Apple IIGS can carry out all the instructions in a program originally written for an Apple II that uses a 6502 microprocessor, thus emulating the 6502.

emulation mode: A manner of operating in which one system imitates another. In the Apple IIGS, the mode the 65C816 is in when the Apple IIGS is running programs written for Apple II's that use the 6502.

Escape character: An ASCII character that, with many programs and devices, allows you to perform special functions when used in combination keypresses.

Escape key: A key on Apple II–family computers that generates the Escape character. The Escape key is labeled *Esc*. In many applications, pressing Esc allows you to return to a previous **menu** or to stop a procedure.

even parity: In data transmission, the use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an even number; used as a means of error checking. Compare **MARK parity, odd parity.**

event–driven: A kind of program that responds to user inputs in real time by repeatedly testing for events posted by interrupt routines. An event–driven program does nothing until it detects an event such as a click of the mouse button.

expansion slot: A socket into which you can install a peripheral card. Sometimes called a *peripheral slot*. See also **auxiliary slot.**

Extended 80–Column Text Card: See **Apple IIe Extended 80–Column Text Card.**

file type: In a directory listing, the code that characterizes the contents of a file and indicates how the file may be used.

firmware: Programs stored permanently in read–only memory (ROM). Such programs (for example, the Applesoft Interpreter and the Monitor program) are built into the computer at the factory. They can be executed at any time but cannot be modified or erased from main memory.

font: In typography, a complete set of type in one size and style of character. In computer usage, a collection of letters, numbers, punctuation marks, and other typographical symbols with a consistent appearance.

format: (n) The form in which information is organized or presented. (v) To divide a disk into tracks and sectors where information can be stored. Blank disks must be formatted before you can save information on them for the first time; same as initialize.

frequency: The rate at which a repetitive event recurs. In alternating current (AC) signals, the number of cycles per second. Frequency is usually expressed in **hertz** (cycles per second), **kilohertz,** or **megahertz.**

function: A programmed sequence of operations that can be carried out on request from any point in a program. A function takes one or more arguments and returns a single value. It can therefore be embedded in an expression.

game I/O connector: A 16–pin connector inside all the open models of the Apple II, originally designed for connecting hand controls to the computer, but also used for connecting some other peripheral devices. Compare **hand control connector.**

GLU: Acronym for *general logic unit,* a class of custom integrated circuits used as interfaces between different parts of the computer.

graph: A pictorial representation of data.

graphics: (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text.**

hand controls: Peripheral devices, with rotating dials and push buttons. Hand controls are used to control game–playing programs, but they can also be used in other applications.

hand control connector: A 9–pin connector on the back panel of the Apple IIe , Apple IIc, and Apple IIGS computers, used for connecting hand controls to the computer. Compare **game I/O connector.**

handshaking: The exchange of status information between a **DCE** and a **DTE** used to control the transfer of data between them. The status information can be the state of a signal connecting the DCE and the DTE, or it can be in the form of a character transmitted with the rest of the data. See **Data Set Ready, Data Terminal Ready, Data Carrier Detect, XON, XOFF.**

hertz: The unit of frequency of vibration or oscillation, defined as the number of *cycles per second.* Named for the physicist Heinrich Hertz and abbreviated *Hz.* See **kilohertz, megahertz.**

hexadecimal: The base–16 system of numbers, using the ten digits 0 through 9 and the six letters A through F. Hexadecimal numbers can be converted easily and directly to binary form, because each hexadecimal digit corresponds to a sequence of four bits. Hexadecimal numbers are usually preceded by a dollar sign ($).

high–level language: A programming language that is relatively easy for people to understand. A single statement in a high–level language typically corresponds to several instructions of machine language. Compare **low–level language.**

high–order byte: The more significant half of a memory address or other multi–byte quantity. In the 6502 microprocessor used in the Apple II family of computers, the **low–order byte** of an address is usually stored first, and the high–order byte second. (In the 68000 microprocessors used in the Macintosh family, the high–order byte is stored first.)

Hi–Res: A high–resolution display mode on the Apple II family of computers, consisting of an array of points, 280 wide by 192 high, with 6 colors.

Hz: See **hertz.**

128K Apple II: Any standard Apple II with both main and auxiliary 64K banks of RAM. That includes all models of the Apple IIc and some models of the Apple IIe, including those with the Extended 80-Column Text Card installed. The Apple IIGS is not a 128K Apple II in the strict sense, even though it includes both 64K banks of RAM and is capable of running programs designed for a 128K Apple II.

IC: See **integrated circuit.**

icon: An image that graphically represents an object, a concept, or a message.

implement: To put into practical effect, as to *implement* a plan. For example, a language translator implements a particular language.

index register: A register in a computer processor that holds an index for use in indexed addressing. The 6502 and 65C816 microprocessors used in the Apple II family of computers have two index registers, called the **X register** and the **Y register.**

indexed addressing: A method used in machine–language programming to specify memory addresses. See also **memory location.**

input: (n) Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem.

input/output (I/O): The process by which information is transferred between the computer's memory and its keyboard or peripheral devices.

instruction: A unit of a machine–language or assembly–language program corresponding to a single action for the computer's processor to perform.

integrated circuit: An electronic circuit—including components and interconnections—entirely contained in a single piece of semiconducting material, usually silicon. Often referred to as an *IC* or a *chip.*

interactive: Operating by means of a dialog between the computer system and a human user.

interface: (1) The point at which independent systems or diverse groups interact. The devices, rules, or conventions by which one component of a system communicates with another. Also, the point of communication between a person and a computer. (2) The part of a program that defines constants, variables, data structures, and procedure–calling conventions, rather than procedures themselves.

interface card: A peripheral card that implements a particular interface (such as a parallel or serial interface) by which the computer can communicate with a peripheral device such as a printer or modem.

interrupt: A temporary suspension in the execution of a program that allows the computer to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

I/O: See **input/output.**

I/O device: Input/output device. A device that transfers information into or out of a computer.

I/O link: A fixed location that contains the address of an input/output subroutine in the computer's Monitor program.

IWM: Abbreviation for *Integrated Woz Machine,* the custom chip used in built–in disk ports on Apple computers.

joystick: A peripheral device with a lever, typically used to move creatures and objects in game programs; a joystick can also used in applications such as computer–aided design and graphics programs.

K: See **kilobyte.**

keyboard: The set of keys, similar to a typewriter keyboard, used for entering information into the computer.

kilobit: A unit of measurement, 1024 **bits,** commonly used in specifying the capacity of memory **ICs.** Not to be confused with **kilobyte.**

kilobyte (K): A unit of measurement of computer **data** or **memory,** consisting of 1024 (2^{10}) **bytes.** When used this way, *kilo* (from the Greek, meaning a thousand) stands for 1024. Thus, 64K memory equals 65,536 bytes. See also **megabyte.**

kilohertz: A unit of measurement of **frequency,** equal to 1000 **hertz** (abbreviated kHz). See also **megahertz.**

KSW: The symbolic name of the location in the computer's memory where the standard input link (namely, to the keyboard) is stored. *KSW* stands for *keyboard switch.*

language: See **programming language.**

language card: A peripheral card that, when installed in slot 0 of a 48K Apple II or Apple II Plus, gives the computer a total of 64K of memory. In Apple II's with 64K or more of memory, the part of memory equivalent to that occupied by a language card is sometimes called language–card memory.

line length: The number of characters that fit in a line on the screen or on a page.

load: To transfer information from a peripheral storage medium (such as a disk) into main memory for use—for example, to transfer a program into memory for execution.

loader: A program that brings files from a disk into the computer's memory.

location: See **memory location.**

logic board: See **main logic board.**

loop: A section of a program that is executed repeatedly until a limit or condition is met, such as an index variable's reaching a specified ending value. See **loop.**

low–level language: A programming language that is relatively close to the form the computer's processor can execute directly. One statement in a low–level language corresponds to a single machine–language instruction. Compare **high–level language.**

low–order byte: The least significant byte of a memory address or other multi–byte quantity. In the 6502 and 65C816 microprocessors used in the Apple II family of computers, the low–order byte of an address is usually stored first, and the **high–order byte** last. (In the 68000 microprocessors used in the Macintosh family, the high–order byte is stored first.)

Lo–Res: The lowest–resolution graphics mode on the Apple II family of computers, consisting of an array of blocks 48 high by 40 wide with 16 colors.

m flag: One of three flag bits in the 65C816 processor that programs use to control the processor's operating modes. In native mode, the setting of the m flag determines whether the accumulator is 8–bits wide or 16–bits wide. See **e flag, x flag.**

machine language: The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor (such as the 6502 microprocessor used in 8–bit Apple II computers) has its own form of machine language.

Macintosh: A family of Apple computers built around 68000 microprocessors, having high–resolution black–and–white displays and using **mouse** devices for choosing commands and for drawing pictures.

macro: A single predefined assembly–language pseudo–instruction that an assembler replaces with several actual instructions. Macros are almost like higher–level instructions that can be used inside assembly–language programs, making the programs easier to write.

macro assembler: A type of assembler that allows the programmer to define sequences of several assembly–language instructions as single pseudo–instructions called **macros.**

main logic board: A large circuit board that holds RAM, ROM, the microprocessor, custom–integrated circuits, and other components that make the computer a computer.

main memory: The part of a computer's memory whose contents are directly accessible to the microprocessor; usually synonymous with **random–access memory (RAM).**

MARK parity: A bit of value 1 appended to a binary number for transmission. The receiving device checks for errors by looking for this value on each character. Compare **even parity, odd parity.**

Mega II: A custom large–scale integrated circuit that incorporates most of the timing and control circuits of the standard Apple II. It addresses 128K of RAM organized as 64K main and auxiliary banks and provides the standard Apple II video display modes, both text (40–column and 80–column) and graphics (Lo–Res, Hi–Res, and Double Hi–Res).

megabit: A unit of measurement, 1,048,576 (2^{16}) bits or 1024 **kilobits,** commonly used in specifying the capacity of memory **ICs.** Not to be confused with **megabyte.**

megabyte: A unit of measurement of computer **data** or **memory,** equal to 1,048,576 **bytes** or 1024 **kilobytes;** abbreviated Mb.

megahertz: A unit of measurement of **frequency,** equal to 1,000,000 **hertz** (abbreviated MHz). See also **kilohertz.**

memory: The hardware component of a computer system that stores information for later retrieval. See also **main memory, random–access memory, read–only memory, read–write memory.**

memory location: A unit of main memory that is identified by an address and can hold a single item of information of a fixed size. In the Apple II family of computers, a memory location holds one byte.

Memory Manager: One of the programs in the Toolbox. Its job is to allocate memory so that applications and desk accessories can run without clobbering each other.

memory–mapped I/O: The method used for I/O operations in Apple II computers, where certain memory locations are attached to I/O devices, and I/O operations are just memory load and store instructions.

memory–resident: (adj) (1) Stored permanently in memory as firmware (ROM). (2) Held continually in RAM even while not in use. DOS is a memory–resident program. Compare **disk–resident.**

menu: A list of choices presented by a program, from which you can select an action. Menus appear when you point to and press menu titles in the **menu bar.** Dragging through the menu and releasing the mouse button while a command is highlighted chooses that command.

menu bar: The horizontal strip at the top of the screen that contains menu titles.

menu title: A word, phrase, or icon in the menu bar that designates one menu. Pressing on the menu title causes the title to be highlighted and its menu to appear below it.

MHz: Abbreviation for megahertz, one million hertz. See **hertz.**

microprocessor: A computer **processor** contained in a single integrated circuit. The microprocessor is the **central processing unit (CPU)** of the microcomputer. Examples include the 6502 and 65C816 microprocessors used in the Apple II family of computers and the 68000 microprocessor used in the Macintosh family.

microsecond: One millionth of a second. Abbreviated *μs*.

millisecond: One thousandth of a second. Abbreviated *ms*.

mode: A state of a computer or system that determines its behavior. A manner of operating.

modem: Short for *MOdulator/DEModulator*. A peripheral device that links a computer to other computers and information services using the telephone lines.

monitor: See **video monitor.**

Monitor program: A system program built into the firmware of Apple II computers, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine–language level.

MOS: Abbreviation for *metal oxide silicon,* a method of semiconductor integrated–circuit fabrication on silicon using layers of silicon dioxide in the make–up of the devices. Compare **CMOS.**

mouse: A small device you move around on a flat surface next to your computer. The mouse controls a pointer on the screen whose movements correspond to those of the mouse. You use the pointer to select operations, to move data, and to draw with in graphics programs.

mouse button: The button on the top of the mouse. In general, pressing the mouse button initiates some action on whatever is under the pointer, and releasing the button confirms the action.

NTSC: (1) Abbreviation for *National Television Standards Committee*. The committee that defined the standard format used for transmitting broadcast video signals in the United States. (2) The standard video format defined by the NTSC, also called **composite,** because it combines all the video information, including color, into a single signal.

object code: See **object program.**

object program: The translated form of a program produced by a language translator such as a compiler or assembler. Also called *object code*. Compare **source program.**

odd parity: In data transmission, the use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an odd number; used as a means of error checking. Compare **even parity, MARK parity.**

128K Apple II: Any standard Apple II with both main and auxiliary 64K banks of RAM. That includes all models of the Apple IIc and some models of the Apple IIe, including those with the Extended 80-Column Text Card installed. The Apple IIGS is not a 128K Apple II in the strict sense, even though it includes both 64K banks of RAM and is capable of running programs designed for a 128K Apple II.

opcode: See **operation code.**

Open Apple: A modifier key on some Apple II–family keyboards; on the Apple IIGS keyboard, the equivalent key is marked with both an Apple icon and a spinner, the icon used on some Macintosh keyboards, and called simply the *Apple key*.

operation code: The machine-language representation of a computer instruction.

system program: A program that makes the resources and capabilities of the computer available for general purposes, such as an **operating system** or a language translator. Compare **application program.**

operating system: A general–purpose program that manages the actions of the parts of the computer and its peripheral devices for the benefit of the application programs. See **Disk Operating System.**

overrun: A condition that occurs when the processor does not retrieve a received character from the receive data register of a communications interface device before the subsequent character arrives to occupy that register.

page: (1) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256. (2) An area of main memory containing text or graphical information being displayed on the screen.

page zero: See **zero page.**

parallel interface: An **interface** in which several bits of information (typically 8 bits, or 1 byte) are transmitted simultaneously over different wires or channels. Compare **serial interface.**

parameter: An argument that determines the outcome of a command. For example, in the command *write(n,msg)*, *n* and *msg* are parameters.

parity: Sameness of level or count, usually the count of 1 bits in each character, used for error checking in data transmission. See **even parity, MARK parity, odd parity, parity bit.**

parity bit: A bit used to check for errors during data transmission. Depending on the number of 1 bits in a transmission, the parity bit is set to 1 or 0 to make the total number of 1 bits even or odd.

Pascal: A high–level programming language with statements that resemble English phrases. Pascal was designed to teach programming as a systematic approach to problem solving. Named after the philosopher and mathematician Blaise Pascal.

peripheral: (adj) At or outside the boundaries of the computer itself, either physically (as a peripheral device) or in a logical sense (as a peripheral card). (n) Short for *peripheral device.*

peripheral card: A removable printed–circuit board that plugs into one of the computer's expansion slots. Peripheral cards enable the computer to use peripheral devices or to perform other subsidiary or peripheral functions.

peripheral device: A piece of hardware—such as a video monitor, disk drive, printer, or modem—used in conjunction with a computer and under the computer's control. Peripheral devices are often (but not necessarily) physically separate from the computer and connected to it by wires, cables, or some other form of interface. They often require **peripheral cards.**

peripheral slot: See **expansion slot.**

phase: (1) A stage in a periodic process. A point in a cycle. For example, the 65C816 microprocessor uses a clock cycle consisting of two phases called Φ0 and Φ1. (2) The relationship between two periodic signals or processes.

pixel: Short for *picture element.* The smallest dot you can draw on the screen. Also, a location in video memory that corresponds to a point on the graphics screen when the viewing window includes that location. In the Macintosh display, each pixel can be either black or white, so it can be represented by a bit; thus, the display is said to be a **bitmap.** In the Super Hi-Res display on the Apple IIGS, each pixel is represented by either two or four bits; the display is not a **bitmap,** but rather a **pixelmap.**

pixelmap: A set of values that represents the positions and states of the set of **pixels** making up an image. Compare **bitmap.**

pop: To retrieve an entry from the top of a **stack,** moving the stack pointer to point to the previous entry. Compare **push.**

port: A socket on the back panel of the computer where you can plug in a cable to connect a peripheral device, another computer, or a network.

PR#: An Applesoft BASIC command that directs output to a slot or a machine–language program. It activates an output routine in the ROM on a peripheral card or in equivalent RAM by changing the address of the standard output routine used by the computer.

procedure: In the Pascal and Logo programming languages, a sequence of instructions that work as a unit; approximately equivalent to the term **function** in C or **subroutine** in BASIC.

processor: The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory. See **microprocessor.**

ProDOS: A **disk operating system** for the Apple II family of computers. ProDOS stands for *Professional Disk Operating System,* and includes ProDOS 8 and ProDOS 16.

ProDOS 8: A **disk operating system** for the Apple II. It runs on 6502 and 65C02 microprocessors and on the 65C816 in 6502 emulation mode.

ProDOS 16: The **disk operating system** designed for the Apple IIGS. ProDOS 16 is similar to ProDOS 8, but was designed to run on the 65C816 microprocessor in the Apple IIGS.

program: (n) A sequence of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language. (v) To write a program.

programmable read-only memory: A type of **ROM** device that is programmed after fabrication, unlike ordinary ROM devices, which are programmed during fabrication.

programming language: A set of symbols and associated rules or conventions for writing programs. BASIC, Logo, and Pascal are programming languages.

PROM: See **Programmable Read-Only Memory.**

prompt: A message on the screen that tells you of some need for response or action. A prompt usually takes the form of a symbol, a message, a dialog box, or a menu of choices.

prompt character: A text character displayed on the screen, usually just to the left of a **cursor,** where your next action is expected. The prompt character often identifies the program or component of the system that's prompting you. For example, Applesoft BASIC uses a square bracket prompt character (]); the system Monitor program, an asterisk (*); and the Mini–assembler, an exclamation point (!).

protocol: A formal set of rules for the interchange of information between two programs or devices; for example, the rules for sending and receiving data on a communication line.

Protocol Converter: A set of ROM–based assembly–language routines used to support external I/O devices such as the Apple Memory Expansion Card and the Apple 3.5 Drive.

push: To add an entry to the top of a **stack,** moving the stack pointer to point to it. Compare **pop.**

queue: A list in which entries are added at one end and removed at the other, causing entries to be removed in first–in, first–out (FIFO) order. Compare **stack.**

QWERTY keyboard: The standard layout of keys on a typewriter keyboard; its name is formed from the first six letters on the top row of letter keys. Compare **Dvorak keyboard.**

RGB monitor: A type of color monitor that receives separate signals for each color (red, green, and blue). See **composite video.**

RAM: See **random-access memory.**

random-access memory (RAM): Memory in which information can be referred to in an arbitrary or random order. As commonly used, RAM means the part of memory available for programs from a disk; the programs and other data are lost when the computer is turned off. (Technically, the read-only memory (ROM) is also *random access,* and what's called RAM should correctly be termed *read-write memory.*) Compare **read-only memory, read-write memory.**

read-only memory (ROM): Memory whose contents can be read, but not changed; used for storing **firmware.** Information is placed into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off. Compare **random-access memory, read-write memory, write-only memory.**

read-write memory: Memory whose contents can be both read and changed (or written to); commonly called **RAM.** The information contained in read-write memory is erased when the computer's power is turned off and is permanently lost unless it has been saved on a disk or other storage device. Compare **random-access memory, read-only memory.**

reentrant: Characteristic of a program routine that is able to accept a call while one or more previous calls to it are pending without invalidating any previous calls.

register: A location in a processor or other device where an item of information is held and modified under program control.

Resource Manager: A Macintosh tool for editing data in program segments without recompiling them.

resident: See **memory-resident, disk-resident.**

return address: The point in a program to which control returns on completion of a subroutine or function.

RGB: Abbreviation for *red-green-blue,* a method of displaying color video by transmitting the three primary colors as three separate signals. There are two ways of using RGB with computers: **TTL RGB,** which allows the color signals to take on only a few discrete values; and **analog RGB,** which allows the color signals to take on any values between their upper and lower limits, for a wide range of colors.

ROM: See **read-only memory.**

routine: A part of a **program** that accomplishes some task subordinate to the overall task of the program.

row: A horizontal line of character cells or graphics **pixels** on the screen.

RS–232: A common standard for serial data–communication interfaces.

RS-422: A standard for serial data–communication interfaces, different from the RS-232 standard in its electrical characteristics and in its use of differential pairs for data signals. The serial ports on the Apple IIGS use RS-422 devices modified so as to be compatible with RS-232 devices.

SANE: See **Standard Apple Numeric Environment.**

SCC: Acronym for *Serial Communications Controller,* a type of communications IC used in the Apple IIGS computer. See **ACIA.**

screen holes: Locations in the text display buffer (text Page 1) used for temporary storage either by I/O routines running in peripheral–card ROM or by firmware routines addressed as if they were in card ROM. Text Page 1 occupies memory from $400 to $7FF; the screen holes are locations in that area that are neither displayed nor modified by the display firmware.

serial interface: An interface in which information is transmitted sequentially, a bit at a time, over a single wire or channel. Compare **parallel interface.**

serial port: The connector for a peripheral device that uses a **serial interface.**

silicon: A solid, crystalline chemical element (symbol Si) from which integrated circuits are made. Silicon is a *semiconductor;* that is, it conducts electricity better than insulators, but not as well as metallic conductors. Silicon should not be confused with silica—that is, silicon dioxide, such as quartz, opal, or sand—or with silicone, any of a group of organic compounds containing silicon.

Simplified Keyboard: See **Dvorak keyboard.**

64K Apple II: Any standard Apple II that has at least 64K of RAM. That includes the Apple IIc, the Apple IIe, and an Apple II or Apple II Plus with 48K of RAM and the Language Card installed.

6502: The microprocessor used in the Apple II, in the Apple II Plus, and in early models of the Apple IIe. The 6502 is an **MOS** device with 8–bit data registers and 16–bit address registers.

65C02: A **CMOS** version of the 6502; the microprocessor used in the Apple IIc and in the enhanced Apple IIe.

65C816: The microprocessor used in the Apple IIGS. The 65C816 is a **CMOS** device with 16–bit data registers and 24–bit address registers.

68000: The microprocessor used in the Macintosh and Macintosh Plus. The 68000 has 32–bit data and address registers.

slot: A narrow socket inside the computer where you can install peripheral cards. Also called an **expansion slot.**

soft switch: A location in memory that produces some specific effect whenever its contents are read or written.

software: A collective term for **programs,** the instructions that tell the computer what to do. They're usually stored on disks. Compare **firmware.**

source code: See **source program.**

source program: The form of a program given to a language translator, such as a compiler or assembler, for conversion into another form; sometimes called *source code.* Compare **object program.**

stack: A list in which entries are added (pushed) or removed (popped) at one end only (the top of the stack), causing them to be removed in last–in, first–out (LIFO) order. Compare **queue.**

Standard Apple II: Any computer in the Apple II family except the Apple IIGS. That includes the Apple II, the Apple II Plus, the Apple IIe, and the Apple IIc.

Standard Apple Numerics Environment: Apple's implementation of IEEE standard floating–point arithmetic, used on the Apple II and Macintosh families of computers.

start bit: One or two bits that indicate the beginning of a character in a string of serially transmitted characters.

start up: To get the system running. Starting up is the process of first reading the **operating system** program from the disk, and then running an application program. See **boot.**

startup disk: A disk with all the necessary program files to set the computer into operation. Sometimes called a *boot disk.*

stop bit: A bit indicating the end of a character in a string of serially transmitted characters.

strobe: A signal whose change is used to trigger some action.

subdirectory: A directory within a directory. A file containing the names and locations of other files.

subroutine: A part of a program that can be executed on request from another point in the program and that, on completion, returns control to that point.

system: A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose—for example, a computer system comprising a processor, keyboard, monitor, and disk drive.

system configuration: See **configuration.**

system program: A program that makes the resources and capabilities of the computer available for general purposes, such as an **operating system** or a language translator. Compare **application program.**

system software: The component of a computer system that supports application programs by managing system resources such as memory and I/O devices.

text: (1) Information presented in the form of readable characters. (2) The display of characters on a display screen. Compare **graphics.**

text window: A **window** on the desktop within which text is displayed and scrolled.

toolbox: A collection of built-in routines that programs can call to perform many commonly–needed functions.

transistor–transistor logic (TTL): (1) A family of integrated circuits having bipolar circuit logic; TTL ICs are used in computers and related devices. (2) A standard for interconnecting such circuits, which defines the voltages used to represent logical zeros and ones.

TTL: See **transistor–transistor logic.**

TTL RGB: A type of video monitor that can accept only a limited number of digital values and display only a correspondingly limited number of colors. Compare **analog RGB.**

type–ahead buffer: A **buffer** that accepts and holds characters that are typed faster than the computer can process them.

user: A person operating or controlling a computer system.

user interface: The rules and conventions by which a computer system communicates with the person operating it.

utilities: Programs that let you rename, copy, format, delete, and otherwise manipulate files and volumes.

value: An item of information that can be stored in a variable, such as a number or a string.

variable: (1) A location in the computer's memory where a value can be stored. (2) The symbol used in a program to represent such a location.

VBL: Short for *vertical blanking,* an interrupt signal generated by the video timing circuit each time it finishes a vertical scan, 60 times a second.

vector: (1) The starting address of a program segment when used as a common point for transferring control from other programs. (2) A memory location used to hold a vector, or the address of such a location.

video: (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode–ray tube. (2) Information organized or transmitted in video form.

video monitor: A display device that can receive video signals by direct connection only, and that cannot receive broadcast signals such as commercial television. Can be connected directly to the computer as a display device.

warm start: The process of transferring control back to the **operating system** in response to a failure in an application program. Compare **cold start.**

window: The area that displays information on a desktop. You view a document through a window. You can open or close a window, move it around on the desktop, and sometimes change its size, scroll through it, and edit its contents.

word: A group of bits that is treated as a unit. The number of bits in a word is a characteristic of each particular computer; in the Apple IIGS, words are sixteen bits wide.

wraparound: The automatic continuation of text from the end of one line to the beginning of the next; wraparound means that you don't have to press the Return key at the end of each line as you type.

write-only memory: A form of computer memory into which information can be stored but never, ever retrieved. For more information, refer to *The Life of Homberg T. Farnsfarfle,* by Bruce Tognazzini.

x flag: One of three flag bits in the 65C816 processor that programs use to control the processor's operating modes. In native mode, the setting of the x flag determines whether the index registers are 8-bits wide or 16-bits wide. See **e flag, m flag.**

XON: A special character (**ASCII** value $13) used for controlling the transfer of data between a **DTE** and a **DCE.** See **handshaking, XOFF.**

XOFF: A special character (**ASCII** value $11) used for controlling the transfer of data between a **DTE** and a **DCE.** When one piece of equipment receives an XOFF character from the other, it stops transmitting characters until it receives an XON. See **handshaking, XON.**

X register: One of the two index registers in the 6502 microprocessor.

Y register: One of the two index registers in the 6502 microprocessor.

zero page: The first page (256 bytes) of memory in the Apple II family of computers, also called *page zero*. Since the high–order byte of any address in this page is zero, only the low–order byte is needed to specify a zero–page address; this makes zero–page locations more efficient to address, in both time and space, than locations in any other page of memory. The 65C816 microprocessor used in the Apple IIGS has a relocatable zero page called the **direct page.**