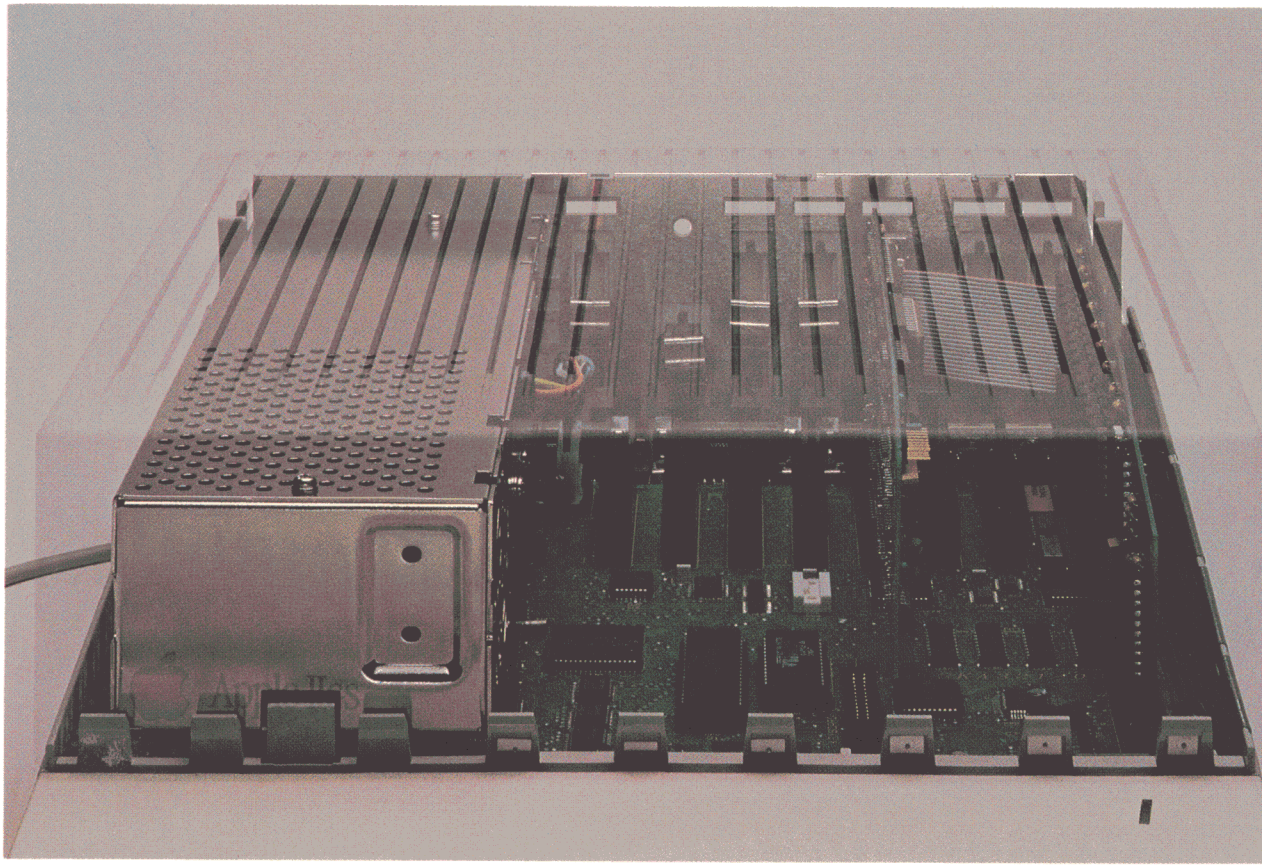
For Apple IIGS and 1MB Apple IIGS

Apple IIGS® Hardware Reference

by Apple Computer, Inc.



Second Edition

For Apple IIGS® and 1 MB Apple IIGS

Apple IIGS® Hardware Reference

Second Edition

Addison-Wesley Publishing Company, Inc.
Reading, Massachusetts Menlo Park, California New York
Don Mills, Ontario Wokingham, England Amsterdam Bonn Sydney
Singapore Tokyo Madrid San Juan

APPLE COMPUTER, INC.

Simultaneously published in the United States and Canada.

ISBN 0–201–52389–2
ABCDEFGHIJ–MU–89
First Printing, September 1989

WARRANTY INFORMATION

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

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

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

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

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

Contents

Figures and tables

Chapter 5 Apple IIGS Sound / 101

Chapter 6 The Apple Desktop Bus / 121

Preface

This is the hardware reference manual for the Apple IIGS® computer, including the original 256K Apple IIGS and the 1 MB Apple IIGS computers. It is primarily for hardware designers and programmers who want to know how to manipulate the hardware, but will also be useful to anyone wanting to know how to take advantage of all the features of these computers.

About this manual

As part of the Apple IIGS technical suite of manuals, the *Apple IIGS Hardware Reference* covers the design and function of the Apple IIGS hardware. It provides hardware design and interface information and is intended for people who have an understanding of digital microprocessor electronics and who are interested in interfacing the Apple IIGS to the outside world. Programmers who are using the sound and graphics capabilities of the Apple IIGS should also refer to this manual so that they can have a good understanding of how software affects the hardware.

The Apple IIGS is, above all, an Apple II. This manual introduces the computer's features and also describes how the Apple IIGS maintains compatibility with previous Apple II models.

Specifically, this manual

- introduces the features of the Apple IIGS hardware
- describes the Apple II–compatible functions
- provides addresses where necessary for manipulating the hardware
- provides input/output (I/O) information for interfacing signals to the outside world

Unlike previous Apple technical reference manuals, this one does not document firmware features; they are covered in detail in the *Apple IIGS Firmware Reference*. The Apple IIGS uses a set of low-level tool utilities to perform certain functions. Programmers are expected to use these tools to perform sound and graphics operations whenever available. Read the *Apple IIGS Toolbox Reference* for more information.

What this manual contains

Each chapter in this manual describes a different aspect of the computer. Each chapter is modular in scope, describing the particular aspect of the Apple IIGS and making references to other chapters and manuals where necessary.

Chapter 1 introduces the features of the Apple IIGS.

Chapter 2, "The Core of the Apple IIGS," describes the heart the Apple IIGS, the Mega II, and the Fast Processor Interface (FPI) custom integrated circuits.

Chapter 3, "Memory," describes the organization of the built-in memory as well as the memory expansion slot, and tells how to design and access a memory expansion card for this special slot.

Chapter 4, "The Video Displays," goes in depth into how the standard Apple II graphics modes work, and how to make the Super Hi-Res graphics work for you.

Chapter 5, "Apple IIGS Sound," shows you how to use the 32 digital oscillators to generate sound.

Chapter 6, "The Apple Desktop Bus," provides details of the hardware and protocol required to design and connect an input device (keyboard, mouse, graphics tablet, and so on) to this input bus.

Chapter 7, "Built-in I/O Ports and Clock," describes the disk port, the serial ports, the game port, and the real-time clock.

Chapter 8, "I/O Expansion Slots," lists the I/O signals available at the expansion slots, and gives loading precautions and programming suggestions for the slot cards. DMA and interrupts are described here also.

Chapter 9 briefly describes the Apple IIGS power supply and lists its specifications.

Chapter 10 covers the 65C816 microprocessor.

Appendix A contains a roadmap to the Apple IIGS technical suite of manuals. Read this appendix to determine which books will help you to learn more about a programming language, the Apple IIGS firmware, or other aspect of the computer.

Appendix B has eight international keyboard layouts.

Appendix C shows you the contents of the character generator—all the characters the Apple IIGS can display.

Appendix D has tables that show what a bit and a byte can represent. Conversion tables between hexadecimal, decimal, and negative decimal, as well as 7-bit ASCII, are provided.

Appendix E contains some of the most frequently used tables taken from throughout the manual.

A glossary follows the appendixes.

An addendum, after the index, contains schematic diagrams showing all of the electrical components of the main circuit board.

Conventions and usage in this manual

The Apple II and Apple II Plus are *standard* Apple II computers. In this manual, reference is made to the compatibility of the Apple IIGS with standard Apple II computers. This means that the Apple IIGS will run software written for an Apple II or Apple II Plus computer. A particular function that the Apple IIGS has in common with the Apple IIe or Apple IIc, for instance, will be mentioned specifically as such.

A revised Apple IIGS, the 1 MB Apple IIGS, was introduced into the Apple II family in mid-1989. This new main logic board differs from the original main logic board in several ways, but primarily in the amount of RAM supplied on the main logic board. The *original* Apple IIGS contains 256K of RAM, and the *1 MB* Apple IIGS contains 1 megabyte of RAM. Other differences between the computers are detailed in the pertinent sections of each chapter. Information throughout this manual pertains to both versions of the main logic board unless otherwise indicated.

Special messages

Some text in this manual is set off from the rest in special ways:

- *Note:* A note usually contains information that is interesting but not necessary for an understanding of the main text.

△ **Important** Text set off like this presents especially important information. △

▲ **Warning** The warning message contains information that, if ignored, could result in loss of data, damage to equipment, or possibly bodily injury. ▲

Numbers preceded by a dollar sign are **hexadecimal** rather than **decimal** representations of values. This convention is used throughout this manual.

Words that appear in **boldface** in the text are defined in the glossary located at the back of this manual.

Memory sizes

Throughout this manual, memory is given in kilobytes, abbreviated by the letter *K;* or in megabytes, abbreviated by the letters *MB.* A kilobyte equals 1024 (2^{10}) bytes, and a megabyte equals 1024K, or 1024 x 1024 bytes. Therefore, memory sizes in kilobytes and megabytes are not in even thousands and millions. For example, 5 MB is 1024 x 1024 x 5 = 5,242,880 bytes, not 5,000,000 bytes.

Altering register contents

When programming the Apple IIGS, you will need to manipulate certain bits within registers and soft switch locations in order to achieve a particular result. Some bits in these registers or soft switches must be left alone, or the system could crash. These bits are labeled *Reserved; do not modify.*

▲ **Warning** In order to manipulate the desired bits in these registers and leave those reserved ones untouched, you must use a read-modify-write technique. Either of two assembly-language commands can be used to accomplish this: the test-and-set-bit (TSB) command or the test-and-reset-bit (TRB) command. Both of these commands allow you to modify any one bit and leave the others untouched. ▲

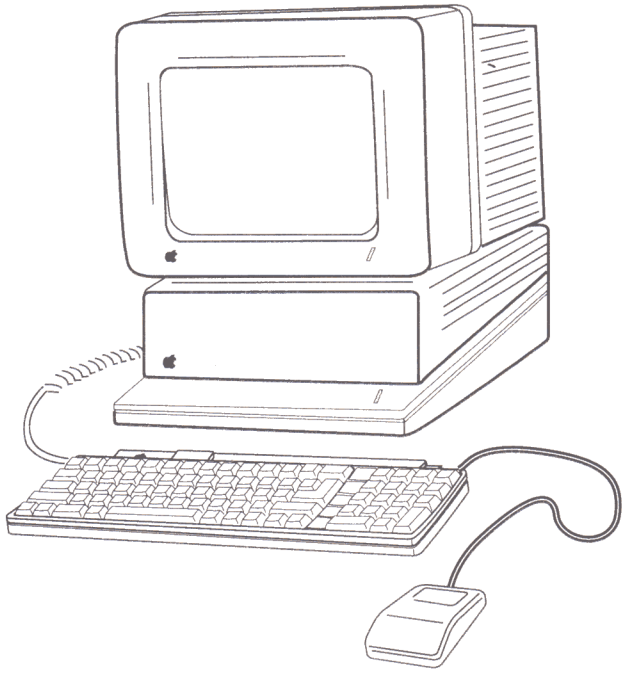
To read about using the TSB and TRB instructions, refer to *65816/65802 Assembly Language Programming* by Michael Fischer, published by Osborne/McGraw-Hill.

Chapter 1 **Introduction to the Apple IIGS**

The Apple IIGS® is a new computer in the Apple® II family. While maintaining its roots in the Apple IIe and Apple IIc, this computer also provides new features that make it the most powerful Apple II yet. This first chapter describes generally how the Apple IIGS fits into the Apple II family and tells what sets it apart from previous Apple II computers.

Figure 1-1 shows the Apple IIGS.

- **Figure 1-1** The Apple IIGS



Apple II compatibility

The Apple IIGS is compatible with the Apple II family of computers. Some of the features the Apple IIGS shares with the Apple IIe and IIc are

- **6502** processor compatibility, which is maintained by the **65C816** microprocessor used by the Apple IIGS
- standard Apple II **graphics**, which include **Lo-Res** mode, **Hi-Res** mode, and **Double Hi-Res** mode color video graphics
- 128K of main **RAM**
- built-in Applesoft BASIC
- two built-in **serial ports**

- seven **peripheral expansion slots**, compatible with the Apple IIe
- a built-in disk interface port like the Apple IIc that will accept either 5.25-inch or 3.5-inch **disk drives**
- built-in Apple II **Monitor program**
- 40-column and 80-column text display capability
- a game **I/O** port for **joysticks** and **hand controls** like the Apple IIe and IIc

Apple IIGS features

Although the Apple IIGS has many features in common with previous Apple II products, it has several new features that enhance its performance. Here are a few examples:

- The 16-bit **CMOS** 65C816 microprocessor, which uses a superset of the 6502 instruction set, includes 11 new **address** modes and 36 new instructions, and is compatible with 6502 code. To learn more about the 6502 and the **65C02** microprocessors, refer to the *Apple IIe Technical Reference* and the *Apple IIc Technical Reference,* respectively.
- High processing speed, which is selectable between 1.024 MHz or 2.8 MHz.
- Super Hi-Res video graphics modes, which offer either 320- or 640-**pixel** horizontal resolution, displaying 16 colors per line; these colors may be chosen from a possible 4096.
- **Analog RGB** color video outputs.
- 256K of RAM built onto the main logic board. You can increase this RAM to over 4 MB by using an optional expansion card in the memory expansion slot; 896K of **ROM** also can be added by using a memory expansion card.
- A **Control Panel** screen, which provides users with means for setting system parameters.
- Built-in AppleTalk® network firmware.
- Built-in **real-time clock (RTC)** with a backup battery, which is accessible through the Control Panel.
- Selectable display border, text, and background colors.
- A sound synthesizer **integrated circuit (IC)** with 32 independent **oscillators** and 64K of dedicated RAM.
- A detachable, international keyboard with keypad.
- Apple Desktop Bus™, whose **protocol** provides for input **devices** such as graphics pads, **mouse** devices, and keyboards.
- Enhanced Monitor firmware, which supports the 65C816 microprocessor.

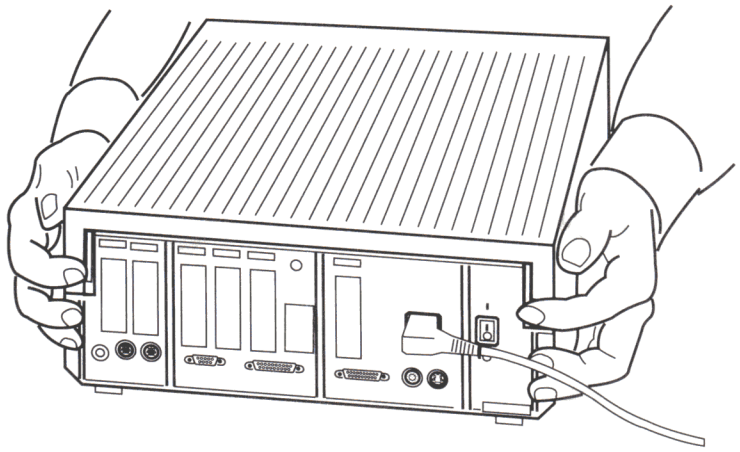
1 MB Apple IIGS additional features

In mid-1989, Apple introduced the 1 MB main logic board for the Apple IIGS. This logic board incorporates several features not found on the original logic board. These new functions of the 1 MB Apple IIGS computer are as follows:

- The main logic board RAM has been expanded to 1 MB. A memory expansion card containing up to 4 MB may be added to expand usable memory.
- The main logic board ROM has been expanded to 256K. The ROM now includes all of the toolbox utilities, previously resident on disk. (Refer to the *Apple IIGS Toolbox Reference* for information on the Apple IIGS Toolbox utilities.) With a ROM expansion card, an additional 768K of ROM can be addressed.
- A new version of the Apple Desktop Bus (ADB) microcontroller IC provides two new features:
  - Sticky keys
  - Keyboard mouse

  Also, the new board revision no longer supports the built-in (Apple IIe) keyboard; only the ADB keyboards are supported by the 1 MB Apple IIGS. The Apple Desktop Bus is covered in Chapter 6.
- Shadowing of text Page 2 memory is available for the 1 MB Apple IIGS. A **bit** in the Shadow register ($C035) controls whether or not this address space in banks $00 and $01 is shadowed into banks $E0 and $E1. Chapter 3 describes memory shadowing in the Apple IIGS.
- The new Apple IIGS logic board provides a reliable means for programs to determine under what circumstances a cold boot sequence was initiated: a bit in the Speed register is set when the power switch is turned on. The Speed register is covered in Chapter 2.

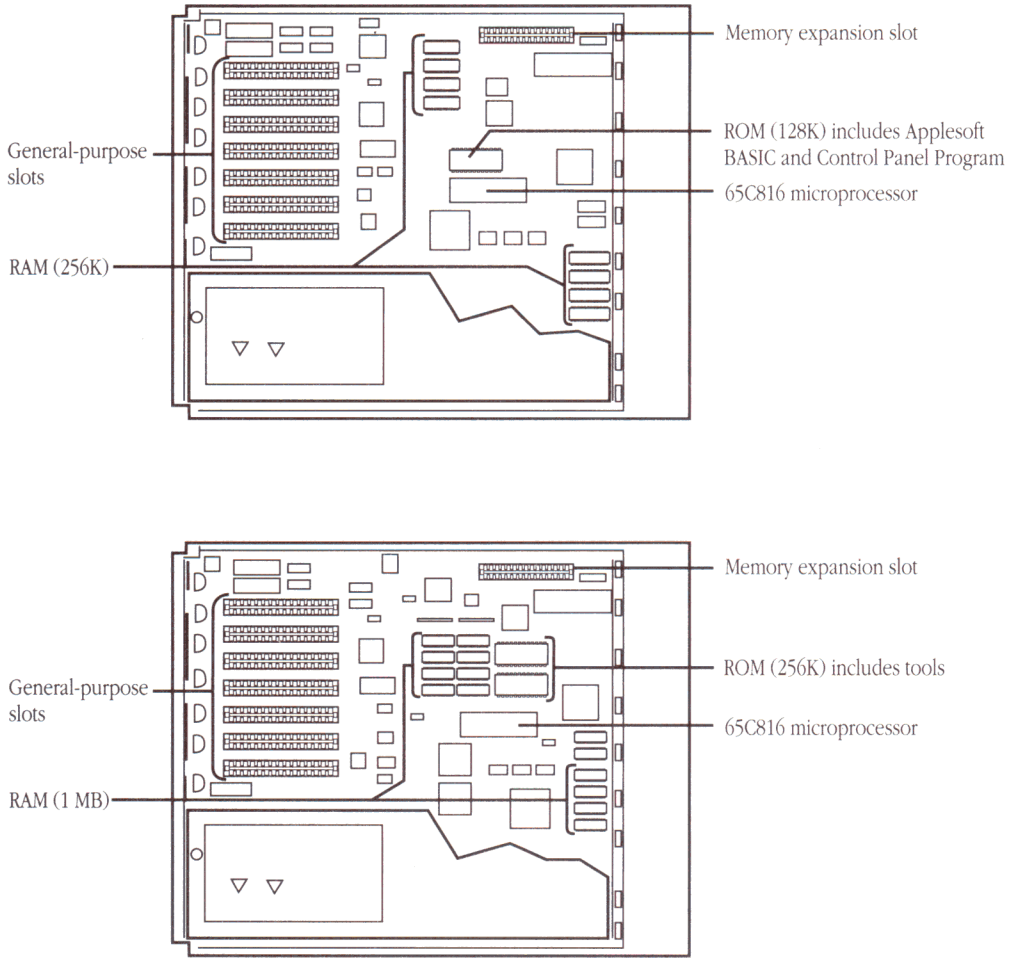
- **Figure 1-2** Releasing the snap locks to remove the cover



Removing the cover

The Apple IIGS has a molded two-piece case. The cover is hinged at the front and is secured at the rear where the upper and lower halves meet. A snap lock is located at each side of the rear panel, as shown in Figure 1-2. To remove the cover, press in on each snap lock while lifting up at the rear of the cover. Pivot the cover at the front and remove it completely. The **main logic board** is now exposed for access to the expansion slots. Figure 1-3 shows the major components of the Apple IIGS.

■ **Figure 1-3** The Apple IIGS with the cover removed. The 256K machine is shown at top and the 1 MB machine at bottom.



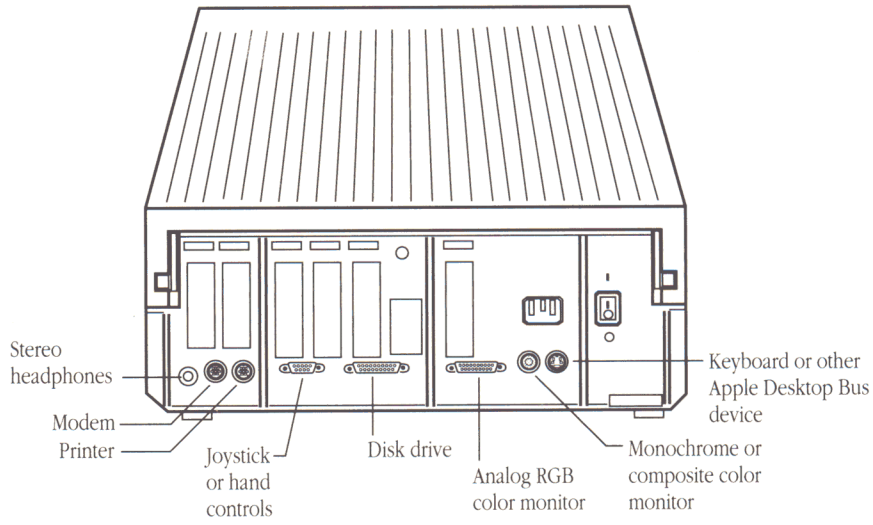
Peripheral expansion slots

The Apple IIGS, like the Apple IIe, has seven expansion slots at the rear of the main logic board. These slots will accept most Apple II–compatible **peripheral cards** designed for any of the Apple II computers. Note that the Apple IIGS does not have an **auxiliary slot** as is found in the Apple IIe. For more information on the peripheral expansion slots, see Chapter 8, "I/O Expansion Slots."

Connectors

At the rear of the computer are several connectors. These connectors allow the computer to be connected to an input device such as a keyboard or a mouse, or a **peripheral device** such as a disk drive, a printer, a **modem,** a network, or the like. Figure 1-4 shows the connectors.

■ **Figure 1-4** Apple IIGS connectors



Serial ports

The two **RS-232-C** and **RS-422** compatible serial ports use mini-**DIN** (Deutsche Industrie Normal) 8-pin connectors. To transmit and receive **data** to and from a device connected to a serial port, use the firmware calls in the system **read-only memory (ROM).** The serial ports are described in Chapter 7, "Built-in I/O Ports and Clock." (To read about how to use the firmware in the Apple IIGS ROM, refer to the *Apple IIGS Firmware Reference.*)

Disk drive port

This connector will accept either 5.25-inch or 3.5-inch Apple disk drives made for the Apple II. This 19-pin connector is similar in function to the one on the Apple IIc. For more information on the disk drive port, see Chapter 7, "Built-in I/O Ports and Clock."

RGB video connector

This connector provides analog red, green, and blue **(RGB)** video signals for an analog-input RGB **video monitor.** Use only an analog-input RGB monitor with this 15-pin connector. See Chapter 4, "The Video Displays," for more information.

Composite video connector

Composite video is available at this connector. A standard Apple composite color monitor can be used to display video. A television may be used to display 40-column text or graphics: This requires a video modulator to connect the Apple IIGS to a television. See Chapter 4, "The Video Displays," for a description of composite video.

Apple Desktop Bus

Connect **Apple Desktop Bus (ADB)** devices to this connector. These devices may be ADB keyboards, ADB mouse devices, ADB graphics tablets, or any other input device designed to the ADB specification. Do not attempt to adapt input devices not designed for ADB to this connector. See Chapter 6, "The Apple Desktop Bus," for more information on using this connector.

Game connector

Connect a **standard Apple II** hand control or joystick to this connector. Do not adapt an ADB device to this connector. ADB devices are completely different, and should not be used. See Chapter 7, "Built-in I/O Ports and Clock," for more information on game connectors and signals.

A closer look

You can think of the Apple IIGS system as containing two separate and unique subsystems. These subsystems are not mutually exclusive; on the contrary, the subsystems share several components without which they could not function. In particular, both share the **microprocessor, input/output (I/O), memory, video,** and expansion support circuitry.

The Mega II subsystem

The first subsystem, referred to as the **Mega II** portion of the system because of the controlling device, consists of the parts of the computer that make the Apple IIGS compatible with other Apple II products. These are

- the 65C816 microprocessor
- the Mega II custom integrated circuit (IC)
- 128K of standard Apple II memory
- the **Video Graphics Controller (VGC)** and video generation circuitry
- built-in devices and external I/O slots

Although the **Digital Oscillator Chip (DOC)** sound **synthesizer** and support circuitry are new to the Apple II family of computers, they also fall under control of the Mega II side of the computer.

The Fast Processor Interface (FPI) subsystem

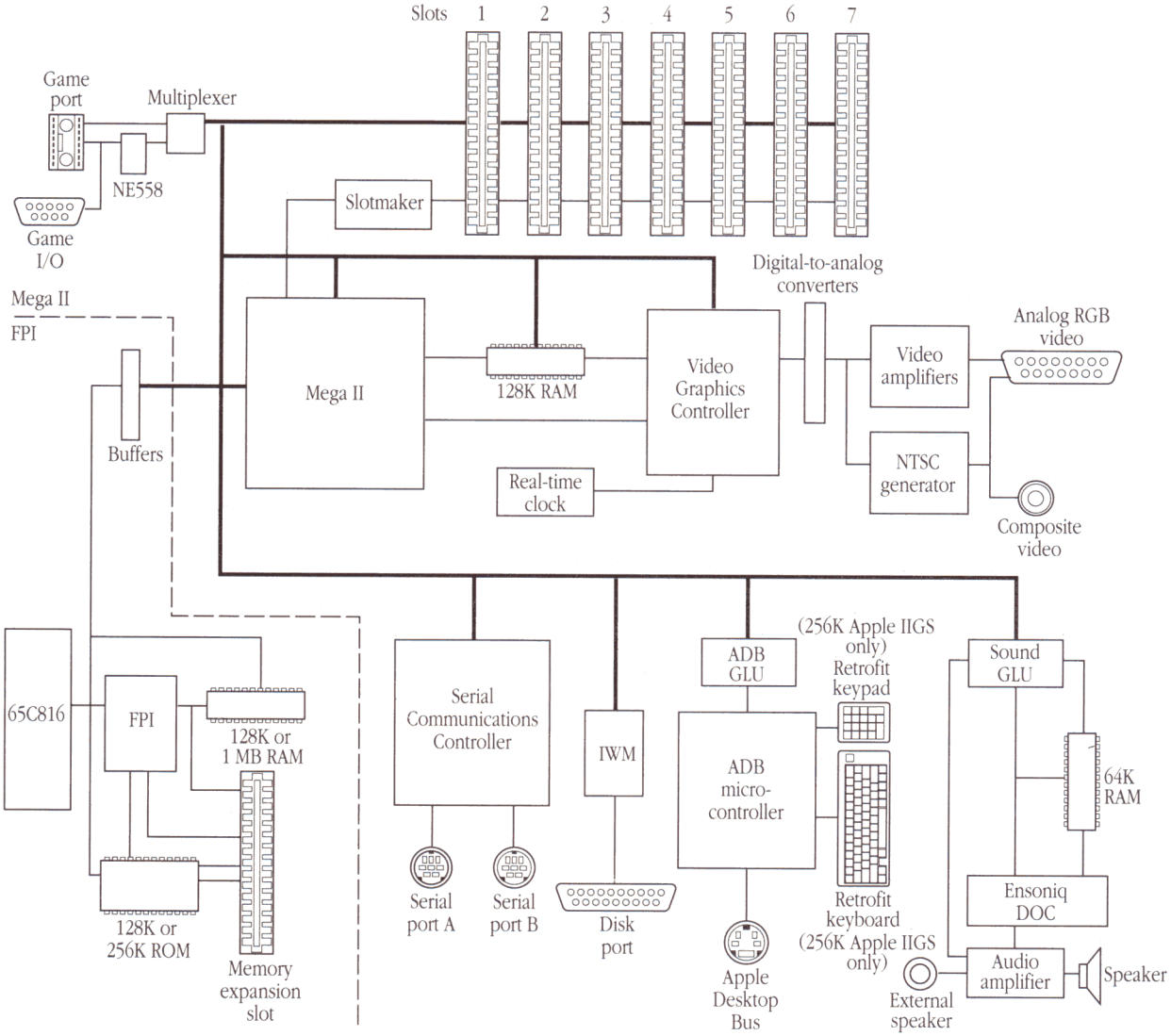
The second subsystem, referred to as the **FPI** portion of the system because of the controlling device, consists of components of the computer that are unique to the Apple IIGS. These are

- the 65C816 microprocessor
- the Fast Processor Interface (FPI) custom IC
- 128K of dynamic **random-access memory (RAM);** the 1 MB Apple IIGS has 1 MB of RAM
- 128K of read-only memory (ROM); the 1 MB Apple IIGS has 256K of ROM

Note that the 65C816 microprocessor is listed as a component of both subsystems. Being a new microprocessor, it has many new instructions that provide this computer with new capabilities. Also, the 65C816 emulates the 6502 microprocessor and will recognize the 6502 instruction set, which means it will run most existing Apple II software.

Figure 1-5 shows the Apple IIGS computer in block form. Note the dashed line separating the two subsystems. Although this is a logical division, it is not absolute: The FPI portion has access to the expansion slots, the Video Graphics Controller, and other components on the Mega II side.

■ **Figure 1-5** Block diagram of the Apple IIGS



Chapter 2 The Core of the Apple IIGS

The design of the Apple IIGS is radically different from that of the standard Apple II. The difference arises primarily from three major components:

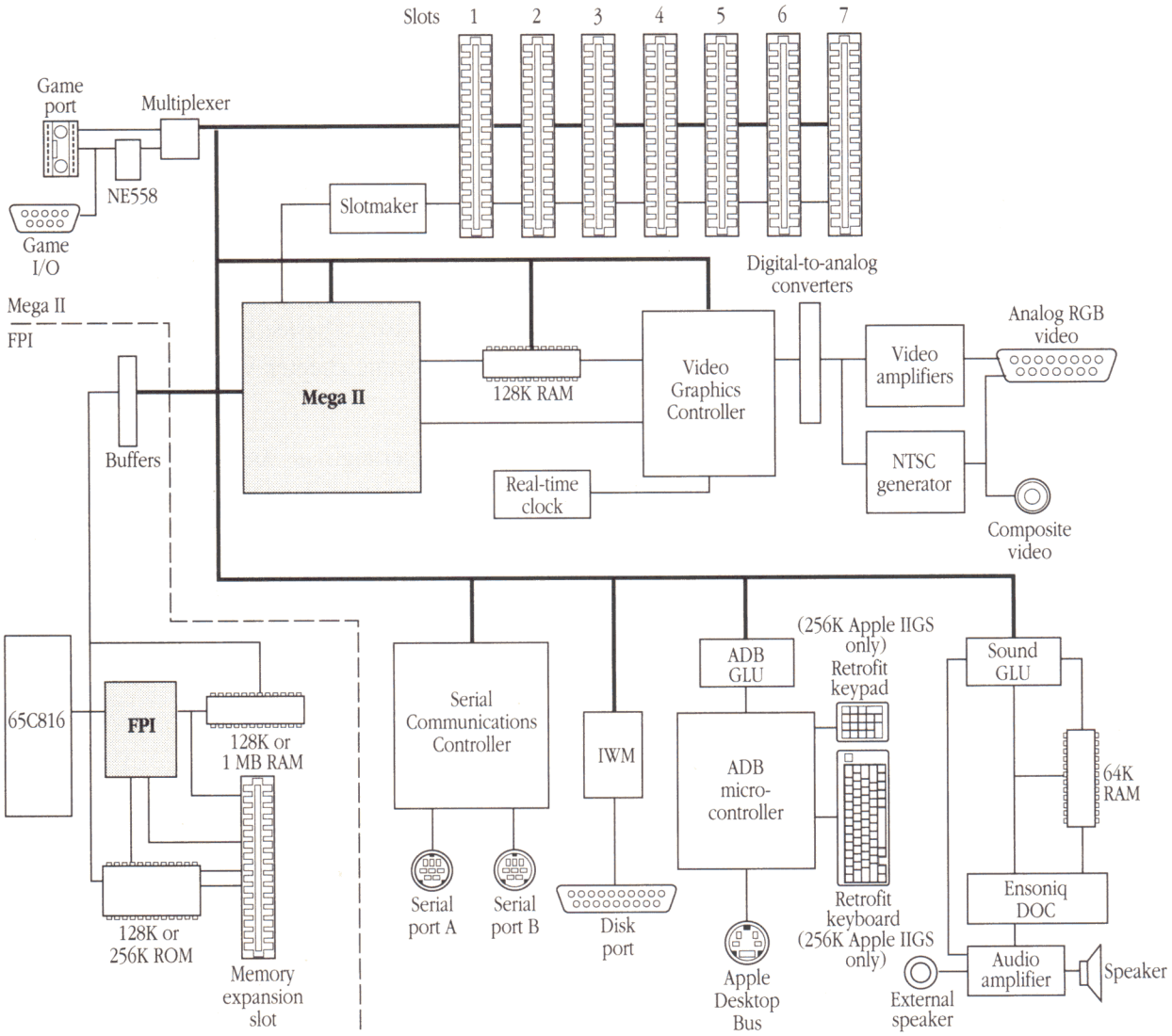
- the 65C816 microprocessor
- the Mega II custom **IC**
- the FPI (Fast Processor Interface) custom IC

The most obvious of these is the 65C816 microprocessor, which is more powerful than the 6502 used in the standard Apple II, yet maintains the ability to execute programs written for the 6502. The 65C816 microprocessor is important enough to have an entire chapter, Chapter 10, devoted to it.

The 65C816 has a larger address space, bigger registers, and the ability to run faster than the standard 1.024-MHz speed of an Apple II. How can the Apple IIGS take advantage of all these capabilities and still be able to run programs written for a standard Apple II? The answer to that question is two custom integrated circuits: the Mega II and the FPI. This chapter describes the way those two ICs work together in the Apple IIGS.

Figure 2-1 is a simplified block diagram of the Apple IIGS, showing all its major components. The Mega II and the FPI are shaded gray. The dashed line separates the part of the Apple IIGS controlled primarily by the Mega II from the part controlled primarily by the FPI.

■ **Figure 2-1** Mega II and FPI subsystems in the Apple IIGS



The Mega II custom IC

The Mega II custom IC combines the functions of several circuits found in the Apple IIe. Those circuits are

- the MMU (Memory Management Unit) custom IC
- the IOU (input/output unit) custom IC
- the **character generator** ROMs
- the video display circuitry

Except for central processor and memory, the Mega II incorporates the logic circuitry for all the major functions of an Apple IIe on a single chip. It works with the I/O expansion slots and the I/O ports built into the Apple IIGS and supports the part of memory that contains the video display **buffers.** The Mega II side of the machine consists of

- the Mega II
- 128K of memory
- the I/O expansion slots
- the built-in I/O ports
- the video display circuitry

The Mega II contains the circuitry that generates video display signals from the data in the display buffers, along with the **soft switches** that select the different display modes.

All I/O in the Apple IIGS is memory mapped. The Mega II provides the address decoding and the soft switches that control the I/O slots and the built-in ports. The Mega II also provides the refresh cycles for the 128K of dynamic RAM under its control.

Because the memory controlled by the Mega II contains the display buffers, it always runs at the 1.024-MHz speed. It is sometimes referred to as *Apple II standard memory,* to distinguish it from the rest of the memory in the Apple IIGS, which normally runs at 2.8 MHz and hence is called *fast memory.*

The FPI custom IC

The FPI (Fast Processor Interface) custom IC supports the 65C816 microprocessor and its large, fast memory. Its name is doubly descriptive: The FPI controls the fast memory itself, and also mediates its interaction with the Mega II side of the machine. Independent control of the two sides enables the Apple IIGS to run programs at 2.8 MHz while maintaining the 1.024-MHz operation required for compatibility with the standard video and I/O circuitry.

For the 65C816 and its fast memory, the FPI provides address multiplexing and control signals. Memory under the control of the FPI includes 128K of built-in RAM (1 MB on the 1 MB Apple IIGS logic board), 128K of built-in ROM (256K on the 1 MB Apple IIGS logic board), up to 4 MB of expansion RAM, and up to 1 MB of expansion ROM. The FPI also generates the refresh cycles needed by the fast dynamic RAM devices. The time required for the refresh cycles reduces the effective processor speed for programs in RAM by about 8 percent. Programs in ROM run at the full 2.8-MHz speed.

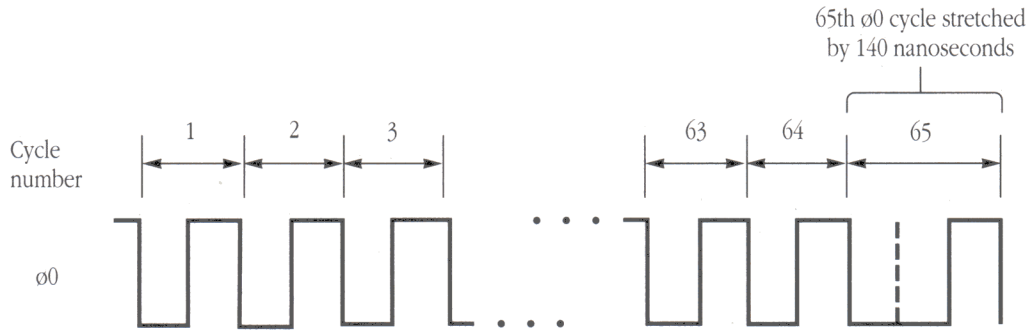
The additional 128K of ROM storage on the 1 MB Apple IIGS logic board is used for storing toolbox utilities as well as enhancements to the system firmware. For complete information on the Apple IIGS firmware, refer to the *Apple IIGS Firmware Reference.* For information on the toolbox utilities, see the *Apple IIGS Toolbox Reference.*

Synchronization

Whenever data have to be transferred between the FPI side and the Mega II side, the FPI IC must first synchronize itself with the 1.024-MHz Mega II. Synchronization may consist of a single Mega II cycle, as when a single I/O location in the Mega II must be accessed, or consecutive Mega II cycles, as when Apple II software must be run at 1.024 MHz for compatibility. For a single Mega II cycle, there is a delay of up to 1 microsecond (average 0.5 microsecond) while waiting for the beginning of the next cycle. For consecutive Mega II cycles, the FPI generates one processor cycle for each Mega II cycle, thus running the processor at 1.024 MHz.

In all Apple II computers, every 65th processor cycle is elongated, or stretched, by 140 nanoseconds. This practice is required for correct colors in the **NTSC** (National Television Standards Committee) video display. Figure 2-2 shows how every 65th clock cycle in all Apple II computers is stretched.

■ **Figure 2-2** Stretched ø0 clock cycle



The Mega II cycle

A Mega II cycle is needed for any central processor or **direct memory access (DMA)** operation that requires access to the 1.024-MHz side of the system. (Refer to Chapter 8, "I/O Expansion Slots," for more information about direct memory access.) These operations are

- all external and most internal I/O operations
- shadowed video-write operations (described in "Memory Shadowing," later in this chapter)
- inhibited memory accesses
- Mega II memory accesses to **banks** $E0 and $E1

A Mega II cycle consists of these steps:

1. A Mega II cycle begins when the FPI recognizes an address that requires access to the 1.024-MHz side of the system—one of the operations just listed.
2. Approximately 90 nanoseconds after the processor ø2 clock signal goes low, the location address and bank address from the processor become valid. The FPI decodes these addresses and determines the type of cycle to be executed before the ø2 clock rises.
3. If the cycle is a Mega II cycle, the FPI holds the ø2 clock high until it synchronizes itself with the Mega II.
4. Memory or I/O access begins.

Mega II auxiliary memory bank access

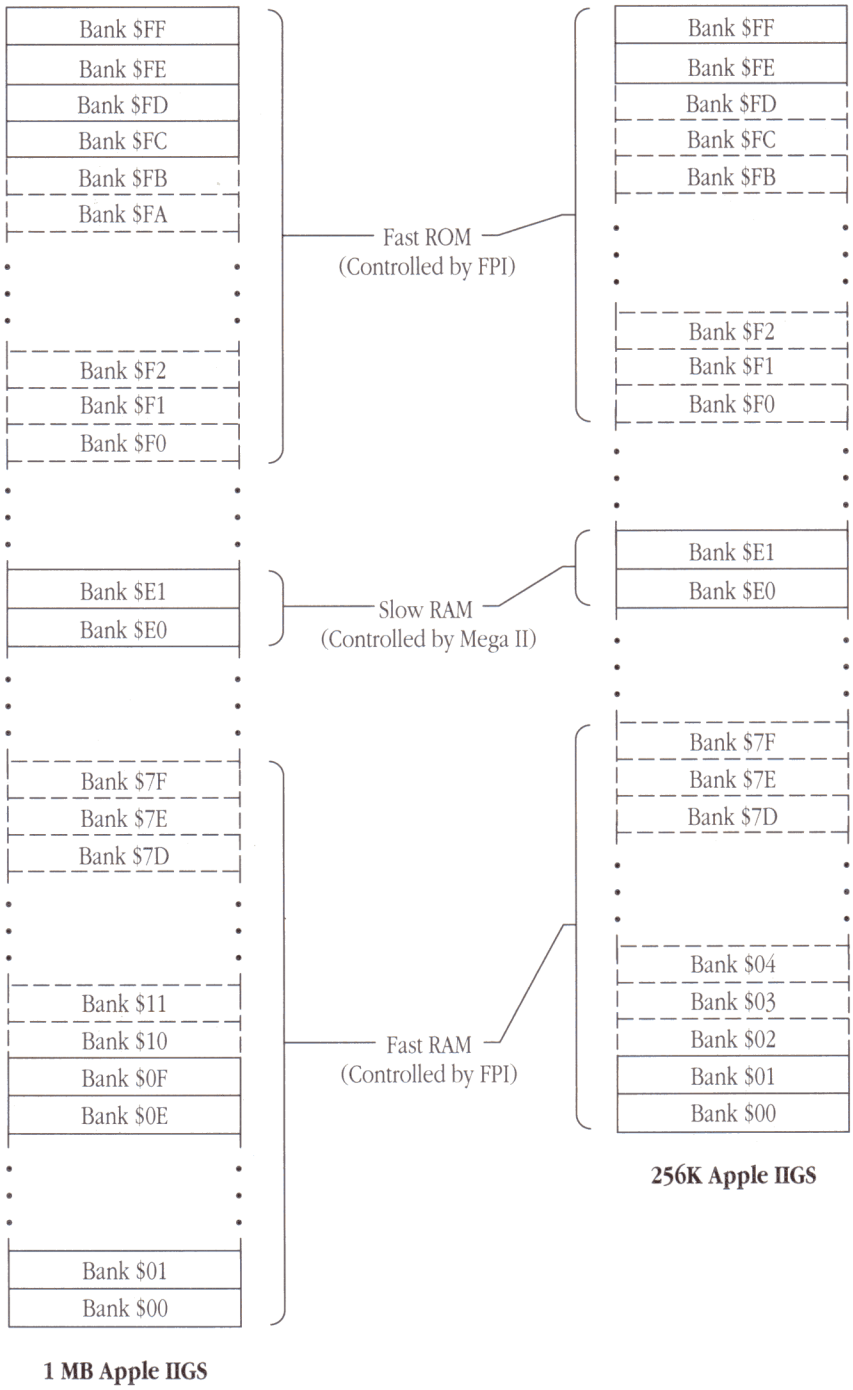
To allow direct access to the Mega II auxiliary memory bank, the FPI passes the least significant bit (lsb) of the bank address to the Mega II during each Mega II cycle. If **shadowing** is enabled (as described in "Memory Shadowing," later in this chapter) or the software is addressing bank $E0 or $E1, an odd-numbered bank address will access the Mega II auxiliary memory automatically, without using the soft switches. (For information on the soft switches, see the discussions on main and auxiliary memory in Chapter 3). For this setup to work, the programmer must first set bit 0 in the New-Video register at $C029 to 1. (See Chapter 4 for information about the New-Video register.) Otherwise, the Mega II ignores the bank bit, and the soft switches must then be used to access the auxiliary 64K through an even-numbered, shadowed bank.

Memory allocation

The FPI controller can access a minimum of 128K of RAM (1 MB on the 1 MB Apple IIGS logic board), which is expandable to 4.3 MB, and to 5 MB on the 1 MB Apple IIGS logic board. This RAM is separate from the 128K of RAM supported by the Mega II. The FPI also has access to 128K of ROM (256K on the 1 MB Apple IIGS logic board), expandable to 1 MB. Figure 2-3 shows a simplified system memory map.

For a full description of memory in the Apple IIGS, refer to Chapter 3.

■ **Figure 2-3** Apple IIGS memory map



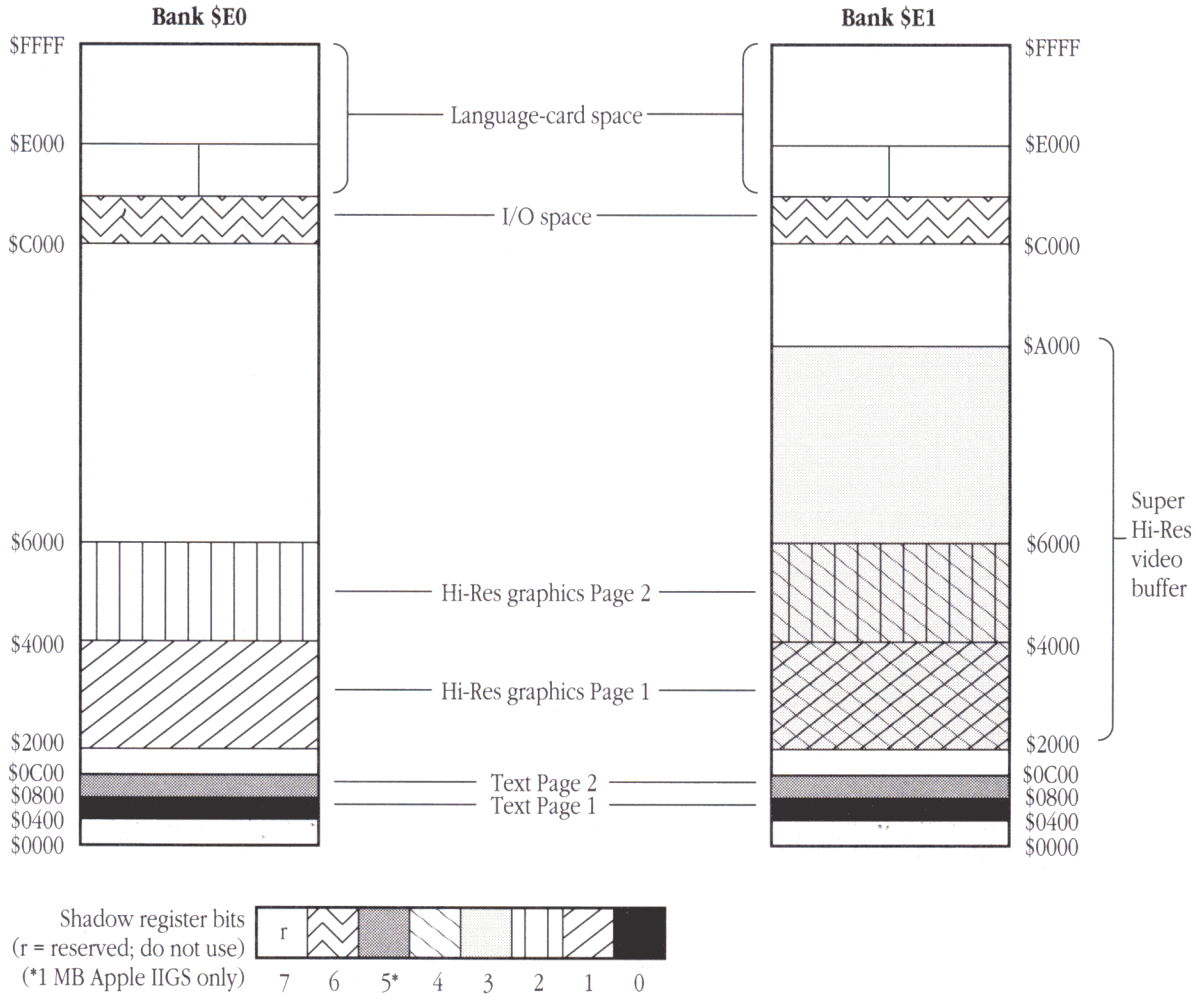
1 MB Apple IIGS

256K Apple IIGS

Memory shadowing

Memory shadowing is the process of reading or writing at one memory location in two different banks. Enabling shadowing duplicates the I/O locations and portions of the video buffers you select (via the Shadow register) in the shadow-enabled RAM banks. Writing into those locations in banks for which shadowing has been enabled results in duplicate writes to those locations in banks $E0 or $E1. Direct access to I/O and the video buffers is not inhibited and may still be obtained through banks $E0 and $E1. (See "I/O Space Addresses," later in this chapter, for more information on I/O addresses.) Figure 2-4 shows banks $E0 and $E1.

■ **Figure 2-4** Shadowed areas of memory



The purpose of shadowing is to provide optimum system speed. By shadowing the I/O and video buffer locations in the high-speed FPI address space, only write instructions to the video locations require the system to operate at 1.024 MHz. A write instruction actually writes to an address in both banks, the Mega II bank, $E0 or $E1, and the shadow-enabled bank, $00 or $01. Read instructions access the high-speed shadowed bank, $00 or $01. Shadowing, therefore, helps minimize the impact of video display updates on the overall system speed. (See "I/O Space Addresses," later in this chapter, for more information on the impact of I/O read operations and write operations on system speed.)

The shadowing options are

- Enable shadowing in banks $00 and $01 only.
- Enable shadowing in all RAM banks (not recommended).

◆ *Note:* Although shadowing is possible in other banks, shadowing in banks other than $00 and $01 should not be attempted under normal operating circumstances; firmware operating in other banks will be corrupted if shadowing is enabled in those banks, resulting in a system crash.

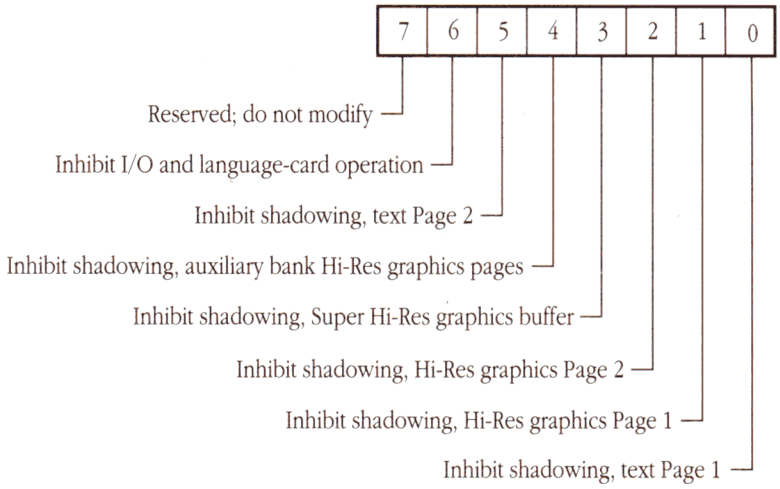
Note that slowing of the system for each write operation is very brief and won't affect program execution speed significantly. Only continuous write accesses would actually be noticeable.

The Shadow register

The Shadow register, located at $C035, determines which address ranges of each shadowed 1.024-MHz RAM bank are duplicated in the FPI RAM display areas. The Shadow register also determines whether or not the I/O space and language-card (IOLC) areas for each bank are activated. Figure 2-5 shows the format of the Shadow register. Table 2-1 is a list of the bits and their functions.

▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 2-5** Shadow register at $C035



■ **Table 2-1** Bits in the Shadow register

Bit	Value	Description
7	–	Reserved; do not modify.
6	0	The I/O and language-card (IOLC) inhibit bit: This bit controls whether the 4K range from $C000 to $CFFF in banks $00 and $01 acts as RAM or as I/O. When this bit is 0, I/O is enabled in the $Cxxx space and the RAM that would normally occupy this space becomes a second $Dxxx RAM space in banks $00 and $01, forming a **language card.** Note that the I/O space and language card in banks $E0 and $E1 are not affected by this bit; this space is always enabled.
	1	When this bit is 1, the I/O space and language card are inhibited, and contiguous RAM is available from $0000 through $FFFF. (For more information on I/O and language-card memory spaces, see Chapter 3, "Memory.")
5	1	Text Page 2 inhibit (available only on the 1 MB logic board): When this bit is 1, shadowing is disabled for text Page 2 and auxiliary text Page 2.

■ **Table 2-1** Bits in the Shadow register (Continued)

Bit	Value	Description
	0	When this bit is 0, shadowing is enabled for text Page 2 and auxiliary text Page 2.
4	1	Inhibit shadowing for auxiliary Hi-Res graphics pages: When this bit is 1, shadowing is disabled for Hi-Res graphics pages 1 and 2 (as determined by bits 0 through 3 in this register) in all auxiliary (odd) banks. Shadowing of Hi-Res graphics pages in the main bank remains unaffected.
	0	When this bit is 0, shadowing is enabled for Hi-Res graphics pages (as determined by bit 1).
3	1	**Super Hi-Res** graphics buffer inhibit: When this bit is 1, shadowing is disabled for the entire 32K video buffer.
	0	When this bit is 0, shadowing is enabled for the Super Hi-Res graphics buffer.
2	1	Hi-Res graphics Page 2 inhibit: When this bit is 1, shadowing is disabled for Hi-Res graphics Page 2 and auxiliary Hi-Res graphics Page 2.
	0	When this bit is 0, shadowing is enabled for Hi-Res video Page 2 and auxiliary Hi-Res video Page 2, unless auxiliary Hi-Res graphics Page 2 shadowing is prohibited by bit 4 of this register.
1	1	Hi-Res graphics Page 1 inhibit: When this bit is 1, shadowing is disabled for Hi-Res graphics Page 1 and auxiliary Hi-Res graphics Page 1.
	0	When this bit is 0, shadowing is enabled for Hi-Res graphics Page 1 and auxiliary Hi-Res graphics Page 1, unless auxiliary Hi-Res graphics Page 1 shadowing is prohibited by bit 4 of this register.
0	1	Text Page 1 inhibit: When this bit is 1, shadowing is disabled for text Page 1 and auxiliary text Page 1.
	0	When this bit is 0, shadowing is enabled for text Page 1 and auxiliary text Page 1.

You can turn shadowing on and off for areas within each shadow-enabled 64K bank by setting the corresponding bit or bits in the Shadow register. You can turn off shadowing (no banks shadowed) by setting all bits in the Shadow register. When the Shadow register is cleared on reset, it defaults to shadowing all video areas.

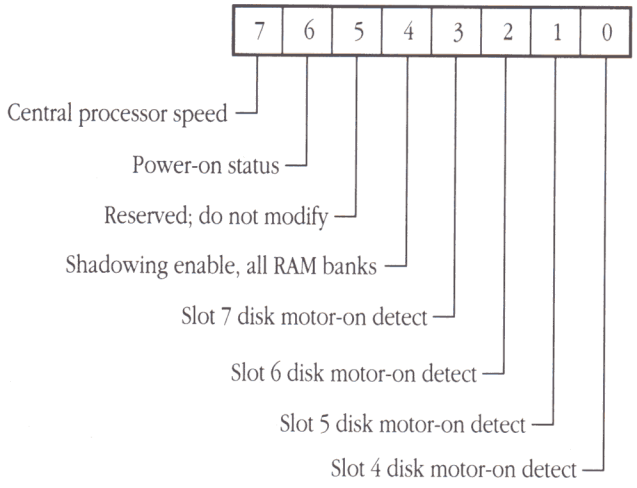
Each bit in the Shadow register is active high, which means that the shadowing of the selected area is inhibited if the corresponding bit is set. Programs that use the Shadow register can turn off shadowing in unused video areas by setting the appropriate bits, thus reclaiming the memory space in the unused video buffers in Mega II banks $00 and $01.

The Speed register

The Speed register, located at $C036, contains bits that control the speed of operation and that determine whether a specific area within a bank is shadowed. The Speed register is cleared on reset or power up, except for bit 6, which on power up is set. Figure 2-6 shows the format of the Speed register. Table 2-2 contains a description of the bits.

▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 2-6** Speed register at $C036



■ **Table 2-2** Bits in the Speed register

Bit	Value	Description
7*	1	System operating speed. When this bit is 1, the system operates at 2.8 MHz.
	0	When this bit is 0, the system operates at 1.024 MHz (as in other Apple II computers).
6	1	Power-on status (available only on the 1 MB logic board): This bit is set to 1 when the system is turned on using the power switch. A boot initiated by any key combination will not alter this bit. This is a read-write bit.
	0	n/a
5	–	Reserved; do not modify.
4	1	Bank shadowing bit: This bit determines memory shadowing in the RAM banks. Shadow register bits 0 through 4 will determine which portion, if any, of the banks will be shadowed. To enable shadowing in all RAM banks, $00 through $7F, set this bit to 1.
	0	To enable shadowing in banks $00 and $01 only, clear this bit. For proper operation of the Apple IIGS operating system, this bit must always be set to 0.
0–3†	1	Disk II motor-on address detectors: To retain Apple II peripheral compatibility, the motor-on detectors change the system speed to 1.024 MHz whenever a Disk II motor-on address is detected.‡ When the disk motor-off address is accessed, the system speed increases to 2.8 MHz again. For example, when bit 1 is 1, the FPI switches to 1.024 MHz when address $C0D9 is accessed, and returns to 2.8 MHz following a $C0D8 access. (See list of addresses below.)

* Drives designed for the Apple IIGS system should use the speed bit (Speed register bit 7) to change the processor speed when accessing disks, rather than the disk motor-on detectors (Speed register bits 0 through 3). By using bit 7, you access drives in slots other than slots 4 through 7 by changing the system speed manually. Be aware that central processor speed changes for drive compatibility may affect **application program** timing; avoid using the motor addresses unless they are used in a fashion consistent with the drive's central processor speed requirements.

† For compatibility with future Apple products, use firmware calls only to manipulate bits 0 to 3 of the Speed register.

‡ Drives designed for previous Apple II computers will function as Apple IIGS peripherals only if the system speed is changed to 1.024 MHz before disk access is attempted.

(Continued)

■ **Table 2-2** Bits in the Speed register (Continued)

Bit	Value	Description
	0	When this bit is 0, the Disk II motor detectors are turned off. Bits 0 through 3 detect the following addresses:

Bit	Slot	Motor on	Motor off
0	4	$C0C9	$C0C8
1	5	$C0D9	$C0D8
2	6	$C0E9	$C0E8
3	7	$C0F9	$C0F8

RAM control

The FPI alone controls the high-speed RAM. This high-speed memory consists of a minimum of 128K of RAM (1 MB in the 1 MB Apple IIGS) on the main logic board and additional expansion RAM on the extended memory card, for a total of 4.3 MB in the 256K logic board, and 5 MB in the 1 MB version of the board.

The FPI provides memory refresh for the high-speed RAM, which incorporates internal refresh-address counters. This refresh scheme frees the address bus so that the FPI can execute ROM cycles while RAM refresh cycles are occurring, thus allowing full-speed operation in the ROM. These cycles occur approximately every 3.5 microseconds and reduce the 2.8-MHz processing speed by approximately 8 percent for programs that run in RAM. When running at 1.024 MHz, refresh cycles are executed during an unused portion of the processor cycle and do not affect the processor speed.

I/O space addresses

The I/O space in the Apple IIGS consists of all the addresses from $C000 through $CFFF. All internal device addresses, register addresses, soft switch addresses, and slot addresses fall within this 4K address range. Any of these addresses can be accessed through banks $E0, $E1, $00, and $01. Access from banks $E0 and $E1 is always enabled; access from banks $00 and $01 is controlled by bit 6 of the Shadow register, and must always be enabled for correct system operation.

When estimating the performance of timing-critical code, you must consider the impact processor speed changes have on execution speed. The Apple IIGS can operate at 2.8 MHz, but must slow down to 1.024 MHz when accessing certain I/O addresses. These I/O addresses include I/O reads and writes, and instruction reads of firmware at slot addresses of $C100 through $CFFF. Additionally, all reads and writes to soft switches and slot I/O devices at addresses $C090 through $C0FF also occur at 1.024 MHz.

◆ *Note:* In order to guarantee that your code will remain compatible with future Apple II computers, do not develop timing-critical code that will not function at system speeds greater than 2.8 MHz.

A microprocessor instruction consists of between two and nine individual cycles. For instructions executed from fast RAM or ROM, only the specific instruction cycles that read from or write to I/O addresses will slow the system to 1.024 MHz. All other cycles of such an instruction will execute as fast cycles. The result is that the majority of instruction cycles occur at high speed. The few that occur at low speed are of variable length. This length can, however, be estimated. The following rules provide a simple method of calculating the minimum and maximum time that an entire instruction will require to execute:

1. If a single (8-bit) slow I/O read or write *cycle* is perfectly synchronized, it takes nearly three fast cycles to complete. A double (16-bit) slow I/O read or write *cycle* takes nearly 6 fast cycles to complete. Thus, an 8-bit read or write *instruction* that would normally take four fast cycles will take at least six fast cycles, an increase of two cycles. A 16-bit read or write instruction that would normally take five fast cycles will take at least nine fast cycles, an increase of four cycles.
2. If either a single or double slow cycle is not synchronized, the maximum delay for synchronization is one extra slow cycle, adding the equivalent of three fast cycles to the count. Thus, the worst-case 8-bit access becomes 2 + 3 or 5 extra fast cycles, and the 16-bit worst case becomes 4 + 3 or 7 extra fast cycles.

These rules can be applied to the cycle times for any instruction executing in fast RAM or ROM to approximate the minimum and maximum times for instructions that reference I/O addresses. Remember to allow an additional 10% in total cycle time to account for RAM refresh delays.

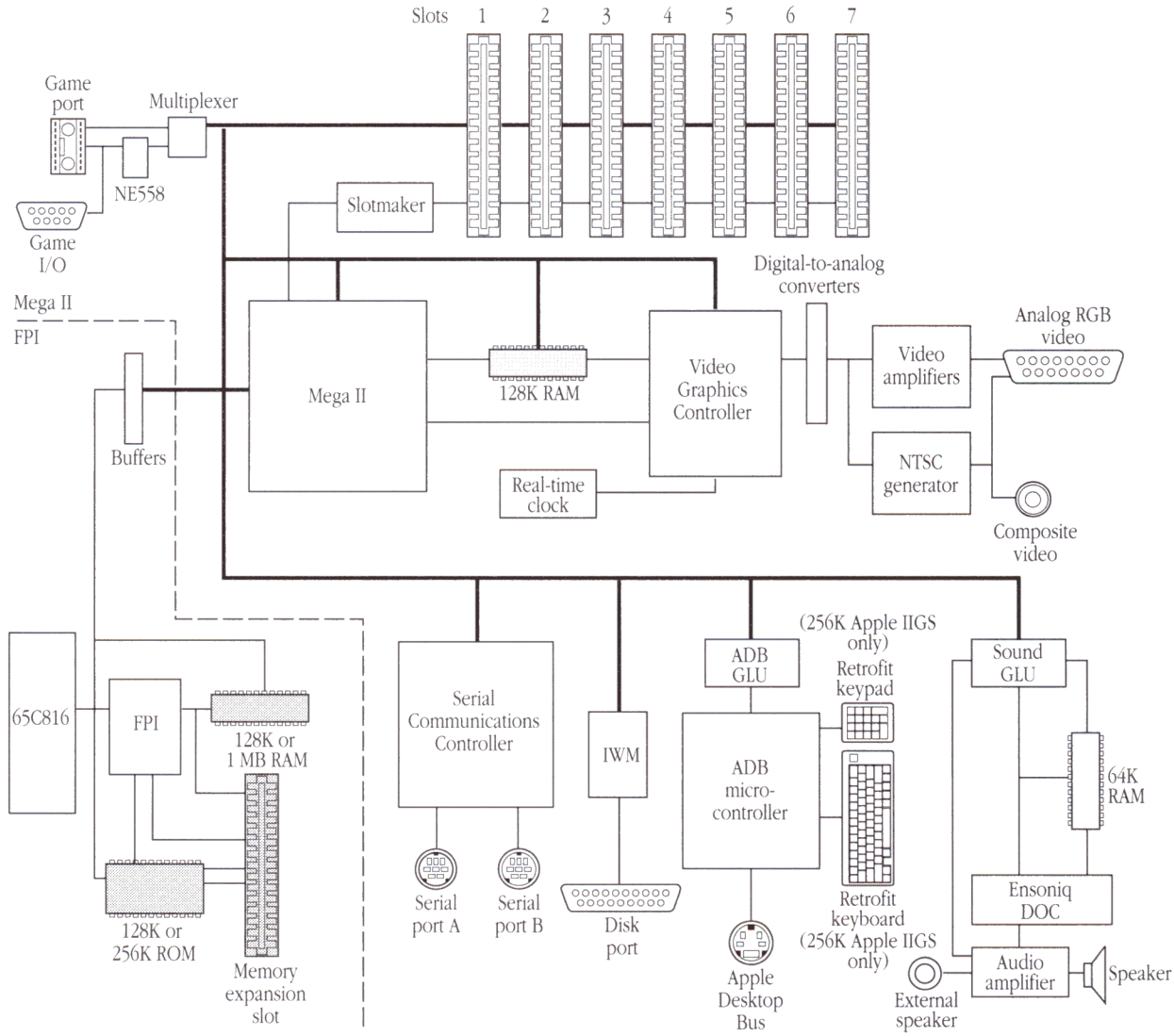
Certain registers internal to the FPI (the DMA register, the Speed register, and the Shadow register) are read and written at high speed. Similarly, reading the interrupt ROM addresses ($C071 through $C07F) does not slow the system. In addition, two registers (the State register and the Slot ROM Select register) that exist in both the FPI and the Mega II ICs are written at 1.024 MHz and read at 2.8 MHz in the FPI address space.

Chapter 3 **Memory**

This chapter describes the internal memory of the Apple IIGS, and shows the RAM and ROM memory layout and how the memory is controlled. There is a memory map for the entire system, and there are individual maps for special features like standard Apple II compatibility and memory shadowing.

The memory in the Apple IIGS is divided into several portions. Figure 3-1 is a block diagram showing the different parts of memory in relation to the rest of the hardware; Figure 3-2 is a memory map showing the addresses of the different parts of memory. As described in Chapter 2, the greater part of the memory is controlled by the FPI, while two 64K banks ($E0 and $E1) are controlled by the Mega II so that the Apple IIGS may function like a standard Apple II.

■ **Figure 3-1** Memory in the Apple IIGS



Built-in memory

The original Apple IIGS comes with 256K of main memory mounted on the circuit board, and the 1 MB Apple IIGS has 1 MB. Additional memory can be added by means of an optional memory expansion card you can plug into the memory expansion slot, which is described in the latter part of this chapter.

As you can see by looking at the block diagram in Figure 3-1, memory in the Apple IIGS is divided into several portions. The original Apple IIGS uses ten 64K–by–4-bit RAM ICs on the main logic board. Four RAM ICs make up the 128K controlled by the Mega II, and four more are the 128K of fast system memory controlled by the FPI. The 1 MB Apple IIGS varies from this configuration slightly, using eight 1-megabit RAM ICs for fast system memory. Besides the main memory, there are also two RAM ICs for the 64K of RAM dedicated to sound generation. The sound RAM is not directly addressable by application programs; for more information about the sound memory, refer to Chapter 5.

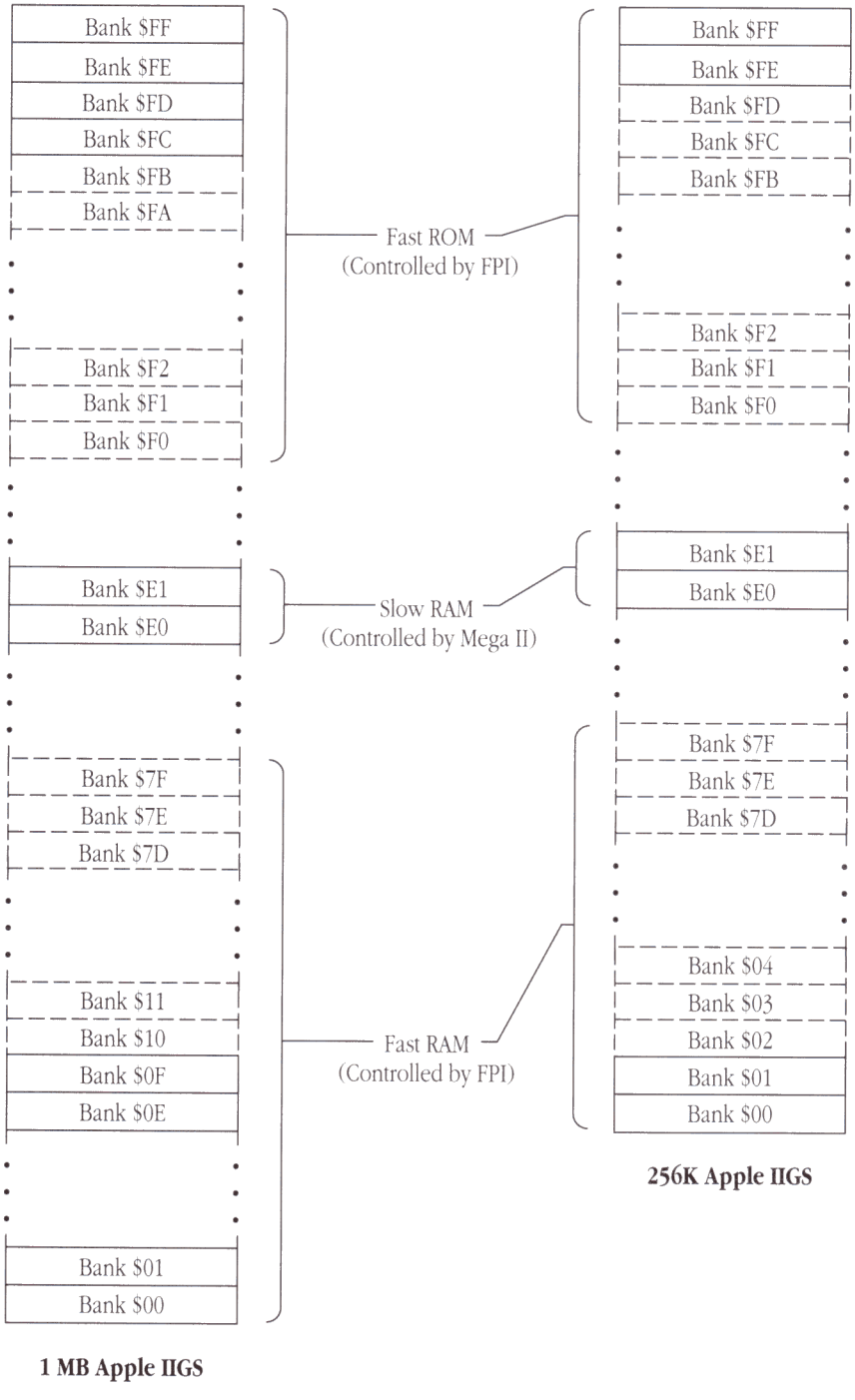
Memory map

The 65C816 microprocessor is capable of addressing up to 16 MB of memory, but only portions of this memory space are utilized in the Apple IIGS. Figure 3-2 shows how that memory space is allocated in the Apple IIGS. A portion of the lower memory space—a maximum of 5 MB—is available for fast RAM under the control of the FPI. The first 128K is built into the original Apple IIGS, 1 MB in the 1 MB Apple IIGS; the rest can be added by means of a memory expansion card.

The 128K of RAM controlled by the Mega II occupies banks $E0 and $E1. No further expansion of this part of memory is possible.

The highest 16 banks are allocated to ROM under the control of the FPI. The top 128K of ROM is built into the original Apple IIGS; the uppermost 256K of ROM is built into the 1 MB Apple IIGS. Additional ROM can be added by means of a memory expansion card.

■ **Figure 3-2** Memory map of the Apple IIGS (Solid lines indicate built-in memory; dashed lines indicate expansion memory.)



Memory bank allocation

The memory in the Apple IIGS is addressed as 64K banks, as shown in Figure 3-2. Bank numbers are in hexadecimal. The built-in memory banks are shown with solid outlines: banks $00 and $01, $E0 and $E1, and $FE and $FF in the 256K Apple IIGS. The parts of the memory space from bank $02 to bank $7F and from bank $F0 to bank $FD are allocated for memory expansion; banks $F8 through $FF are reserved for current system and future expansion of system firmware. Memory spaces from $80 through $EF are not available in the Apple IIGS.

The memory bank distribution in the 1 MB Apple IIGS is similar to that of the 256K system, with a few variations. The built-in banks are banks $00 through $0F, $E0 and $E1, and $FC through $FF. Banks $10 through $7F, and $F0 through $FB in the 1 MB system are available for memory expansion.

Address wrapping

In general, the 65C816 microprocessor used in the Apple IIGS addresses memory as continuous across bank boundaries, but there are exceptions. One kind of exception involves the 65C816's instructions themselves. For data at the highest address in a bank, the next **byte** is the lowest one in the next bank, but instructions themselves wrap around on bank boundaries, rather than advancing to the next bank. That means that the maximum size of a program segment is normally limited to 64K. For more information about the 65C816, refer to Chapter 10.

Another exception to the continuity of memory arises from the way certain banks are used for special purposes. For example, parts of banks $E0 and $E1 are set aside as video display buffers and are not normally used for program code, although the hardware doesn't prevent such use. The **Memory Manager,** which is part of the **Apple IIGS Toolbox,** takes such restrictions into account. For information about the Memory Manager, refer to the *Apple IIGS Toolbox Reference.*

ROM memory

The two highest banks in the 256K system, and the four highest banks in the 1 MB system, are used for built-in ROM that contains system programs and part of the Apple IIGS Toolbox. Additional memory in banks $F0 to $FD ($FB in the 1 MB system) is available for ROM on a memory expansion card. Of that memory, part is available for application programs stored as a **ROM disk,** and part is reserved for future expansion of system programs. For information about ROM disks, refer to the *Apple IIGS ProDOS 16 Reference.*

Bank $00 memory allocation

Memory bank $00 preserves many features found in the 64K of main memory in the Apple IIe or the Apple IIc that make it possible to run programs originally written for those machines or for the Apple II Plus.

Reserved memory pages

Most of bank $00 is available for storing programs and data. However, a few pages of bank $00 are reserved for the use of the Monitor firmware and the BASIC interpreter. The reserved pages are described in the following sections.

△ **Important** The system does not prevent your using these pages, but if you do use them, you must be careful not to disturb the system data they contain, or you will cause the system to malfunction. △

- *Apple II note:* Some of the reserved areas described in the sections that follow are used only by programs written for models of the Apple II that preceded the Apple IIGS. Programs written specifically for the Apple IIGS normally do not deal with hardware features directly, but rely on routines in the toolbox, as described in the *Apple IIGS Toolbox Reference.* Some reserved areas are used by the built-in firmware: Refer to the *Apple IIGS Firmware Reference.*

Direct page: Several of the 65C816 microprocessor's addressing modes require the use of addresses in a specified page of bank $00 called the **direct page.** Like the **zero page** in a 6502 microprocessor, the direct page is used for indirect addressing.

The direct page works differently in the two microprocessor modes. When the 65C816 is in **emulation mode,** the direct page is located at address $0000 in bank $00, like the zero page in a 6502 microprocessor's 64K address space. When the 65C816 is in **native mode,** the direct page can be located anywhere in bank $00, making it possible for different programs to have different direct page locations. (For more information about emulation mode and native mode, see Chapter 10.)

To use indirect addressing in your assembly-language programs, you must store base addresses in a direct page. At the same time, you must avoid interfering with direct-page memory used by other programs such as the Monitor program, the BASIC interpreter, and the **disk operating systems.** The best way to avoid conflicts is to request your own direct-page space from the Memory Manager: Refer to the *Apple IIGS Toolbox Reference.*

The 65C816 stack: The 65C816 microprocessor uses a **stack** to store subroutine return addresses in last-in, first-out sequence. Many programs also use the stack for temporary storage of the registers and for passing parameters to subroutines.

The 65C816 uses the stack two ways—in emulation mode and native mode. In emulation mode, the stack **pointer** is 8 bits long, and the stack is located in page 1 (locations $100 through $1FF, hexadecimal) and can hold 256 bytes of information. When you store the 257th byte in the stack, the stack pointer repeats itself, or wraps around, so that the new byte replaces the first byte stored, which is then lost. This writing over old data is called *stack overflow*. The program continues to run normally until the lost information is needed, whereupon the program may behave unpredictably, or, possibly, terminate catastrophically.

▲ **Warning** The wrapping around of the stack pointer does not occur consistently; in some addressing modes the stack will continue to page 2. In either case, a system crash is imminent. ▲

In native mode, the stack pointer is 16 bits long, and the stack can hold up to 64K of information at a time. To read more about using the 65C816 stack, see Chapter 10.

The input buffer: The GETLN input routine, which is used by the built-in Monitor program and Applesoft BASIC interpreter, uses page 2 of bank $00 as its keyboard-input buffer. The size of this buffer sets the maximum size of input strings. (Note that BASIC uses only the first 237 bytes, although it permits you to type in 256 **characters.**) If you know that you won't be typing any long input strings, you can store temporary data at the upper end of page 2.

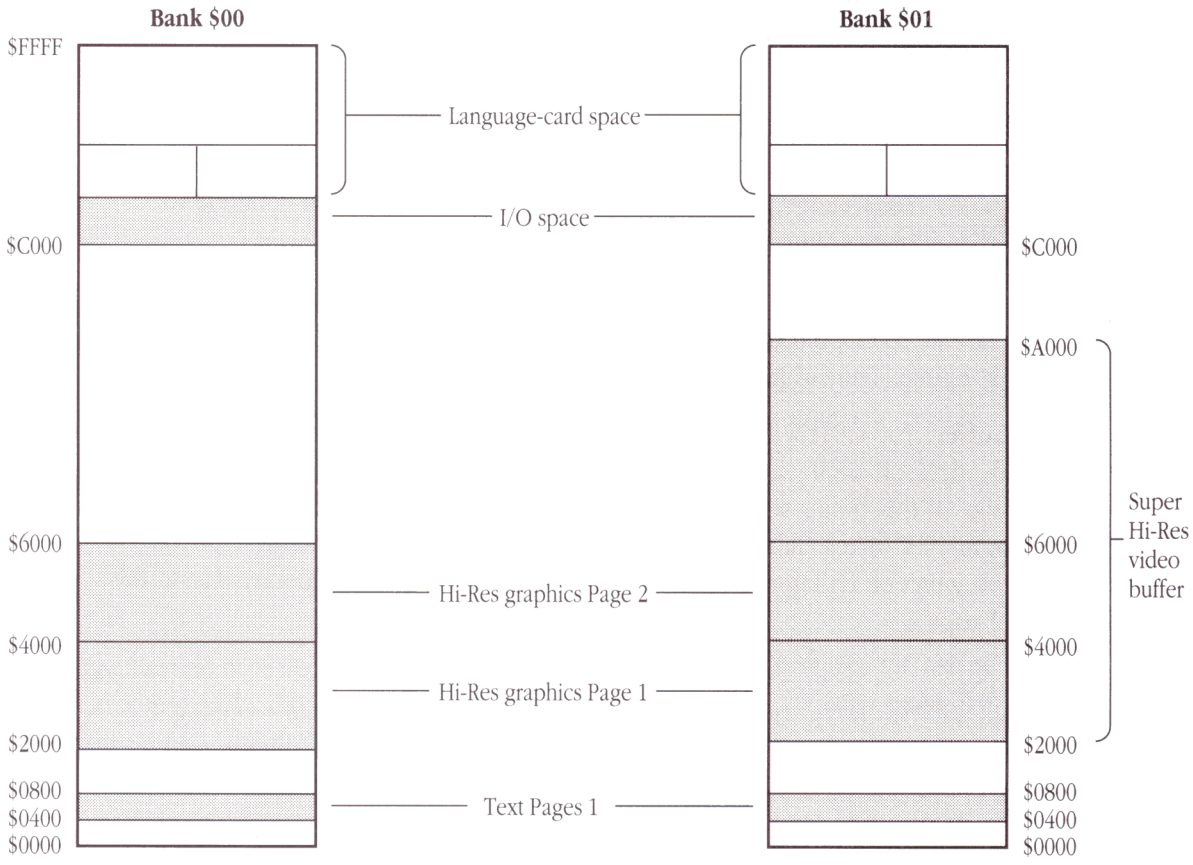
◆ *Note:* Routines that use the input buffer run in emulation mode; programs running in native mode must first switch to emulation mode to call such routines. Refer to the *Apple IIGS Firmware Reference* for more information.

Link-address storage: The Monitor program, **ProDOS®**, and **DOS** 3.3 all use the upper part of page 3 for link addresses or vectors. BASIC programs sometimes need short assembly-language routines. These routines are usually stored in the lower part of page 3.

Shadowed display spaces: The display buffers in the Apple IIGS are actually located in banks $E0 and $E1, but programs written for the Apple II Plus, the Apple IIe, and the Apple IIc put display information into the corresponding locations in bank $00 and require display shadowing to be on. Figure 3-3 shows the shadowed display spaces. For more information about shadowing, refer to Chapter 2.

◆ *Note:* Display buffers in bank $00 are normally used only by programs written for earlier models of the Apple II, except for text Page 1, which is also used by the Control Panel desk accessory. Shadowing of the display buffers is enabled by a switch in the Shadow register, described in Chapter 2.

■ **Figure 3-3** Shadowed display spaces in banks $00 and $01 (Shading indicates shadowed memory locations.)



The primary text and Lo-Res graphics display buffer uses memory locations $0400 through $07FF. This 1024-byte area is called text Page 1, and it is not usable for program and data storage when shadowing is on. There are 64 locations in this area that are not displayed on the screen; these locations, called **screen holes,** are reserved for use by the peripheral cards and the built-in ports. See the section "Peripheral-Card RAM Space," in Chapter 8, for the locations of the screen holes.

- *Text Page 2:* The original Apple IIGS doesn't shadow text Page 2. To make it possible to run Apple II programs that use text Page 2 for their displays, the firmware includes a desk accessory, Alternate Display Mode, that automatically transfers data from text Page 2 of bank $00 into text Page 2 of bank $E0, where it can be displayed. Refer to the *Apple IIGS Firmware Reference* for more information. Note that the 1 MB Apple IIGS has available a Shadow register bit that allows you to shadow Text Page 2.

When the primary Hi-Res graphics display buffer, Hi-Res graphics Page 1, is shadowed, it uses memory locations $2000 through $3FFF. If your program doesn't use Hi-Res graphics, this area is usable for programs or data.

Hi-Res graphics Page 2 uses memory locations $4000 through $5FFF. Most programs do not use Hi-Res graphics Page 2, so they can use this area for program or data storage.

The primary Double Hi-Res graphics display buffer, called Double Hi-Res graphics Page 1, uses memory locations $2000 through $3FFF in both main and auxiliary memory (banks $00 and $01). If your program doesn't use Hi-Res or Double Hi-Res graphics, this area of memory is usable for programs or data.

Language-card memory space

- *Apple II note:* The language-card space is a carryover from earlier models of the Apple II and is normally used only by programs written for those machines and running in emulation mode. Like the bank $00 shadowing of the display buffers, the peculiar features of the language-card space are enabled by a switch in the Shadow register, which is described in Chapter 2.

- *Memory banks:* The language-card space is a feature both of bank $00 and of bank $01. Refer to the section "Bank $01 (Auxiliary Memory)," later in this chapter, for more information.

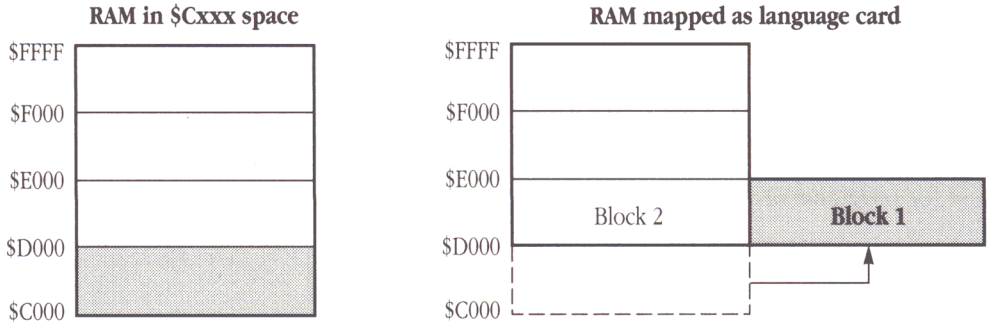
When the language-card feature is enabled, the memory address space from $D000 through $FFFF is doubly allocated: It is used for both ROM and RAM. The 12K of ROM in this address space contains the Monitor program and the Applesoft BASIC interpreter. Alternatively, there are 16K of RAM in this space. The RAM is normally used by the disk **operating system.**

You may be wondering why this part of memory has such a split personality. Some of the reasons are historical: The Apple IIGS is able to run software written for a standard Apple II because it uses this part of memory in the same way a standard Apple II does. It's convenient to have the Applesoft BASIC interpreter in ROM, but the Apple IIGS is also able to use that address space for other things when Applesoft is not needed.

You may also be wondering how 16K of RAM are mapped into only 12K of address space. The usual answer is that it's done with mirrors, and that isn't a bad analogy: The 4K address space from $D000 through $DFFF is used twice.

Switching different blocks of memory into the same address space is called *bank switching*. There are actually two examples of bank switching going on here: First, the entire address space from $D000 through $FFFF is switched between ROM and RAM, and second, the address space from $D000 to $DFFF is switched between two different blocks of RAM. If the language card is not enabled, the first of these blocks of RAM, block 1, occupies address space from $C000 to $CFFF, as shown in Figure 3-4. (Note that the banks involved here are not the same as the 64K memory banks.)

■ **Figure 3-4** Language-card memory map



Setting language-card bank switches: You switch banks in the language-card space in the same way you switch other functions in a standard Apple II: by using soft switches. Read operations to the soft-switch locations do three things: select either RAM or ROM in this memory space, enable or inhibit writing to the RAM, and select the first or second 4K bank of RAM in the address space $D000 to $DFFF.

▲ **Warning** Do not use these switches without careful planning. Careless switching between RAM and ROM is almost certain to have catastrophic effects on your program. ▲

Table 3-1 shows the addresses of the soft switches for enabling all combinations of reading and writing in this memory space. All the hexadecimal values of the addresses are of the form $C08x. Notice that several addresses perform the same function: This is because the functions are activated by single address bits. For example, any address of the form $C08x with a 1 in the **low-order** bit enables the RAM for writing. Similarly, bit 3 of the address selects which 4K block of RAM to use for the address space $D000 to $DFFF; if bit 3 is 0, the first bank of RAM is used, and if bit 3 is 1, the second bank is used.

When RAM is not enabled for reading, the ROM in this address space is enabled. Even when RAM is not enabled for reading, it can still be written to if it is write-enabled.

■ **Table 3-1** Language-card bank-select switches

Name	Action	Location	Function
	R	$C080	Read this location to read RAM, write-protect RAM, and use $D000 bank 2.
ROMIN	RR	$C081	Read this location twice to read ROM, write-enable RAM, and use $D000 bank 2.
	R	$C082	Read this location to read ROM, write-protect RAM, and use $D000 bank 2.
LCBANK2	RR	$C083	Read this location twice to read RAM, write-enable RAM, and use $D000 bank 2.
	R	$C088	Read this location to read RAM, write-protect RAM, and use $D000 bank 1.
	RR	$C089	Read this location twice to read ROM, write-enable RAM, and use $D000 bank 1.
	R	$C08A	Read this location to read ROM, write-protect RAM, and use $D000 bank 1.
	RR	$C08B	Read this switch twice to read RAM, write-enable RAM, and use $D000 bank 1.
RDLCBNK2	R7	$C011	Read this location and test bit 7 for switch status: $D000 bank 2 (1) or bank 1 (0).
RDLCRAM	R7	$C012	Read this location and test bit 7 for switch status: RAM (1) or ROM (0).
SETSTDZP	W	$C008	Write this location to use main bank, page 0 and page 1.
SETALTZP	W	$C009	Write this location to use auxiliary bank, page 0 and page 1.
RDALTZP	R7	$C016	Read this location and test bit 7 for switch status: auxiliary (1) or main (0) bank.

When you turn power on or reset the Apple IIGS, the bank switches are initialized for reading from the ROM and writing to the RAM, using the second bank of RAM. Note that this is different from the reset on the Apple II Plus, which doesn't affect the **bank-switched memory** (the language card). On the Apple IIGS, you can't use the reset key sequence to return control to a program in bank-switched memory, as you can on the Apple II Plus.

- *Reading and writing to RAM banks:* You can't read one RAM bank and write to the other; if you select either RAM bank for reading, you get that one for writing as well.

- *Reading RAM and ROM:* You can't read from ROM in part of the bank-switched memory and read from RAM in the rest. Specifically, you can't read the Monitor program in ROM while reading bank-switched RAM. If you want to use the Monitor firmware with a program in bank-switched RAM, copy the Monitor program from ROM (locations $F800 through $FFFF) into bank-switched RAM. You can't do this from Pascal or ProDOS.

To see how to use these switches, look at the following section of an assembly-language program:

```
LDA $C083    *SELECT 2ND 4K BANK & READ/WRITE
LDA $C083    *BY TWO CONSECUTIVE READS
LDA #$D0     *SET UP...
STA BEGIN    *...NEW...
LDA #$FF     *...MAIN-MEMORY...
STA END      *...POINTERS...
JSR YOURPRG *...FOR 12K BANK
LDA $C08B    *SELECT 1ST 4K BANK
JSR YOURPRG *USE ABOVE POINTERS
LDA $C088    *SELECT 1ST BANK & WRITE PROTECT
LDA #$80
INC SUM
JSR YOURSUB
LDA $C080    *SELECT 2ND BANK & WRITE PROTECT
INC SUM
LDA #PAT12K
JSR YOURSUB
LDA $C08B    *SELECT 1ST BANK & READ/WRITE
LDA $C08B    *BY TWO CONSECUTIVE READS
INC NUM      *FLAG RAM IN READ/WRITE
INC SUM
```

The LDA instruction, which performs a read operation to the specified memory location, is used for setting the soft switches. The unusual sequence of two consecutive LDA instructions performs the two consecutive reads that write-enable this area of RAM; in this case, the data that are read are not used.

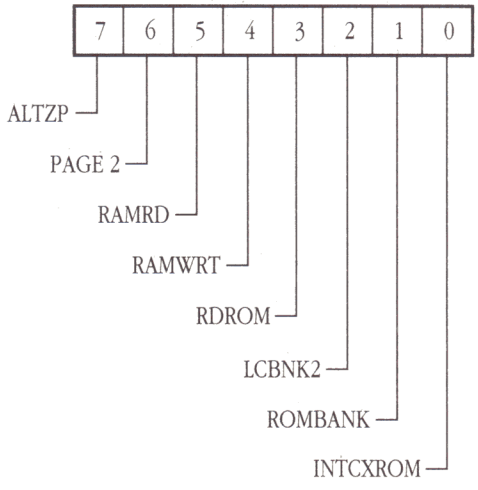
Reading bank switches: You can find out which language-card bank is currently switched in by reading the soft switch at $C011. You can find out whether the language card or ROM is switched in by reading $C012. The only way that you can find out whether or not the language-card RAM is write-enabled is by trying to write some data to the card's RAM space.

The State register

The State register is a read/write register containing eight commonly used standard Apple II soft switches. Compared to the use of separate soft switches, the single-byte format of the State register simplifies the process of interrupt handling. Reading and storing this byte before executing interrupt routines allows you to restore the system soft switches to the previous state in minimum time after returning from the interrupt routine. Write operations to the State register will slow the system momentarily. (See Figure 3-5 and Table 3-2.)

▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 3-5** State register at $C068



■ **Table 3-2** Bits in the State register

Bit	Value	Description
7	1	ALTZP: If this bit is 1, then bank-switched memory, stack, and direct page are in main memory.
	0	If this bit is 0, then bank-switched memory, stack, and direct page are in auxiliary memory.
6	1	PAGE2: If this bit is 1, text Page 2 is selected.
	0	If this bit is 0, text Page 1 is selected.
5	1	RAMRD: If this bit is 1, auxiliary RAM bank is read-enabled.
	0	If this bit is 0, main RAM bank is read-enabled.
4	1	RAMWRT: If this bit is 1, auxiliary RAM bank is write-enabled.
	0	If this bit is 0, main RAM bank is write-enabled.
3	1	RDROM: If this bit is 1, the selected language-card ROM is read-enabled.
	0	If this bit is 0, the selected language-card RAM bank is read-enabled.
2	1	LCBNK2: If this bit is 1, language-card RAM bank 1 is selected.
	0	If this bit is 0, language-card RAM bank 2 is selected.
1	–	ROMBANK: The ROM bank select switch must always be 0. To maintain system integrity, do not modify this bit.
0	1	INTCXROM: If this bit is 1, the internal ROM at $Cx00 is selected.
	0	If this bit is 0, the peripheral-card ROM at Cx00 is selected.

Bank $01 (auxiliary memory)

◆ *Apple II note:* The following sections describe the operation of the auxiliary memory (bank $01) as it applies to programs originally written for the Apple IIc or for 128K versions of the Apple IIe. Programs written specifically for the Apple IIGS don't normally use bank $01 in this fashion, but use the Memory Manager to obtain whatever memory space they need, without specifying whether it is in bank $00, bank $01, or expansion memory in higher banks.

When display shadowing is on, some of the display modes use memory in auxiliary memory (bank $01). Specifically, half of the 80-column text display page is there, along with half of each of the Double Hi-Res graphics display pages and all of the Super Hi-Res display buffer, if those displays are shadowed. For descriptions and memory maps of those display pages, refer to Chapter 4. Apple II programs that use one of those display modes cannot use the corresponding pages of bank $01 for program and data storage.

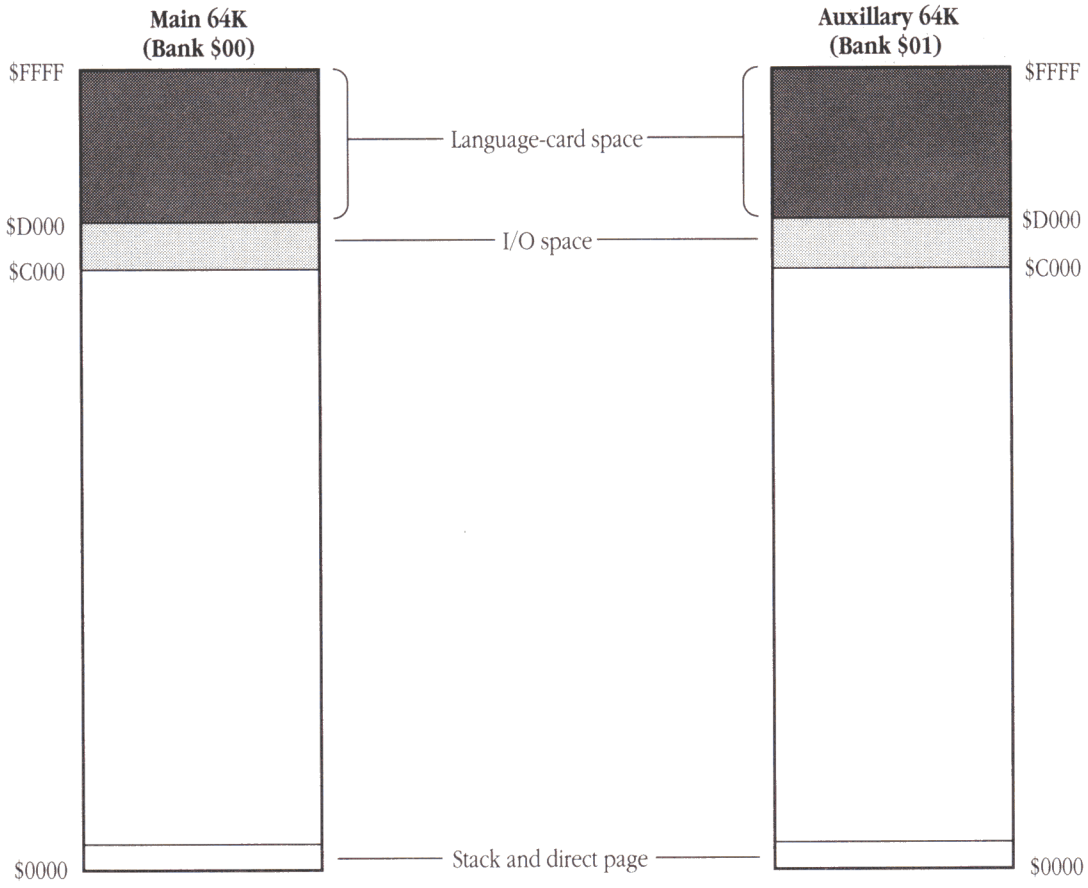
▲ **Warning** Do not attempt to switch in the auxiliary memory from a BASIC program. The BASIC interpreter uses several areas in main RAM, including the stack and the direct page. If you switch to alternate memory in these areas, the BASIC interpreter fails and you must reset the system and start over. ▲

As you can see by studying the memory map in Figure 3-6, the auxiliary memory is divided into two large sections and one small one. The largest section is switched into the memory address space from 512 to 49151 ($0200 through $BFFF). This space includes the display buffer pages: Space in auxiliary memory is used for one-half of the 80-column text display and the Double Hi-Res graphics display. You can switch to the auxiliary memory for this entire memory space, or you can switch just the display pages: See the section "Bank Switching for Auxiliary Memory," later in this chapter.

△ **Important** A program that uses auxiliary memory only for the 80-column display can write into the display page in auxiliary memory by using the SET80COL and TXTPAGE2 soft switches described in the section "Display Mode Switching" in Chapter 4. △

The other large section of auxiliary memory is the language-card space, which is switched into the memory address space from 52K to 64K ($D000 through $FFFF). This memory space and the switches that control it are described earlier in this chapter in the section "Language-Card Memory Space." The language-card soft switches have the same effect on the auxiliary RAM that they do on the main RAM: The language-card bank switching is independent of the auxiliary RAM switching.

■ **Figure 3-6** Memory map of main and auxiliary memory



- *Note:* The soft switches for the language-card memory, described in the previous section, do not change when you switch to auxiliary RAM. In particular, if ROM is enabled in the language-card space before you switch to auxiliary memory, the ROM will still be enabled after you switch. Any time you switch the language-card section of auxiliary memory in and out, you must also make sure that the bank switches are set properly.

When you switch in the auxiliary RAM in the language-card space, you also switch in the first two pages, from 0 to 511 ($0000 through $01FF). This part of memory contains the direct page and the 65C816 stack when running in 6502 emulation mode. The stack and direct page are switched this way so that standard Apple II **system software** running in the language-card space can maintain its own stack and direct page while it manipulates the 48K address space (from $0200 to $BFFF) in either main memory or auxiliary memory.

Bank switching for auxiliary memory

Switching the 48K section of memory is performed by two soft switches: The switches named RDMAINRAM and RDCARDRAM select main or auxiliary memory for reading, and the switches named WRMAINRAM and WRCARDRAM select main or auxiliary memory for writing. As shown in Table 3-3, there are two switches for each function: one to select main memory, and the other to select auxiliary memory. Enabling the read and write functions independently makes it possible for a program whose instructions are being fetched from one memory space to store data into the other memory space.

▲ **Warning** Do not use these switches without careful planning. Careless switching between main and auxiliary memories is almost certain to have catastrophic effects on the operation of your program. ▲

Writing to the soft switch at location $C003 turns RDCARDRAM on and enables auxiliary memory for reading; writing to location $C002 turns RDMAINRAM on and enables main memory for reading. Writing to the soft switch at location $C005 turns WRCARDRAM on and enables the auxiliary memory for writing; writing to location $C004 turns WRMAINRAM on and enables main memory for writing. By setting these switches independently, you can use any of the four combinations of reading and writing in main or auxiliary memory.

You can use auxiliary memory corresponding to text Page 1 and Hi-Res graphics Page 1 as part of the address space from $0200 to $BFFF by using RAM read and RAM write soft switches as described above. You can also control these areas in auxiliary RAM separately by using the switches named SET80COL, TXTPAGE2, and HIRES.

■ **Table 3-3** Auxiliary-memory select switches

Name	Function	Location Hex	Location Dec	Notes
RDCARDRAM	Read auxiliary memory	$C003	49155	Write
RDMAINRAM	Read main memory	$C002	49154	Write
RDRAMRD	Read switch status	$C013	49171	Read and test bit 7 (1=auxiliary, 0=main)
WRCARDRAM	Write auxiliary memory	$C005	49157	Write
WRMAINRAM	Write main memory	$C004	49156	Write
RDRAMWRT	Read switch status	$C014	49172	Read and test bit 7 (1=auxiliary, 0=main)
SET80COL	Access display page	$C001	49153	Write
CLR80COL	Use RAM switches ($C002-5,13,14)	$C000	49152	Write
RD80COL	Read switch status	$C018	49176	Read and test bit 7 (1=80-column access on, 0=80-column access off)
TXTPAGE2	Text Page 2 on (auxiliary memory)*	$C055	49237	Read or write
TXTPAGE1	Text Page 1 on (main memory)*	$C054	49236	Read or write
RDPAGE2	Read switch status	$C01C	49180	Read and test bit 7 (1=Page 2, 0=Page 1)
HIRES	Access Hi-Res pages†	$C057	49239	Read or write
LORES	Use RAM switches ($C002-5,13,14)†	$C056	49238	Read or write
RDHIRES	Read switch status	$C01D	49181	Read and test bit 7 (1=HIRES on, 0=off)
SETALTZP	Auxiliary stack and direct page	$C009	49161	Write
SETSTDZP	Main stack and direct page	$C008	49160	Write
RDALTZP	Read switch status	$C016	49174	Read and test bit 7 (1=auxiliary, 0=main)

* When SET80COL is enabled, TXTPAGE2 and TXTPAGE1 select main or auxiliary display memory.

† When SET80COL is enabled, HIRES and LORES enable you to use TXTPAGE2 and TXTPAGE1 to switch between the Hi-Res Page 1 area in main memory or auxiliary memory.

As shown in Table 3-3, the SET80COL switch functions as an enabling switch: With it on, you can select main memory or auxiliary memory by writing to either TXTPAGE1 or TXTPAGE2. With the HIRES switch off, the memory space switched by TXTPAGE2 is text Page 1 ($0400 to $07FF); with HIRES on, TXTPAGE2 switches both text Page 1 and Hi-Res graphics Page 1 ($2000 to $3FFF).

If you are using both the auxiliary RAM control switches (SET80COL, CLR80COL, TXTPAGE1, TXTPAGE2, and HIRES) and the auxiliary display page control switches (RDMAINRAM, RDCARDRAM, WRMAINRAM, and WRCARDRAM), the display page control switches take priority. That is, if CLR80COL is on, the RAM read and write switches toggle the entire auxiliary and main memory space from $0200 to $BFFF.

If SET80COL is on, the RAM switches have no effect on the display page; if SET80COL is on and LORES is on, the TXTPAGE1 and TXTPAGE2 switches control text Page 1, regardless of the settings of the RAM read and write switches. Likewise, if SET80COL and HIRES are both on, TXTPAGE1 and TXTPAGE2 control both text Page 1 and Hi-Res graphics Page 1, again regardless of the RAM read and RAM write switches.

A single soft switch named ALTZP (for alternate zero page) switches the bank-switched memory and the associated stack and direct-page area between main and auxiliary memory. As shown in Table 3-3, writing to location $C009 turns ALTZP on and selects auxiliary memory stack and direct page; writing to the soft switch at location $C008 turns ALTZP off and selects main memory stack and direct page for both reading and writing.

There is a third soft switch (RDRAMRD and RDRAMWRT) associated with each pair of auxiliary memory switches listed above. The **high-order** bits of the byte you read at this location tell you the setting of the associated soft switches. For example, the byte you read at location $C013 has its high bit set to 1 if the auxiliary memory is read-enabled, or to 0 if the 48K block of main memory is read-enabled.

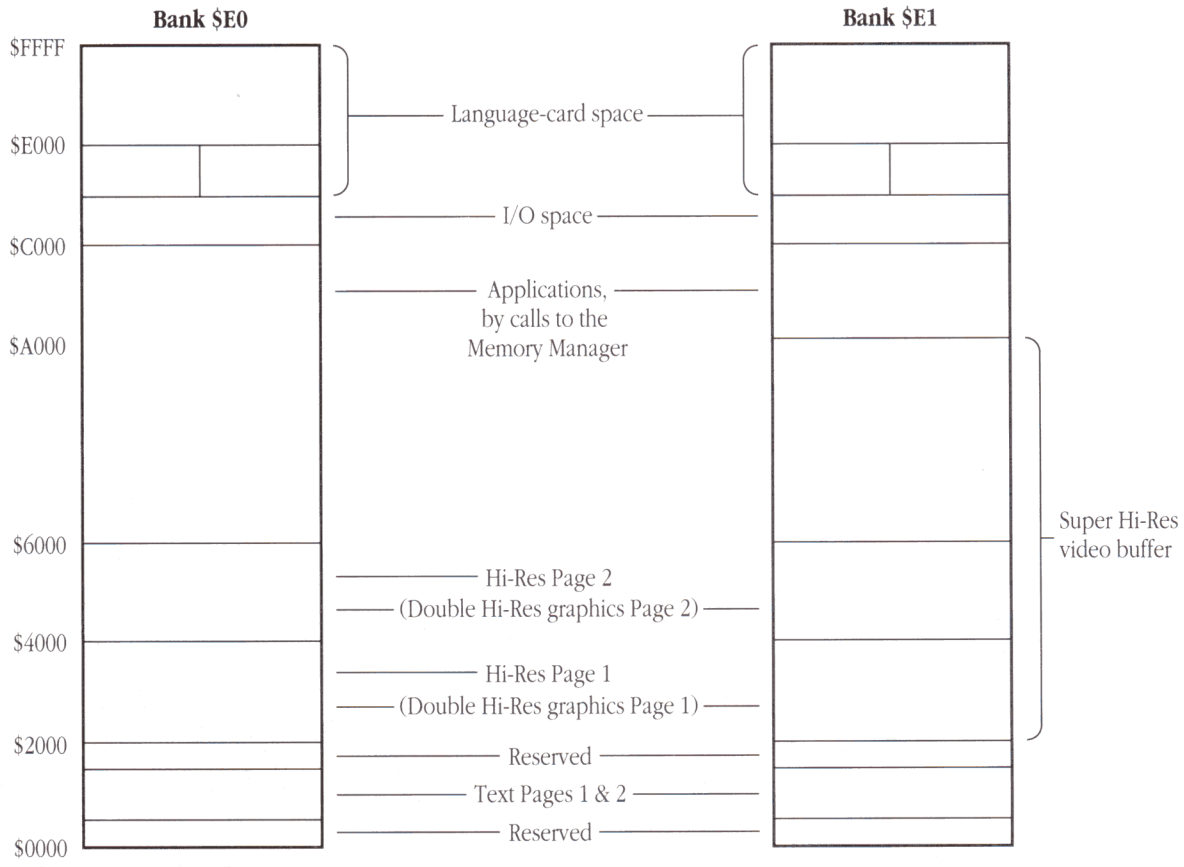
- *Sharing memory:* In order to have enough memory locations for all the soft switches and to remain compatible with the Apple II and Apple II Plus, the soft switches listed in Table 3-3 share their memory locations with the keyboard functions listed in Table 6-2. The read or other operations shown in Table 3-3 for controlling the auxiliary memory are just the ones that are not used for reading the keyboard and clearing the strobe.

Banks $E0 and $E1

Banks $E0 and $E1 are the memory banks controlled by the Mega II. These banks have special characteristics that make it appropriate to use them in special ways. First, they contain the display buffers, so they have to run at the standard 1.024-MHz speed. Second, because these banks are broken up by the special allocations shown in Figure 3-7, they are the logical place for working storage used by the toolbox and other firmware programs. Using banks $E0 and $E1 for such purposes leaves all the higher-speed memory in the low-numbered banks for application programs and data.

- *Note:* In banks $E0 and $E1, language-card mapping, I/O space, and display buffers are always active.

Figure 3-7 Memory map of banks $E0 and $E1



The display buffers

The display buffers are permanently assigned to locations in banks $E0 and $E1 because they are tied directly to the hardware—the Mega II IC—that reads the data stored there and generates the display signals. Display-memory shadowing makes it possible to run old-style Apple II programs that store display data in banks $00 and $01, as described earlier in this chapter in the section "Shadowed Display Spaces," and in Chapter 2.

The primary text and Lo-Res graphics display buffers occupy memory locations $0400 through $07FF in banks $E0 and $E1. The 1024-byte area in bank $E0 is text Page 1, the display buffer for 40-column text mode. The 80-column text display uses text Page 1 locations in both bank $E0 and bank $E1. The Control Panel and other firmware programs use these display buffers, so applications must not use them for program or data storage.

Text Page 2, the alternate text and Lo-Res graphics display buffer, occupies memory locations $0800 through $0BFF. Most programs do not use text Page 2 for displays, and the original Apple IIGS doesn't shadow it. The 1 MB Apple IIGS has a bit in the Shadow register that allows you to enable shadowing of text Page 2.

There are two Hi-Res graphics buffers, each of which requires 8192 bytes (8K) of memory. Hi-Res graphics Page 1 occupies memory locations $2000 through $3FFF in bank $E0. Hi-Res graphics Page 2 occupies memory locations $4000 through $5FFF in bank $E0.

The Double Hi-Res graphics buffers require a total of 16384 bytes each, 8192 in each bank. Double Hi-Res graphics Page 1 occupies memory locations $2000 through $3FFF in banks $E0 and $E1. Double Hi-Res graphics Page 2 occupies memory locations $4000 through $5FFF in banks $E0 and $E1.

The Super Hi-Res graphics display buffer, with its associated palette and scan-line control data storage, occupies 32K of memory at locations $2000 through $9FFF in bank $E1. Note that, unlike the other Hi-Res graphics displays, it doesn't use any space in bank $E0.

In principle, programs that don't use specific displays can use the corresponding display buffers for data storage. In practice, programs call on the Memory Manager to allocate space for data storage, and the Memory Manager keeps track of which display spaces are available. If you try to load your programs and data directly into display memory in banks $E0 and $E1, you risk interference from **desk accessories** or other programs that may be resident in memory along with your programs.

Firmware workspace

As described in the previous section, banks $E0 and $E1 are always broken up by display buffers, so they are the logical place for working storage used by the toolbox and other system programs. Figure 3-7 shows the memory areas reserved for those programs.

Several different system programs use RAM space in banks $E0 and $E1, including

- the Monitor program
- desk accessories
- the Memory Manager
- the Tool Locator
- the Apple Desktop Bus **tool set**
- the AppleTalk driver

In addition, portions of banks $E0 and $E1 are reserved for future use. For specific uses of these programs and utilities, refer to the *Apple IIGS Firmware Reference*.

▲ **Warning** Several of the built-in programs use RAM areas in banks $E0 and $E1. To avoid conflicts with those programs, applications must not use these areas; instead, applications should request memory from the Memory Manager. Refer to the *Apple IIGS Toolbox Reference* for information about the Memory Manager. ▲

Apple II program memory use

Earlier models of the Apple II use a microprocessor, the 6502, that can address only 65536 bytes (64K) of memory. The Apple IIe and the Apple IIc double this, to 128K, by switching to an auxiliary 64K bank. In order for programs written for these machines to run on an Apple IIGS, all of the Apple II features must be present in the part of memory that corresponds to main and auxiliary memory—banks $00 and $01. This section describes those features.

Banks $00 and $01

For standard Apple II programs, banks $00 and $01 take on the features of the main and auxiliary banks. With the 65C816 microprocessor running in emulation mode and shadowing set appropriately, all of the standard features are present, including

- direct (zero) page, from $0000 to $00FF of bank $00
- stack, from $0100 to $01FF of bank $00
- text Page 1, from $0400 to $07FF of both banks
- text Page 2, from $0800 to $0BFF of both banks (available in the 1 MB Apple IIGS only)
- Hi-Res graphics Page 1, from $2000 to $3FFF of bank $00
- Hi-Res graphics Page 2, from $4000 to $5FFF of bank $00
- Double Hi-Res graphics Page 1, from $2000 to $3FFF of both banks
- Double Hi-Res graphics Page 2, from $4000 to $5FFF of both banks
- I/O space, from $C000 to $CFFF of either bank
- language-card space, from $D000 to $FFFF of both banks

Figure 3-3 is a memory map of banks $00 and $01 showing these features.

Shadowing

The display buffers in the Apple IIGS are located in banks $E0 and $E1, as described earlier in this chapter. For compatibility with standard Apple II programs, shadowing must be switched on for the display buffers needed by those programs. For more information about shadowing, refer to Chapter 2 and to the section "Shadowed Display Spaces" earlier in this chapter.

Screen holes

When shadowing is on for text Page 1, programs and peripheral cards that use the text Page 1 locations known as the *screen holes* run normally. For more information about the screen holes, refer to the section "Peripheral-Card RAM Space" in Chapter 8.

Memory expansion

The original Apple IIGS has 256K of RAM and 128K of ROM built in, and the 1 MB Apple IIGS has 1 MB RAM and 256K of ROM. This memory can be expanded to a total of 5 MB of RAM (4 MB RAM on a memory expansion card), and 1 MB total ROM (768K ROM on a memory expansion card). Memory expansion up to 8 MB of RAM is possible by using the memory expansion slot, but complications requiring memory support logic keep this expansion from being practical. The hardware and firmware in the Apple IIGS are designed to support only a 5 MB maximum memory space. Addresses above 8 MB are not available to applications programs.

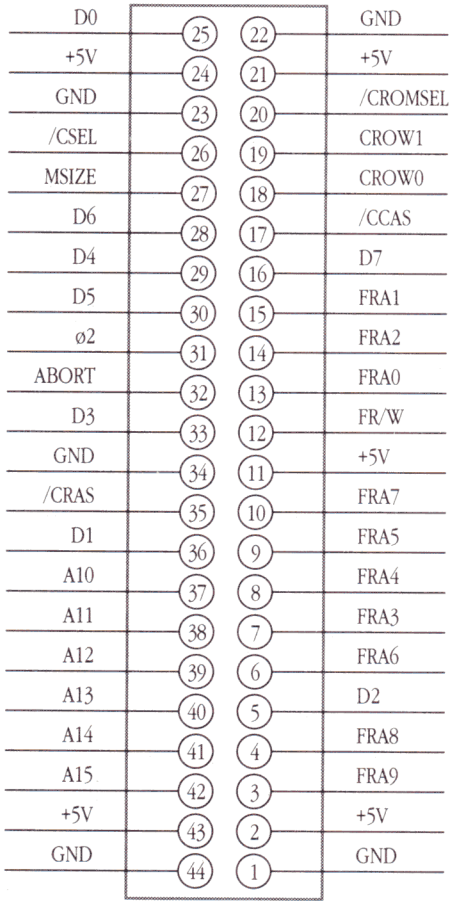
The memory expansion slot

The extended memory-card slot enables you to expand the memory of your Apple IIGS by adding a memory card holding up to 4 MB of RAM and 786K of ROM memory. The slot supports additional memory only and is not to be used for any other purpose. RAM cards of 1 MB or 4 MB can be constructed by using 256K rows or 1 MB rows of RAM ICs.

Memory expansion signals

The memory expansion slot provides a group of signals to support dynamic RAM and additional general purpose signals to support ROM decoding and selection.
Figure 3-8 shows these signals available at the pins of the memory expansion slot. Table 3-4 describes each of the signals.

■ **Figure 3-8** Memory expansion slot



■ **Table 3-4** Memory-card interface signals

Pin	Signal	Description
	FRA0–9	10 bits of multiplexed RAM address for RAM cycles—the 10 least significant bits of the ROM address
12	FR/W	Write enable to RAMs; R/W from microprocessor or DMA
17	/CCAS	RAM column address strobe
18–19	CROW0–1	2 bits select one of four RAM rows
20	/CROMSEL	Card ROM select; low for accesses to banks $F0–$FD
26	/CSEL	Card data buffer direction control; signal goes high when reading card data
27	MSIZE	Output from card; indicates RAM row size
	D0–D7	8 bits of bidirectional data—microprocessor data bus
31	ø2CLK	Microprocessor clock; rising edge indicates valid bank address on D0–D7
	A10–15	The 6 high-order address bits; used to decode ROM address
32	ABORT	Connects to 65C816 ABORT pin
35	/CRAS	RAM row address strobe
	+5V	+5 volts ± 5 percent; 600 mA maximum

Extended RAM

Using dynamic RAM ICs available when the Apple IIGS was introduced, up to 4 MB of RAM can be installed in the extended memory card. That much memory corresponds to 64 banks of 64K each. The memory on the card is organized as 4 rows of 8 ICs each. With 256-kilobit–by–1-bit RAMs, each row would hold 256K for a total of 1 MB; with 1 megabit by 1 bit RAMs, each row would hold 1 MB for a total of 4 MB.

While the memory expansion slot has sufficient address lines available to decode addresses up to 8 MB, memory expansion cards of greater than 4 MB are not recommended. This is because memory expansion locations beyond 4 MB cannot be accessed via direct memory access (DMA), and also must provide on-board memory refresh support circuitry for the additional memory chips.

To control and select individual rows of RAM, the FPI provides /CRAS (card row address strobe), /CCAS (card column address strobe), CROW0 (card row select 0), and CROW1 (card row select 1) signals. Signals /CRAS and /CCAS are the basic memory timing signals

common to most dynamic RAMs. Signals CROW0 and CROW1 are row selects that, when taken as a pair, indicate the row number to be accessed. Typically, CROW0 and CROW1 are used as the select signals for a dual 1-of-4 decoder (74F139 or equivalent) that demultiplexes /CRAS and /CCAS into a separate /RAS and /CAS for each 8-chip segment.

Extended RAM mapping

Figure 3-9 depicts a 1 MB extended RAM card using four rows of 256K per row, totaling 1 MB. The RAM banks above bank $11 are *ghosts* (repeated **images**) of the RAM in banks $2 through $11. A partially populated card causes holes in the memory map unless there is an option on the card to alter the address decoding. Therefore, contiguous memory for banks $2 through $11 is available only for 256K, 512K, and 1 MB expansion cards used with the 256K Apple IIGS. Memory expansion cards using 256K, 512K, 768K, and 1024K may be used with the 1 MB system.

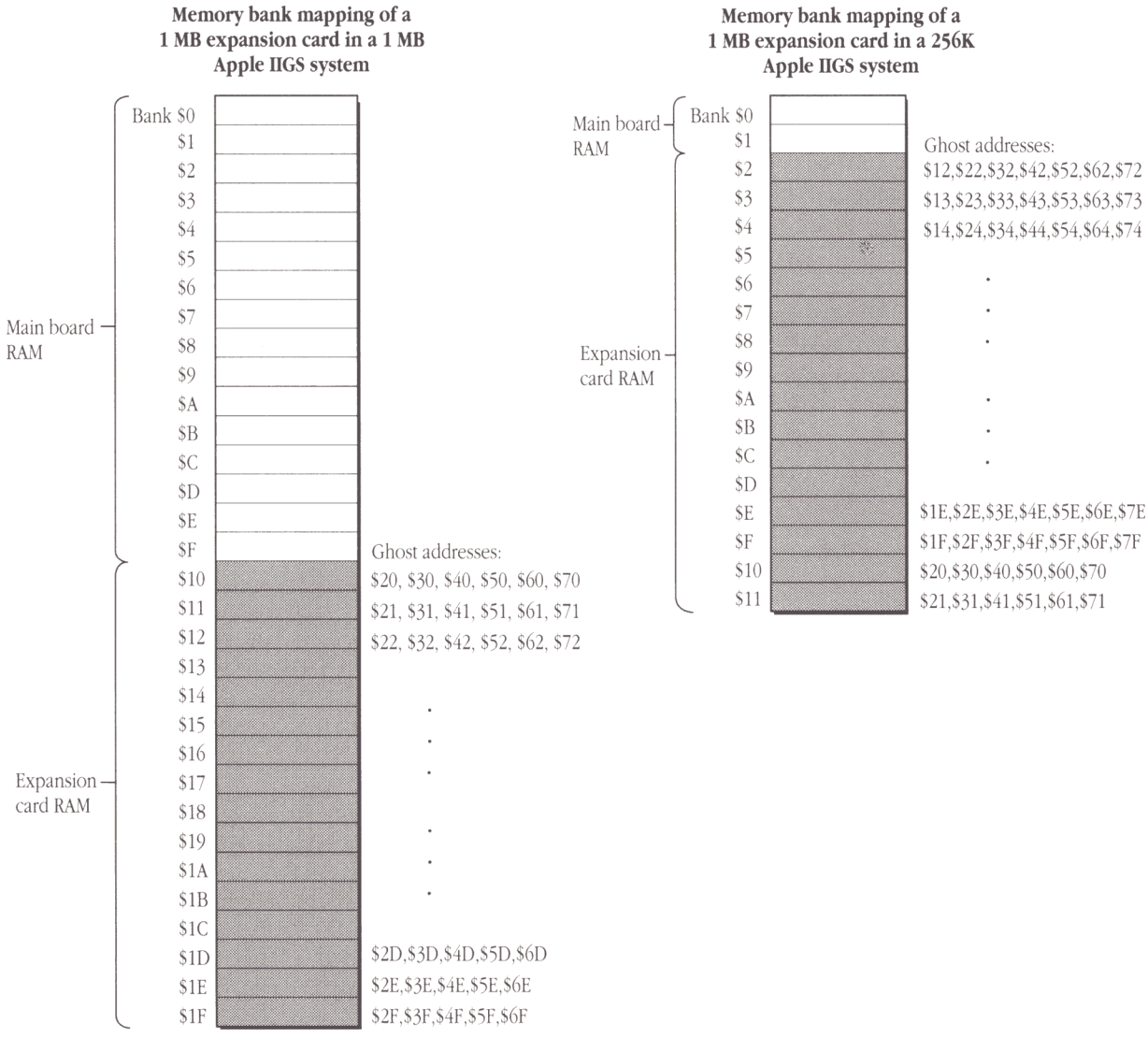
The MSIZE signal: A signal on the memory expansion slot, MSIZE, flags the type of memory chips being used on the memory expansion card. If the MSIZE pin is not connected (when using 256-kilobit RAMs), the FPI multiplexes 18 address bits onto RA0 through RA8 and generates the CROW0 through CROW1 row selects for rows of 256K. If the MSIZE pin is tied to ground (for 1-megabit RAMs), the FPI multiplexes 20 address bits onto RA0 through RA9 and generates the CROW0 and CROW1 row selects for rows of 1 MB.

Ghost addresses: A 1 MB expansion card is enabled for accesses in banks $2 through $80 in the 256K system, and banks $10 through $80 in the 1 MB system. The card provides only 1 MB of actual RAM (banks $2 through $11 in the 256K system, and banks $10 through $19 in the 1 MB system). CROW 0 and CROW1 individually select four rows of RAMs on the card. For a 1 MB card with 256K rows (MSIZE =1), the selected RAM row number is given by the bank number modulo 4. For banks $00 and $01 (banks $00 through $0F in the 1 MB Apple IIGS system), the extended memory card is not accessed. This method of card and row selection causes multiple images or *ghosts* of the RAM areas on the card; whenever accessing addresses beyond the RAM expansion-card limit (hexadecimal address $FFFFF with a 1 MB card, and $3FFFFF with a 4 MB card), locations in a corresponding low bank are accessed.

Extended ROM

Additional ROM space (up to 896K in the 256K Apple IIGS, and up to 786K in the 1MB Apple IIGS) is available in banks $F0 through $FD ($F0 through $FB on the 1MB logic board). To obtain this additional space, an additional bank-address latch-decoder is required on the memory card. The FPI provides a signal (CROMSEL) that selects one bank; however, the card must provide the additional decoding to select individual ROMs within the selected bank.

■ **Figure 3-9** Extended RAM mapping



Address multiplexing

The FPI multiplexes the RAM addresses onto either eight, nine, or ten RAM address lines to provide support for RAM with 64-kilobit, 256-kilobit, or 1-megabit RAM ICs. On the 256K Apple IIGS, the main logic board RAMs (banks $00 and $01) are 64-kilobit chips, requiring eight address lines. On the 1 MB Apple IIGS, the main logic board RAMs (banks $00 through $0F) are 1-megabit chips requiring ten address lines. The RAM expansion slot can support cards using 256-kilobit–by–1-bit, 256-kilobit–by–4-bit, 1-megabit–by–1-bit, or 1-megabit–by–4-bit RAMs. The expansion-card manufacturer indicates **word** size of the RAMs on the memory card by the MSIZE signal from the card. (See "The MSIZE Signal," earlier in this chapter.)

Chapter 4 **The Video Displays**

The Apple IIGS can display several video modes. These include display modes that are compatible with the rest of the Apple II family (but with some enhancements to these existing modes) and some completely new display modes. These new video modes provide higher resolution, greater color flexibility, and greater programming ease than was previously available in the Apple II product line. This chapter describes

- enhancements to the standard Apple II video modes
- new video features including the new video display modes

Apple IIGS display features

The Apple IIGS brings new features to the existing Apple II video modes. These include

- selectable screen border color
- selectable background color
- selectable text color
- selectable color or black-and-white composite video

These enhancements are described in this chapter. The new graphics modes—Super Hi-Res graphics and Color Fill mode—are also described in this chapter.

Video from the Mega II IC

The Mega II generates all video information in standard Apple II video modes. The Mega II outputs a 4-bit linear-weighted **binary** code, which represents one of the 16 possible standard Apple II colors. This **digital** value is input into the Video Graphics Controller (VGC) and is used as a look-up address for an equivalent 12-bit resolution RGB color output value.

A digital-to-analog converter changes the 12-bit color code into three analog RGB video signals. The RGB output signals drive the video amplifiers and the NTSC video generator chip. The output of the video amplifier boosts the the RGB signals, while the NTSC chip mixes the RGB and **sync signals,** resulting in a composite video signal.

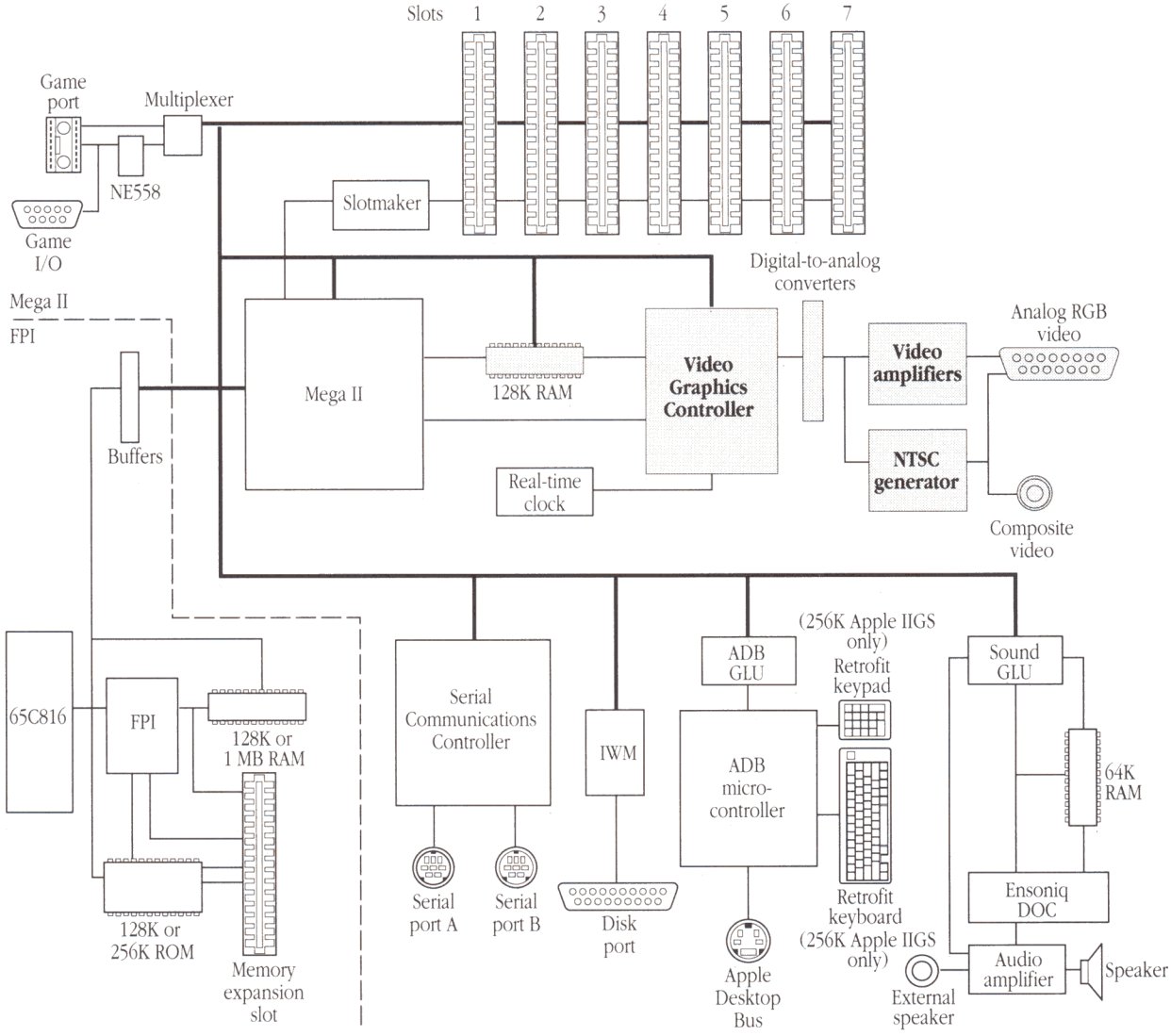
The Video Graphics Controller

The Video Graphics Controller (VGC) custom IC is responsible for generating all video output. The VGC provides these functions:

- takes standard Apple II video information from the Mega II and generates the video output
- adds enhancements to existing Apple II video modes
- supports the new video modes
- provides interrupt handling for two interrupt sources

The VGC generates all video output in all video modes, whereas the Mega II is responsible for maintaining the video RAM. All write operations to the video display buffers in bank $E0 and bank $E1 are done via the Mega II. Figure 4-1 shows the relationships of the VGC, Mega II, main RAM, and auxiliary RAM.

■ **Figure 4-1** Video components in the Apple IIGS



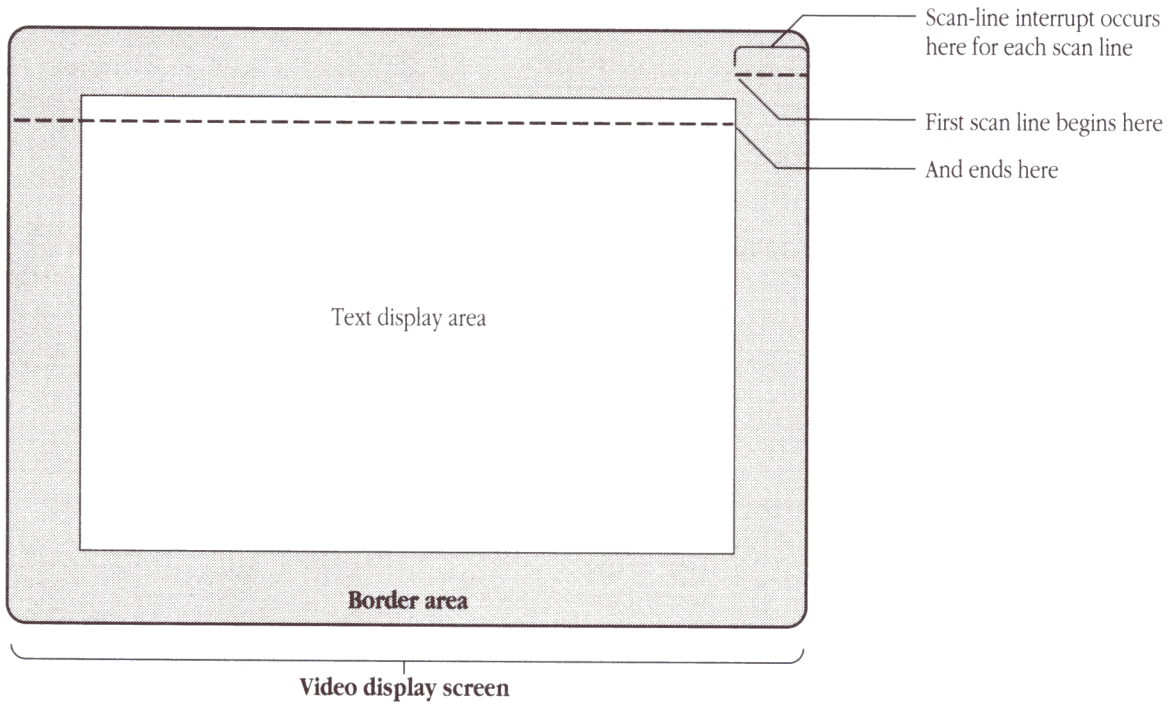
VGC interrupts

Video display in the Apple IIGS is enhanced by VGC-generated interrupts. The VGC generates two internal interrupts: the one-second interrupt and the scan-line interrupt.

A 1-Hz input signal from the real-time clock (RTC) chip sets the one-second interrupt status bit. The scan-line interrupt occurs at the beginning of a video display scan line that has the generate-interrupt bit set in the corresponding scan-line control byte. Scan-line interrupts are generated when the computer is operating in the Super Hi-Res video graphics modes only, and are not available in other video modes.

Figure 4-2 depicts the video screen, consisting of the text display area and the display border. The scan-line interrupt occurs at the beginning of the scan line, which is defined as the beginning of the right-hand border area.

■ **Figure 4-2** Scan-line interrupt



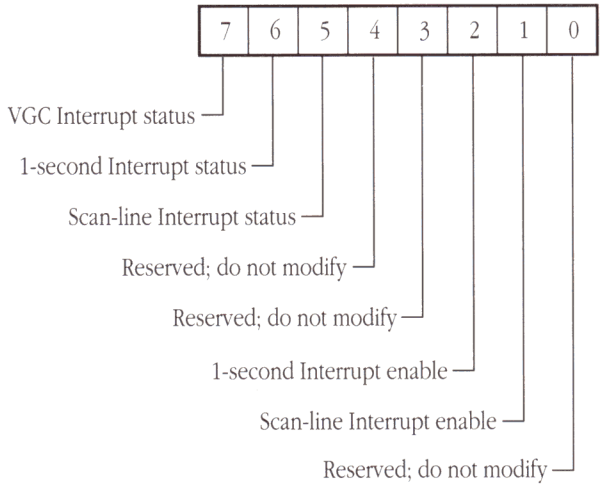
The VGC Interrupt register

The VGC Interrupt register ($C023) contains a status bit and an enable bit for each of the two interrupts. When an interrupt occurs, the interrupt status bit for that interrupt is set. The VGC interrupt bit (bit 7) is set and the interrupt request (IRQ) line is asserted if the interrupt status bit *and* interrupt enable bit are set for one or more interrupts.

You enable an interrupt by writing to the appropriate positions in the VGC Interrupt register; the interrupt source hardware sets the status bits. Software can directly manipulate only the enable bits in the VGC Interrupt register; writing to the other bit positions has no effect. Figure 4-3 shows the format of the VGC Interrupt register. Table 4-1 gives a description of each register bit.

▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 4-3** VGC Interrupt register at $C023



■ **Table 4-1** Bits in the VGC Interrupt register

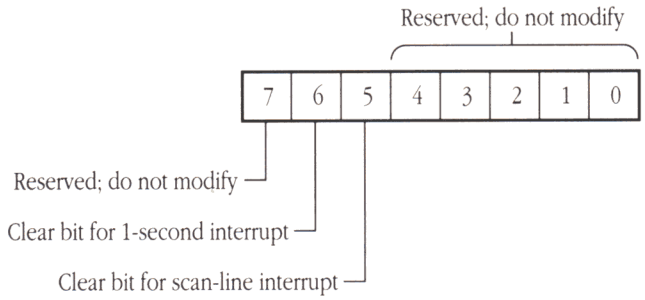
Bit	Value	Description
7	1	VGC interrupt status: This bit is set when the interrupt bit and the status bit are set for one or more of the interrupts.
	0	This bit is 0 when all interrupts have been cleared.
6	1	One-second interrupt status: 1 = interrupt has occurred.
	0	0 = interrupt is cleared.
5	1	A scan-line interrupt status: 1 = interrupt has occurred.
	0	0 = interrupt is cleared.
4–3	–	Reserved; do not modify.
2	1	One-second interrupt is enabled.
	0	Interrupt is disabled.
1	1	Scan-line interrupt is enabled.
	0	Interrupt is disabled.
0	–	Reserved; do not modify.

The VGC Interrupt-Clear register

Once an interrupt has occurred, the interrupt routine must proceed to clear the interrupt and take some predetermined interrupt-handling action. To clear the scan-line and one-second status bits, write a 0 into the corresponding bit position in the VGC Interrupt-Clear register at $C032. Bit 5 clears the scan-line interrupt, and bit 6 clears the one-second interrupt in the VGC Interrupt-Clear register. Writing a 1 into these positions or writing into the other bit positions has no effect. Figure 4-4 shows the format of the VGC Interrupt-Clear register. Table 4-2 gives a description of each bit.

▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 4-4** VGC Interrupt-Clear register at $C032



■ **Table 4-2** Bits in the VGC Interrupt-Clear register

Bit	Value	Description
7	–	Reserved; do not modify.
6	1	Undefined result.
	0	Write a 0 here to clear the one-second interrupt.
5	1	Undefined result.
	0	Write a 0 here to clear the scan-line interrupt.
4–0	–	Reserved; do not modify.

Video outputs

The Apple IIGS shares several display modes with previous Apple II computers. The Apple IIGS supports and enhances these existing Apple II video modes:

- 40-column and 80-column text modes
- mixed text/graphics mode
- Lo-Res graphics mode
- Hi-Res graphics mode
- Double Hi-Res graphics mode

Enhancements to the existing Apple II video modes include the following:

- The ability to select unique text and background colors from the list in Table 4-3.
- The ability to select the border color for the perimeter of the video image on an RGB monitor; you can choose this color from the list in Table 4-3.
- The ability to display gray-scale video; you can display color video output on **monochrome** monitors in shades of gray rather than in dot patterns that represent color, which increases contrast between graphics colors on a monochrome monitor.

- **Table 4-3** Text and background colors

Color value	Color	Color value	Color
$0	Black	$8	Brown
$1	Deep red	$9	Orange
$2	Dark blue	$A	Light gray
$3	Purple	$B	Pink
$4	Dark green	$C	Green
$5	Dark gray	$D	Yellow
$6	Medium blue	$E	Aquamarine
$7	Light blue	$F	White

Removing color from the composite video signal in 40-column and 80-column text modes makes text more readable. Color is not removed when the computer is running in mixed text/graphics modes, and the four lines of text at the bottom of the display will exhibit **color fringing** on composite color monitors.

Apple II video

All Apple II computers can display video in several different ways, displaying text as well as color graphics. The standard Apple II text and graphics modes are discussed here, while the new Super Hi-Res graphics modes are discussed later in this chapter.

The primary output device is the video display. You can use any ordinary video monitor, either color or black-and-white, to display video information from the Apple IIGS. An ordinary monitor is one that accepts composite video compatible with the standard set by the National Television Standards Committee (NTSC). If you use standard Apple II color graphics with a monochrome (single-color) monitor, the display will appear as that color (black, for example) and various patterns made up of shades of that color.

If you are using only 40-column text and Lo-Res graphics modes, you can use a television set for your video display. If the television set has an input connector for composite video, you can connect it directly to your computer; if it does not, you'll need to attach a **radio-frequency (RF)** video modulator between the Apple IIGS and the television set.

- *Note:* The Apple IIGS can produce an 80-column text display. However, if you use an ordinary color or black-and-white television set, 80-column text will be too blurry to read. For a clear 80-column display, you must use a high-resolution video monitor with a bandwidth of 7 MHz or greater.

The specifications for the video display are summarized in Table 4-4.

The video signal produced by the Apple IIGS is NTSC-compatible composite color video. It is available at two places: at the RCA-type phono jack and at the RGB video connector, both on the back of the computer. Use the RCA-type phono jack to connect a composite video monitor or an external video modulator; use the RGB video connector to connect an analog-input RGB monitor.

The Apple IIGS can also display Super Hi-Res graphics, although it is not a standard Apple II video display mode. Super Hi-Res graphics are discussed more fully later in this chapter.

■ **Table 4-4** Standard Apple II video display specifications

Display modes	40-column text; map: Figure 4-5. 80-column text; map: Figure 4-6. Lo-Res color graphics; map: Figure 4-7. Hi-Res color graphics; map: Figure 4-8. Double Hi-Res color graphics; map: Figure 4-9.
Text capacity	24 lines by 80 columns (character positions).
Character set	128 **ASCII** characters. (See Appendix C for a list of display characters.)
Display formats	Normal, inverse, flashing, MouseText (Table 4-10).
Lo-Res color graphics	16 colors (Table 4-14): 40 horizontal by 48 vertical; map: Figure 4-7.
Hi-Res color graphics	6 colors (Table 4-15): 140 horizontal by 192 vertical (restricted). Black-and-white: 280 horizontal by 192 vertical; map: Figure 4-8.
Double Hi-Res color graphics	16 colors (Table 4-16): 140 horizontal by 192 vertical (no restrictions). Black-and-white: 560 horizontal by 192 vertical; map: Figure 4-9.

The 40-column and 80-column text modes can display all 128 ASCII (American Standard Code for Information Interchange) characters: uppercase and lowercase letters, numbers, and symbols. (See the display maps in Figures 4-5 and 4-6.) The Apple IIGS can also display MouseText characters.

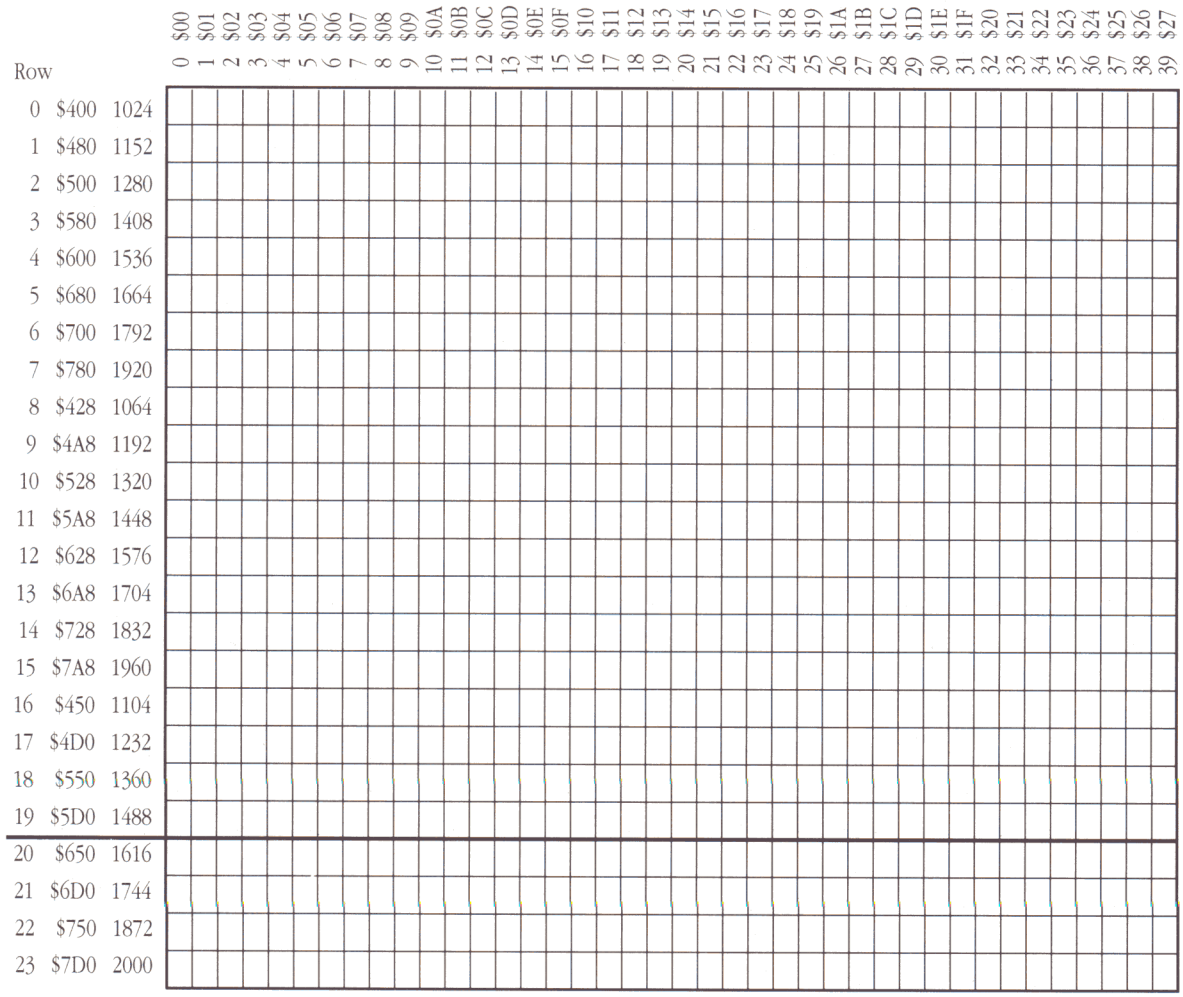
Any of the graphics displays can have four lines of text at the bottom of the screen. The text may be either 40-column or 80-column, except that Double Hi-Res graphics may have only 80-column text at the bottom of the screen. Graphics displays with text at the bottom are called *mixed-mode displays.*

The Lo-Res graphics display is an array of colored blocks, 40 wide by 48 high, in any of 16 colors. (See the map in Figure 4-7.) In mixed mode, 4 lines of text replace the bottom 8 rows of blocks, leaving 40 rows of 40 blocks each.

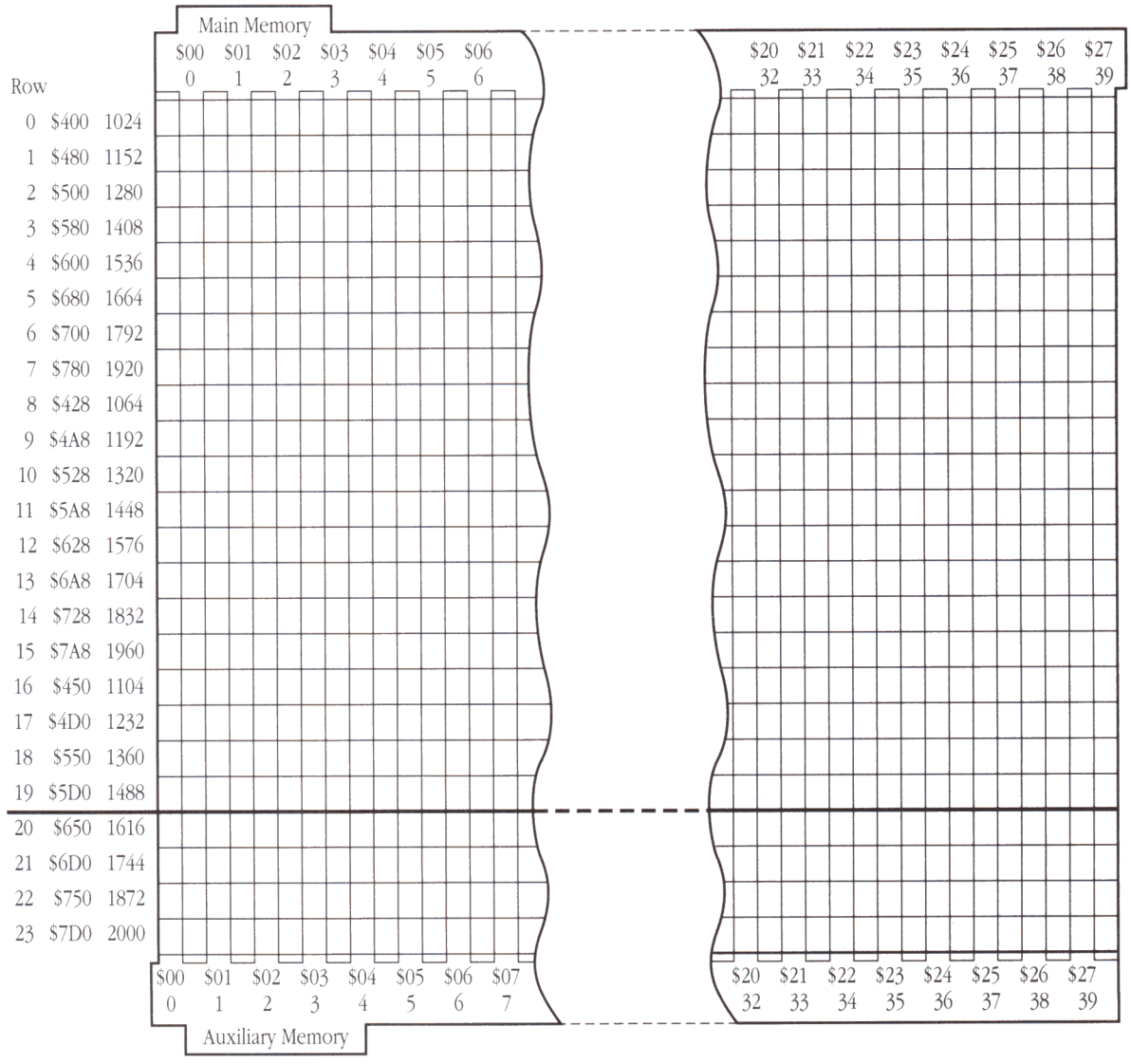
The Hi-Res graphics display is an array of pixels, 280 wide by 192 high. (See the map in Figure 4-8.) There are six colors available in Hi-Res displays, but a given pixel can use only four of the six colors. If color is used, the display is 140 pixels wide by 192 high. If monochrome video is desired, the display is 280 pixels wide by 192 high. In mixed mode, the four lines of text replace the bottom 32 rows of pixels, leaving 160 rows of 140 (or 280) pixels each.

The Double Hi-Res graphics display uses main and auxiliary memory to display an array of pixels, 560 wide by 192 high. (See the map in Figure 4-9.) All the pixels are visible in black and white. If color is used, the display is 140 pixels wide by 192 high with 16 colors available. If monochrome video is desired, the display is 560 pixels wide by 192 high. In mixed mode, the four lines of text replace the bottom 32 rows of pixels, leaving 160 rows of 140 (or 560) pixels each. In mixed mode, the text lines can be 80 columns wide only.

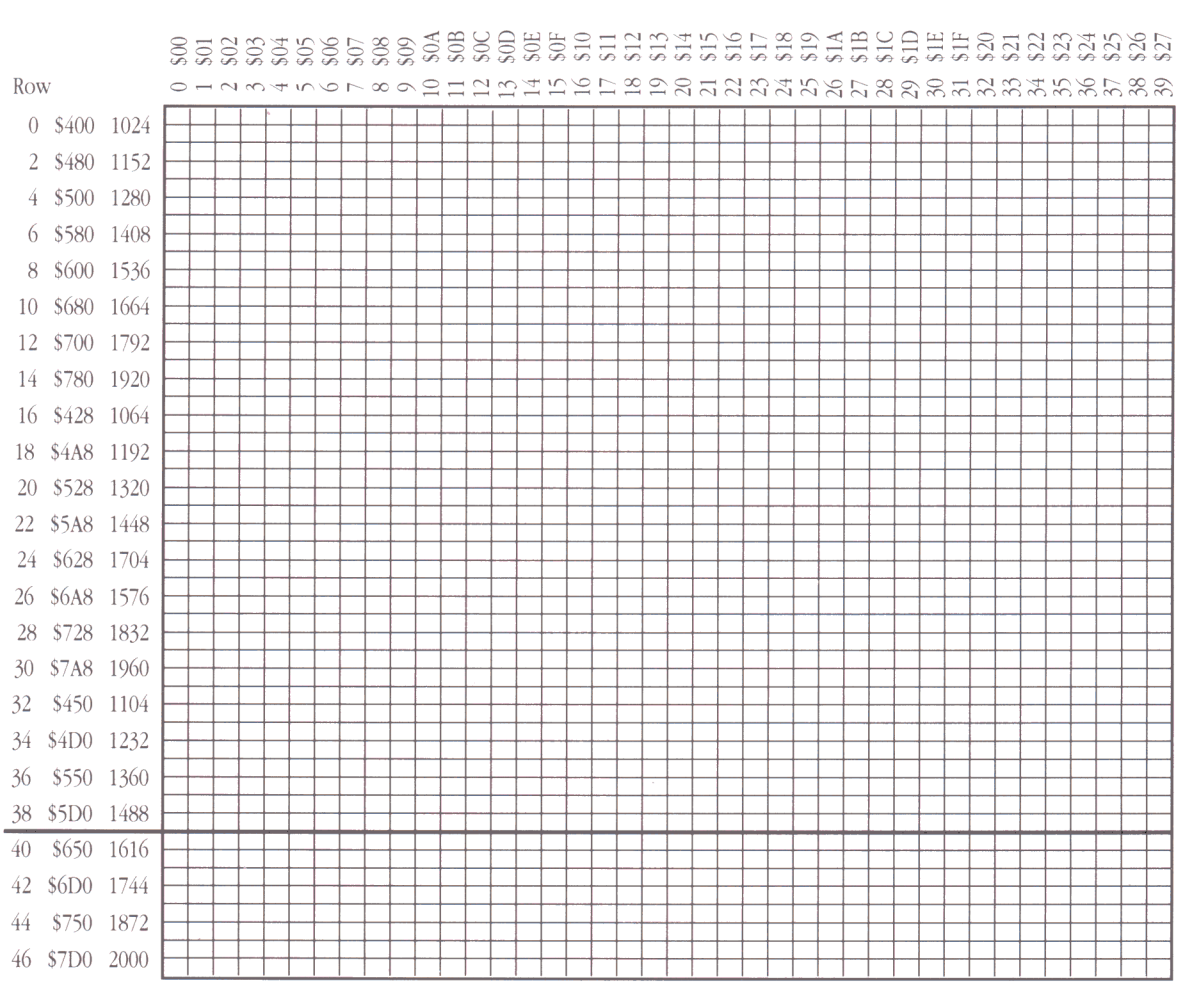
■ **Figure 4-5** Map of 40-column text Page 1 display (Add 1024 [$400] to get Page 2 addresses.)



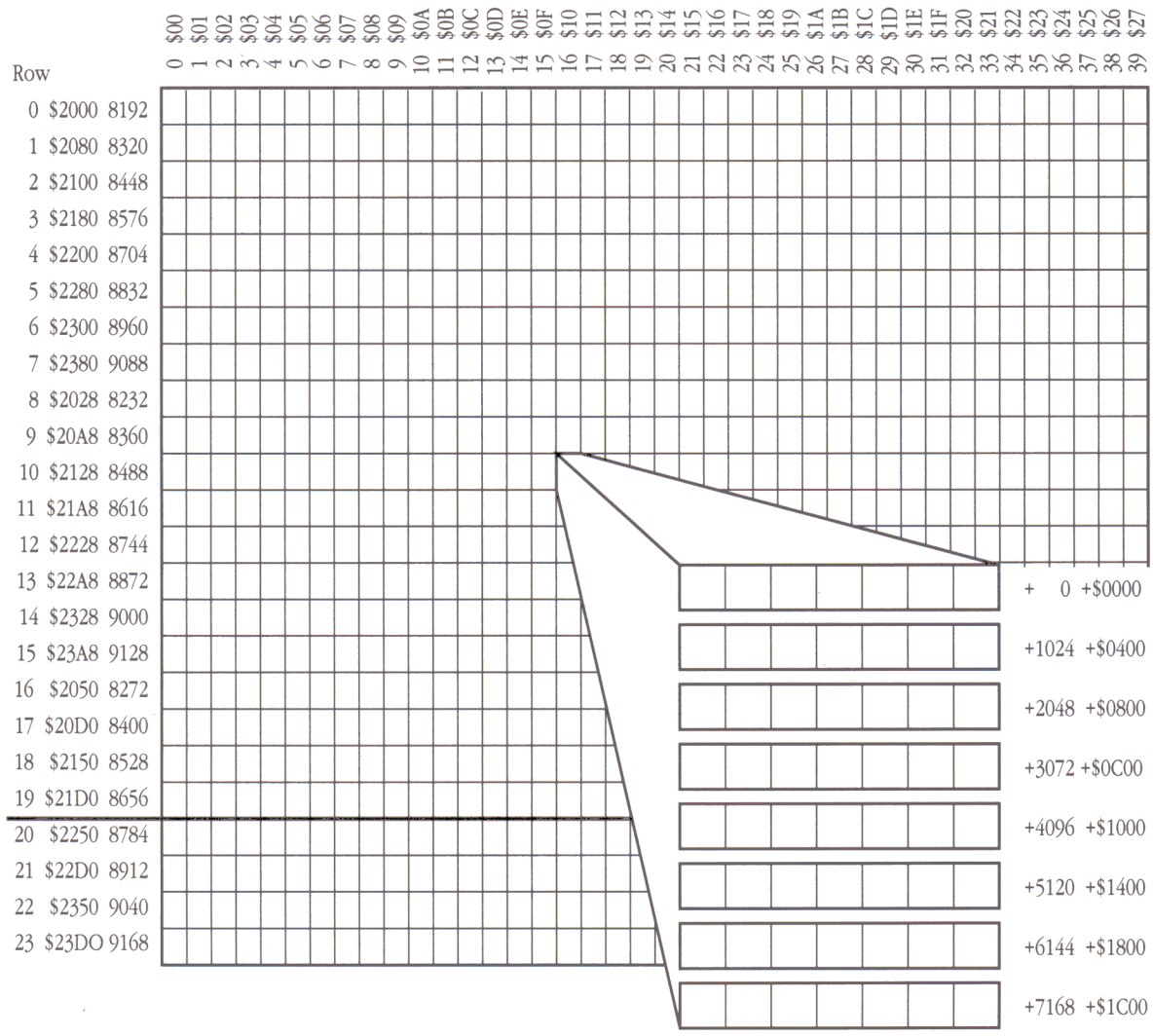
■ **Figure 4-6** Map of 80-column text display



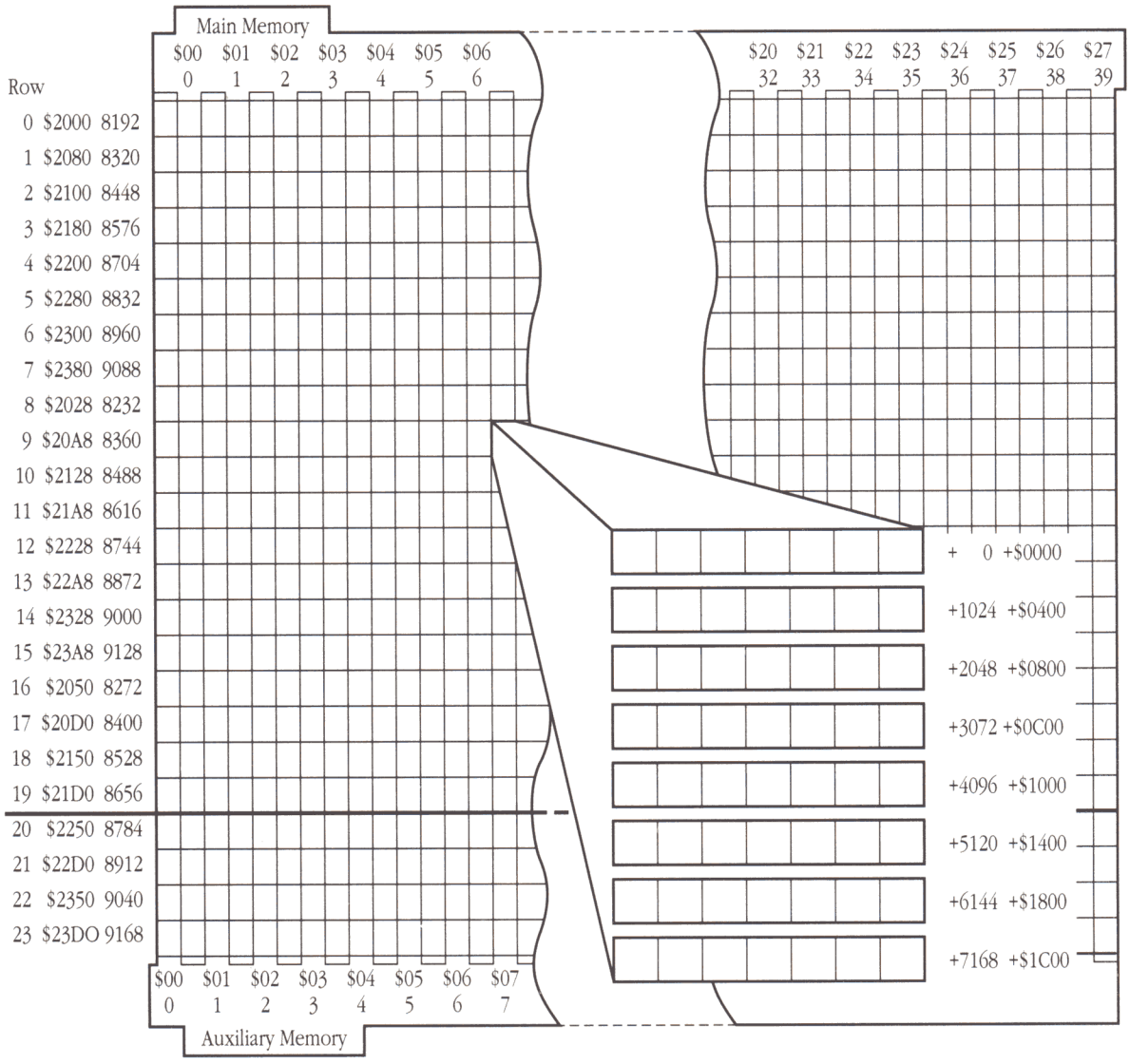
■ **Figure 4-7** Map of Lo-Res graphics Page 1 display (Add 1024 [$400] to get Page 2 addresses.)



■ **Figure 4-8** Map of Hi-Res graphics Page 1 display (Add 8192 [$2000] to get Page 2 addresses.)



■ **Figure 4-9** Map of Double Hi-Res graphics display

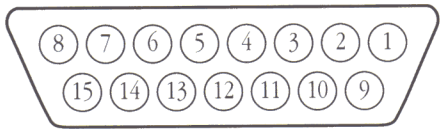


NTSC versus RGB video

The composite video signal, available at the composite video connector at the rear of the Apple IIGS case, will drive a standard NTSC composite color video monitor.

The RGB video signals are three separate color signals, which individually control the three colors (red, green, and blue) within an RGB color video monitor. The RGB video connector is located at the rear of the computer. Connect only an RGB video monitor with analog inputs to this connector. Figure 4-10 shows the pin diagram of this connector, and Table 4-5 lists the signal associated with each pin.

■ **Figure 4-10** RGB video connector



■ **Table 4-5** RGB video signals

Pin	Signal	Description
1	GND	Ground reference and supply
2	RED	Red analog video signal
3	COMP	Composite sync signal
4	N.C.	No connection
5	GREEN	Green analog video signal
6	GND	Ground reference and supply
7	–5V	–5-volt supply
8	+12V	+12-volt supply
9	BLUE	Blue analog video signal
10	N.C.	No connection
11	SOUND	Analog sound output
12	NTSC/**PAL**	Composite video output
13	GND	Ground reference and supply
14	N.C.	No connection

Video display pages

The Apple IIGS generates its video displays by using data stored in specific areas in memory. These areas, called display pages, serve as buffers where your programs can put data to be displayed. Each byte in a display buffer controls an object at a certain location on the display. In text mode, the object is a single character; in Lo-Res graphics mode, the object is two stacked colored blocks; and in Hi-Res and Double Hi-Res modes, it is a line of seven adjacent pixels.

The 40-column text and Lo-Res graphics modes use two display pages of 1024 bytes each. These are called text Page 1 and text Page 2, and they are located at 1024 through 2047 ($0400 through $07FF) and 2048 through 3071 ($0800 through $0BFF) in main memory. Normally, only text Page 1 is used, but you can put text or graphics data into text Page 2 and switch displays instantly. Either page can be displayed as 40-column text, Lo-Res graphics, or mixed mode (four rows of text at the bottom of a graphics display).

The 80-column text mode displays twice as much data as the 40-column mode—1920 bytes—but it cannot switch pages. The 80-column text display uses a combination page made up of text Page 1 in main memory plus another page in auxiliary memory. This additional memory is not the same as text Page 2—in fact, it occupies the same address space as text Page 1, and there is a special soft switch that enables you to store data into it. (See the next section, "Display Mode Switching.") The built-in firmware I/O routines, described in the *Apple IIGS Firmware Reference*, take care of this extra addressing automatically; that is one reason to use these routines for all your normal text output.

The Hi-Res graphics mode also has two display pages, but each page is 8192 bytes long. In the 40-column text and Lo-Res graphics modes, each byte controls a display area 7 pixels wide by 8 pixels high. In Hi-Res graphics mode each byte controls an area 7 pixels wide by 1 pixel high. Thus, a Hi-Res display requires 8 times as much data storage, as shown in Table 4-6.

The Double Hi-Res graphics mode uses Hi-Res graphics Page 1 in both main and auxiliary memory. Each byte in those pages of memory controls a display area 7 pixels wide by 1 pixel high. This gives you 560 pixels per line in black and white, and 140 pixels per line in color. A Double Hi-Res display requires twice the total memory of Hi-Res graphics, and 16 times as much as a Lo-Res display.

■ **Table 4-6** Video display locations

Display mode	Display page	Lowest address Hex	Lowest address Dec	Highest address Hex	Highest address Dec
40-column text, Lo-Res graphics	1	$0400	1024	$07FF	2047
	2*	$0800	2048	$0BFF	3071
80-column text	1	$0400	1024	$07FF	2047
	2*	$0800	2048	$0BFF	3071
Hi-Res graphics	1	$2000	8192	$3FFF	16383
	2	$4000	16384	$5FFF	24575
Double High-Res graphics	1†	$2000	8192	$3FFF	16383
	2†	$4000	16384	$5FFF	24575

* Lo-Res graphics on Page 2 is not supported by firmware; for instructions on how to switch pages, refer to the next section, "Display Mode Switching."

† See the section "Double Hi-Res Graphics" later in this chapter.

Display mode switching

You select the display mode that is appropriate for your application by reading or writing to a reserved memory location called a soft switch. In the Apple IIGS, most soft switches have three memory locations reserved for them: one for turning the switch on, one for turning it off, and one for reading the current state of the switch.

Table 4-7 shows the reserved locations for the soft switches that control the display modes. For example, to switch from mixed mode to full-screen graphics in an assembly-language program, you could use the instruction

```
STA   $C052
```

To do this in a BASIC program, you could use the instruction

```
POKE  49234,0
```

Some of the soft switches in Table 4-7 must be read, some must be written to, and for some you can use either action. When writing to a soft switch, it doesn't matter what value you write; the action occurs when you address the location, and the value is ignored.

■ **Table 4-7** Display soft switches

Name	Action*	Location	Function
CLR80COL	W	$C000 (49152)	Disable 80-column store.
SET80COL	W	$C001 (49153)	Enable 80-column store.
CLR80VID	W	$C00C (49164)	Disable 80-column hardware.
SET80VID	W	$C00D (49165)	Enable 80-column hardware.
CLRALTCHAR	W	$C00E (49166)	Normal lowercase character set; flashing uppercase character set.
SETALTCHAR	W	$C00F (49167)	Normal, inverse character set; no flashing.
RD80COL	R7	$C018 (49176)	Read CLR/SET80COL switch: 1 = 80-column store enabled.
RDVBL BAR	R7	$C019 (49177)	Read vertical blanking **(VBL):** 1 = not VBL.
RDTEXT	R7	$C01A (49178)	Read TXTCLR/TXTSET switch: 1 = text mode enabled.
RDMIX	R7	$C01B (49179)	Read MIXCLR/MIXSET switch: 1 = mixed mode enabled.
RDPAGE2	R7	$C01C (49180)	Read TXTPAGE1/TXTPAGE2 switch: 1 = text Page 2 selected.
RDHIRES	R7	$C01D (49181)	Read HIRES switch: 1 = Hi-Res mode enabled.
ALTCHARSET	R7	$C01E (49182)	Read CLRALTCHAR/SETALTCHAR switch: 1 = alternate character set in use.
RD80VID	R7	$C01F (49183)	Read CLR80VID/SET80VID switch: 1 = 80-column hardware in use.
RDDHIRES	R5	$C046 (49222)	Read SETAN3/CLRAN3 switch: 0 = Double Hi-Res graphics mode selected.

(Continued)

■ **Table 4-7** Display soft switches (Continued)

Name	Action*	Location	Function
TXTCLR	R/W	$C050 (49232)	Select standard Apple II graphics mode, or, if MIXSET on, mixed mode.
TXTSET	R/W	$C051 (49233)	Select text mode only.
MIXCLR	R/W	$C052 (49234)	Clear mixed mode.
MIXSET	R/W	$C053 (49235)	Select mixed mode.
TXTPAGE1	R/W	$C054 (49236)	Select text Page 1.
TXTPAGE2	R/W	$C055 (49237)	Select text Page 2, or, if SET80COL on, text Page 1 in auxiliary memory.
LORES	R/W	$C056 (49238)	Select Lo-Res graphics mode.
HIRES	R/W	$C057 (49239)	Select Hi-Res graphics mode, or, if SETAN3 is on, select Double Hi-Res graphics mode.
CLRAN3	R/W	$C05E (49246)	See Table 4-8.
SETAN3	R/W	$C05F (49247)	See Table 4-8.

* *W* means write anything to the location, *R* means read the location, *R/W* means read or write, *R7* means read the location and then check bit 7, and *R5* means read the location and then check bit 5.

◆ *Note:* You may not need to deal with these functions by reading and writing directly to the memory locations in Table 4-7. Many of the functions shown here are selected automatically if you use the display routines in the various **high-level languages** on the Apple IIGS.

Any time you read a soft switch, you get a byte of data. However, the only information the byte contains is the state of the switch, and this occupies only one bit—bit 7, the high-order bit. The other bits in the byte are always 0.

If you read a soft switch from a BASIC program, you get a value between 0 and 255. Bit 7 has a value of 128, so if the switch is on, the value will be equal to or greater than 128; if the switch is off, the value will be less than 128.

Mixing address modes

It is possible to display combinations of modes on the video display. The combination can be any mode of graphics combined with either 40-column or 80-column text, graphics only, or text-only modes. Table 4-8 lists the possible combinations, and the state of the soft switches to achieve the display modes.

■ **Table 4-8** Video display mode combinations

New-Video reg. ($C029) bit 5	AN3 ($C046)	TEXT ($C01C)	HIRES ($C01D)	80COL ($C018)	Video mode
-	-	1	-	0	40-column text
-	-	1	-	1	80-column text
-	1	0	0	0	Lo-Res graphics and 40-column text
-	1	0	0	1	Lo-Res graphics and 80-column text
-	0	0	0	1	Medium-Res (80-column) graphics
-	1	0	1	0	Hi-Res graphics and 40-column text
-	1	0	1	1	Hi-Res graphics and 80-column text
0	0	0	1	1	Double-Hi-Res, 16-color
1	0	0	1	1	Double-Hi-Res, black-and-white

Addressing display pages directly

Before you decide to use the display pages directly, consider the alternatives. Most high-level languages enable you to write statements that control the text and graphics displays. Similarly, if you are programming in assembly language, you may be able to use the display features of the built-in I/O firmware. You should store directly into display memory only if the existing programs can't meet your requirements.

The display memory maps are shown in Figures 4-5, 4-6, 4-7, 4-8, and 4-9. All the different display modes use the same basic addressing scheme: Characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes ($80 hexadecimal). By folding the display data into memory this way, the Apple IIGS stores all 960 characters of displayed text within 1K of memory.

The Hi-Res graphics display is stored in much the same way as text, but there are eight times as many bytes to store, because eight rows of pixels occupy the same space on the display as one row of characters. The subset consisting of all the first rows from the groups of eight is stored in the first 1024 bytes of the Hi-Res display page. The subset consisting of all the second rows from the groups of eight is stored in the second 1024 bytes, and so on for a total of eight times 1024, or 8192 bytes. In other words, each 1024 bytes of Hi-Res video memory contains one row of pixels from every group of eight rows. The individual rows are stored in sets of three 40-byte rows, the same as the text display.

All of the display modes except 80-column text mode and Double Hi-Res and Super Hi-Res graphics modes can use either of two display pages. The display maps show addresses for each mode's Page 1 only. To obtain addresses for text or Lo-Res graphics Page 2, add 1024 ($400) to the Page 1 addresses; to obtain addresses for Hi-Res graphics Page 2, add 8192 ($2000) to the Page 1 addresses.

The 80-column text display and Double Hi-Res graphics modes work a little differently. Half of the data are stored in the normal text Page 1 main memory, and the other half are stored in auxiliary memory using the same addresses as for text Page 1. The display circuitry fetches bytes from these two memory areas simultaneously and displays them sequentially: first the byte from the auxiliary memory, then the byte from the main memory. The main memory stores the characters in the odd columns of the display, and the auxiliary memory stores the characters in the even columns.

To store display data in the 80-column text display, first turn on the SET80COL soft switch by writing to location $C001. With SET80COL on, the page-select switch, TXTPAGE2, selects between the portion of the 80-column display memory in Page 1 of main memory and the portion stored in the 80-column text display memory. To enable the 80-column text display, turn the TXTPAGE2 soft switch on by reading or writing at location $C055.

The text window

After you have started up the computer or after a reset, the firmware uses the entire video display. However, you can restrict video activity to any rectangular portion of the display you wish. The active portion of the display is called the **text window.** You can set the top, bottom, left side, and width of the text window by storing the appropriate values into four locations in memory. Using these memory locations allows you to control the placement of text in the display and to protect other portions of the screen from being written over by new text.

Memory location $20 contains the number of the leftmost column in the text window. This number is normally 0, the number of the leftmost column in the display. In a 40-column display, the maximum value for this number is $27; in an 80-column display, the maximum value is $4F.

Memory location $21 holds the width of the text window. For a 40-column display, it is normally $28; for an 80-column display, it is normally $50.

▲ **Warning** Be careful not to let the sum of the window width and the leftmost position in the window exceed the width of the display you are using (40 or 80). If this happens, it is possible to put characters into memory locations outside the display page, which might destroy programs or data. ▲

Memory location $22 contains the number of the top line of the text window. This is normally 0, the topmost line in the display. Its maximum value is $17.

Memory location $23 contains the number of the bottom line of the screen, plus 1. It is normally $18 for the bottom line of the display. Its minimum value is $01.

After you have changed the text window boundaries, nothing is affected until you send a character to the screen.

▲ **Warning** Any time you change the boundaries of the text window, you should make sure that the current cursor horizontal position (CH, stored at $24) and cursor vertical position (CV, stored at $25) are within the new window values. If they are outside, it is possible to put characters into memory locations outside the display page, which might destroy programs or data. ▲

Table 4-9 summarizes the memory locations and the possible values for the window parameters.

■ **Table 4-9** Text window memory locations

Window parameter	Location		Minimum value		Normal values				Maximum values			
					40-column		80-column		40-column		80-column	
	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
Left edge	32	$20	00	$00	00	$00	00	$00	39	$27	79	$4F
Width	33	$21	00	$00	40	$28	80	$50	40	$28	80	$50
Top edge	34	$22	00	$00	00	$00	00	$00	23	$17	23	$17
Bottom edge	35	$23	01	$01	24	$18	24	$18	24	$18	24	$18

Text displays

The Apple IIGS, like all standard Apple II computers, can display text in two ways: 40 columns wide by 24 rows, or 80 columns wide by 24 rows. Many character sets are available, including standard alphanumeric characters, special characters, and MouseText characters.

Text on the Apple IIGS can also be displayed in color: The text, background, and border each can be a different color. The following sections give details about the text displays.

Text modes

The text characters displayed include the uppercase and lowercase letters, the ten numerical digits, punctuation marks, and special characters. Each character is displayed in an area of the screen that is seven pixels wide by eight pixels high. The characters are formed by a pixel matrix five pixels wide, leaving two blank columns of pixels between characters in a row, except for MouseText characters, some of which are seven pixels wide. Except for lowercase letters with descenders and some MouseText characters, the characters are only seven pixels high, leaving one blank line of pixels between rows of characters.

The normal display has white pixels on a medium blue background. (Other color text on other color backgrounds is also possible, as described later in this chapter.) Characters can also be displayed in inverse format with blue pixels on a white background.

Text character sets

The Apple IIGS can display either of two selected text character sets: the primary set or an alternate set. The forms of the characters in the two sets are actually the same, but the available display formats are different. The display formats are

- normal
- inverse
- flashing, alternating between normal and inverse

With the primary character set, the Apple IIGS can display uppercase and special characters in all three formats: normal, inverse, and flashing. Lowercase letters can be displayed in normal format only. The primary character set is compatible with most software written for other Apple II models, which can display text in flashing format but which don't have lowercase characters.

The alternate character set displays characters in either normal or inverse format. In normal format, you can get

- uppercase letters
- lowercase letters
- numbers
- special characters

In inverse format, you can get

- MouseText characters
- uppercase letters
- lowercase letters
- numbers
- special characters

You select the character sets by means of the alternate-text soft switch, SETALTCHAR, described earlier in this chapter in the section "Display Mode Switching." Table 4-10 shows the character codes in hexadecimal for the primary and alternate character sets in normal, inverse, and flashing formats.

Each character on the screen is stored as one byte of display data. The low-order six bits make up the ASCII code of the character being displayed. The remaining two (high-order) bits select inverse or flashing format and uppercase or lowercase characters. In the primary character set, bit 7 selects inverse or normal format and bit 6 controls character flashing. In the alternate character set, bit 6 selects between uppercase and lowercase, according to the ASCII character codes, and flashing format is not available.

■ **Table 4-10** Display character sets

Hex values	Primary character set		Alternate character set	
	Character type	Format	Character type	Format
$00–$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
$20–$3F	Special characters	Inverse	Special characters	Inverse
$40–$5F	Uppercase letters	Flashing	MouseText	Inverse
$60–$7F	Special characters	Flashing	Lowercase letters	Inverse
$80–$9F	Uppercase letters	Normal	Uppercase letters	Normal
$A0–$BF	Special characters	Normal	Special characters	Normal
$C0–$DF	Uppercase letters	Normal	Uppercase letters	Normal
$E0–$FF	Lowercase letters	Normal	Lowercase letters	Normal

40-column versus 80-column text: The Apple IIGS has two modes of text display: 40-column and 80-column. The number of pixels in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare Figures 4-11 and 4-12. On an ordinary color or black-and-white television set, the narrow characters in the 80-column display blur together; you must use the 40-column mode to display text on a television set.

■ **Figure 4-11** 40-column text display

```
/UTILITIES

 NAME                TYPE   BLOCKS  MODIFIED
       CREATED                 ENDFILE SUBTYPE

*STARTUP              BAS         3   31-JUL-85
 0:00   <NO DATE>                1005
*SI                   BAS         3   31-JUL-85
 0:00   <NO DATE>                1005
*SU2C                 BAS        38   31-JUL-85
 0:00   <NO DATE>               18886
*SU2E                 BAS        34   31-JUL-85
 0:00   <NO DATE>               16465
*SU1.OBJ              BIN        31   31-JUL-85
 0:00   <NO DATE>               15211 A=$3200
*SU2.OBJ              BIN         9   31-JUL-85
 0:00   <NO DATE>                3696 A=$2000
*SU3.OBJ              BIN        62   31-JUL-85
 0:00   <NO DATE>               31152 A=$0E00
*SU4.OBJ              VAR        18   31-JUL-85
 0:00   <NO DATE>                8535
*SU5.OBJ              BIN         1   31-JUL-85
 0:00   <NO DATE>                  95 A=$86AC
*SU6.OBJ              VAR        15   31-JUL-85
 0:00   <NO DATE>                6848
*PRODOS               SYS        30   18-SEP-84
 0:00   <NO DATE>               14848
*BASIC.SYSTEM         SYS        21   18-JUN-84
 0:00   <NO DATE>               10240

BLOCKS FREE: 1328      BLOCKS USED:   272
     TOTAL BLOCKS: 1600
```

■ **Figure 4-12** 80-column text display

```
/UTILITIES

 NAME           TYPE  BLOCKS  MODIFIED         CREATED          ENDFILE SUBTYPE

*STARTUP         BAS       3  31-JUL-85  0:00  <NO DATE>           1005
*SI              BAS       3  31-JUL-85  0:00  <NO DATE>           1005
*SU2C            BAS      38  31-JUL-85  0:00  <NO DATE>          18886
*SU2E            BAS      34  31-JUL-85  0:00  <NO DATE>          16465
*SU1.OBJ         BIN      31  31-JUL-85  0:00  <NO DATE>          15211 A=$3200
*SU2.OBJ         BIN       9  31-JUL-85  0:00  <NO DATE>           3696 A=$2000
*SU3.OBJ         BIN      62  31-JUL-85  0:00  <NO DATE>          31152 A=$0E00
*SU4.OBJ         VAR      18  31-JUL-85  0:00  <NO DATE>           8535
*SU5.OBJ         BIN       1  31-JUL-85  0:00  <NO DATE>             95 A=$86AC
*SU6.OBJ         VAR      15  31-JUL-85  0:00  <NO DATE>           6848
*PRODOS          SYS      30  18-SEP-84  0:00  <NO DATE>          14848
*BASIC.SYSTEM    SYS      21  18-JUN-84  0:00  <NO DATE>          10240

BLOCKS FREE: 1328     BLOCKS USED:  272     TOTAL BLOCKS: 1600
```

Color text

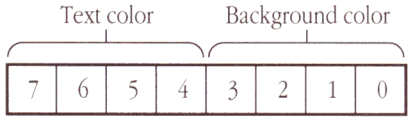
New to the Apple IIGS is the ability to display the text, background, and border in color. These colors may be set manually through the Control Panel, or under program control, via **control registers.**

Text and background color

The Apple IIGS provides the capability of colored text on a colored background on an RGB monitor. To select colors for text and background, write the appropriate color values to the Screen Color register located at $C022.

The Screen Color register is an 8-bit dual-function register. First, the most significant 4 bits determine the text color. Second, the least significant 4 bits determine the background color. You can choose these colors from the 16 available Apple II colors given in Table 4-3. The user can also select these colors from the Control Panel. Figure 4-13 shows the format of the Screen Color register. Table 4-11 gives a description of each bit in the register.

■ **Figure 4-13** Screen Color register at $C022



■ **Table 4-11** Bits in the Screen Color register

Bit	Value	Description
7–4	–	Text color
3–0	–	Background color

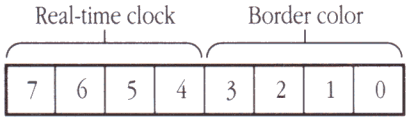
Border color

The colored border area surrounds the video display text area. You may select a color for the border by writing the appropriate color value to the Border Color register located at $C034. You can choose this color from the 16 Apple II colors listed in Table 4-3. Alternately, the user can select the border color from the Control Panel.

The Border Color register is an 8-bit read/write register serving two functions. First, the least significant 4 bits determine the border color. Second, the most significant 4 bits are the control bits for the real-time clock chip interface logic. See the section on the real-time clock interface in Chapter 7, "Built-in I/O Ports and Clock," for more information on the RTC. Figure 4-14 shows the Border Color register format. Table 4-12 gives a description of each bit.

▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 4-14** Border Color register at $C034



■ **Table 4-12** Bits in the Border Color register

Bit	Value	Description
7–4	–	Real-time clock control bits; do not modify bits 7–4 when changing bits 3–0
3–0	–	Border color

Monochrome/Color register

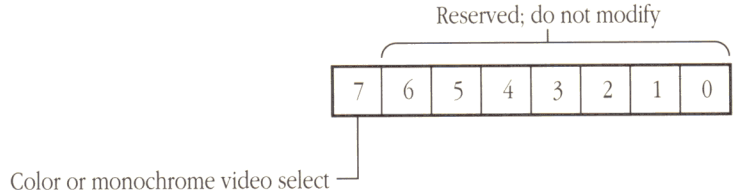
The Apple IIGS video is displayed in either color or black-and-white. Located at $C021, the Monochrome/Color register controls whether the composite video signal consists of color or gradations of gray. If bit 7 is a 1, video displays in black-and-white; if it is a 0, video displays in color.

If you are using a monochrome monitor, set bit 7 to 1. Displaying text in black-and-white results in a better-looking, more readable display. In text mode, all color information is removed from the composite video signal, resulting in a monochrome text display. The exception to this is the mixed text and graphics mode, which results in color text and color fringing.

The remaining bits in the Monochrome/Color register are reserved; do not modify them when writing to this location. You can also select color or monochrome video from the Control Panel. Figure 4-15 shows the format of the Monochrome/Color register. Table 4-13 gives a description of each bit in the register.

▲ **Warning** Be careful when changing bit 7 in this register. Use only a read-modify-write instruction sequence when manipulating bit 7. See the warning in the preface. ▲

■ **Figure 4-15** Monochrome/Color register at $C021



■ **Table 4-13** Bits in the Monochrome/Color register

Bit	Value	Description
7*	1	Composite gray-scale video output
	0	Composite color video output
6–0	–	Reserved; do not modify

* Changing bit 7 does not affect the RGB outputs.

◆ *Note:* Reading the Monochrome/Color register returns a meaningless value. Bit 7, therefore, can be referred to as write-only.

Graphics displays

The Apple IIGS can produce standard Apple II video graphics in three different modes, as well as two new graphics resolutions. All the graphics modes treat the screen as a rectangular array of spots. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes in these arrays; this section describes the way the resulting graphics data are stored in memory.

Standard Apple II graphics modes

Apple IIGS graphics can be displayed in several different resolutions. All standard Apple II graphics modes are supported:

- Lo-Res graphics mode
- Hi-Res graphics mode
- Double Hi-Res graphics mode

Each of these graphics modes is described in the following sections.

Lo-Res graphics

In the Lo-Res graphics mode, the Apple IIGS displays an array of 48 rows by 40 columns of colored blocks. Each block can be any of 16 colors, including black-and-white. On a black-and-white monitor or television set, these colors appear as black, white, and three shades of gray. There are no blank pixels between blocks; adjacent blocks of the same color merge to make a larger shape.

Data for the Lo-Res graphics display are stored in the same part of memory as the data for the 40-column text display. Each byte contains data for two Lo-Res graphics blocks. The two blocks are displayed one atop the other in a display space the same size as a 40-column text character, 7 pixels wide by 8 pixels high.

Half a byte—4 bits, or 1 nibble—is assigned to each graphics block. Each nibble can have a value from 0 to 15, and this value determines which one of 16 colors appears on the screen. The colors and their corresponding nibble values are shown in Table 4-14. In each byte, the low-order nibble sets the color for the top block of the pair, and the high-order nibble sets the color for the bottom block. Thus, a byte containing the hexadecimal value $D8 produces a brown block atop a yellow block on the screen.

■ **Table 4-14** Lo-Res graphics colors

Nibble value			Nibble value		
Dec	**Hex**	**Color**	**Dec**	**Hex**	**Color**
0	$00	Black	8	$08	Brown
1	$01	Deep red	9	$09	Orange
2	$02	Dark blue	10	$0A	Light gray
3	$03	Purple	11	$0B	Pink
4	$04	Dark green	12	$0C	Light green
5	$05	Dark gray	13	$0D	Yellow
6	$06	Medium blue	14	$0E	Aquamarine
7	$07	Light blue	15	$0F	White

Note: Colors may vary, depending on the controls on the monitor or television set.

As explained earlier in this chapter in the section "Video Display Pages," the text display and the Lo-Res graphics display use the same area in memory. Most programs that generate text and graphics clear this part of memory when they change display modes, but it is possible to store data as text and display them as graphics, or vice versa. All you have to do is change the mode switch, described earlier in this chapter in the section "Display Mode Switching," without changing the display data. This usually produces meaningless jumbles on the display, but some programs have used this technique to good advantage for producing complex Lo-Res graphics displays quickly.

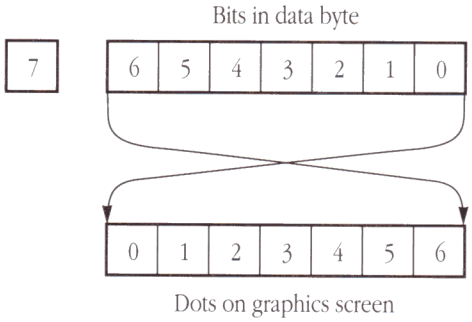
Hi-Res graphics

In the Hi-Res graphics mode, the Apple IIGS displays an array of 192 rows of 280 monochrome pixels, or 140 colored pixels. The smaller number of pixels in color is due to the fact that it takes two bits in display memory to make one color pixel on the screen; in monochrome, one bit makes one pixel. The colors available are black, white, purple, green, orange, and blue.

Data for the Hi-Res graphics displays are stored in either of two 8192-byte areas in memory. These areas are called Hi-Res graphics Page 1 and Page 2. It is in these buffer areas that your high-level language program creates and manipulates the bit images that will appear on the screen. This section describes the way the graphics data bits are converted to pixels on the screen.

The Hi-Res graphics display is bit-mapped: Each pixel on the screen corresponds to a bit (or, in color, 2 bits) in memory. The 7 low-order bits of each display memory byte control a row of 7 adjacent pixels on the screen, and 40 adjacent bytes in memory control a row of 280 (7 times 40) pixels. The least significant bit of each byte is displayed as the leftmost pixel in a row of 7, followed by the second least significant bit, and so on, as shown in Figure 4-16. The eighth bit (the most significant) of each byte is not displayed; it selects one of two color sets, as described later in this chapter.

■ **Figure 4-16** Hi-Res graphics display bits



On a black-and-white monitor, there is a simple correspondence between bits in memory and pixels on the screen. A pixel is white if the bit controlling it is on (1), and the pixel is black if the bit is off (0). On a black-and-white television set, pairs of pixels blur together; alternating black-and-white pixels merge to a continuous gray.

On an NTSC color monitor or a color television set, a pixel whose controlling bit is off (0) is black. If the bit is on, the pixel will be white or a color, depending on its position, the pixels on either side, and the setting of the high-order bit of the byte.

Call the leftmost column of pixels column 0 and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control pixels in even-numbered columns (0, 2, 4, and so forth) are on, the pixels are purple; if the bits that control odd-numbered columns are on, the pixels are green—but only if the pixels on both sides of a given pixel are black. If two adjacent pixels are both on, they are both white.

You can select the other two colors, blue and orange, by turning the high-order bit (bit 7) of a data byte on (1). The colored pixels controlled by a byte with the high-order bit on are either blue or orange: The pixels in even-numbered columns are blue, and the pixels in odd-numbered columns are orange—again, only if the pixels on both sides are black.

Within each horizontal line of seven pixels controlled by a single byte, you can have black, white, and one pair of colors. To change the color of any pixel to one of the other pair of colors, you must change the high-order bit of its byte, which affects the colors of all seven pixels controlled by the byte.

In other words, Hi-Res graphics displayed on a color monitor or television set are made up of colored pixels, according to the following rules:

- Pixels in even columns can be black, purple, or blue.
- Pixels in odd columns can be black, green, or orange.
- If adjacent pixels in a row are both on, they are both white.
- The colors in each row of seven pixels controlled by a single byte are either purple and green, or blue and orange, depending on whether the high-order bit is off (0) or on (1).

These rules are summarized in Table 4-15. The blacks and whites are numbered to remind you that the high-order bit is different.

Table 4-15 Hi-Res graphics colors

Bits 0-6	Bit 7 off	Bit 7 on
Adjacent columns off	Black 1	Black 2
Even columns on	Purple	Blue
Odd columns on	Green	Orange
Adjacent columns on	White 1	White 2

Note: Colors may vary, depending on the controls on the monitor or television set.

The peculiar behavior of the Hi-Res colors reflects the way NTSC color television works. The pixels that make up the Apple IIGS video signal are spaced to coincide with the **frequency** of the color subcarrier used in the NTSC system. Alternating black-and-white pixels at this spacing causes a color monitor or TV set to produce color, but 2 or more white pixels together do not. Effective horizontal resolution with color is 140 pixels per line (280 divided by 2).

Double Hi-Res graphics

In the Double Hi-Res graphics mode, the Apple IIGS displays an array of 140 colored pixels or 560 monochrome pixels wide and 192 rows deep. There are 16 colors available for use with Double Hi-Res graphics. (See Table 4-16.)

■ **Table 4-16** Double Hi-Res graphics colors

Repeated color pattern	ab0	mb1	ab2	mb3	Bit
Black	$00	$00	$00	$00	0000
Deep red	$08	$11	$22	$44	0001
Brown	$44	$08	$11	$22	0010
Orange	$4C	$19	$33	$66	0011
Dark green	$22	$44	$08	$11	0100
Dark gray	$2A	$55	$2A	$55	0101
Green	$66	$4C	$19	$33	0110
Yellow	$6E	$5D	$3B	$77	0111
Dark blue	$11	$22	$44	$08	1000
Purple	$19	$33	$66	$4C	1001
Light gray	$55	$2A	$55	$2A	1010
Pink	$5D	$3B	$77	$6E	1011
Medium blue	$33	$66	$4C	$19	1100
Light blue	$3B	$77	$6E	$5D	1101
Aquamarine	$77	$6E	$5D	$3B	1110
White	$7F	$7F	$7F	$7F	1111

Double Hi-Res graphics is a bit-mapping of the low-order 7 bits of the bytes in the main-memory and auxiliary-memory pages at $2000 through $3FFF. The bytes in the main-memory and auxiliary-memory pages are interleaved in exactly the same manner as the characters in 80-column text: Of each pair of identical addresses, the auxiliary-memory byte is displayed first, and the main-memory byte is displayed second. Horizontal resolution is 560 pixels when displayed on a monochrome monitor.

Unlike Hi-Res color, Double Hi-Res color has no restrictions on which colors can be adjacent. Color is determined by any 4 adjacent pixels along a line. Think of a 4-pixel-wide window moving across the screen: At any given time, the color displayed will correspond to the 4-bit value from Table 4-16 that corresponds to the window's position (Figure 4-9). Effective horizontal resolution with color is 140 (560 divided by 4) pixels per line.

To use Table 4-16, divide the display column number by 4, and use the remainder to find the correct column in the table: ab0 is a byte residing in auxiliary memory, corresponding to a remainder of zero (byte 0, 4, 8, and so on); mb1 is a byte residing in main memory, corresponding to a remainder of one (byte 1, 5, 9 and so on), and similarly for ab3 and mb4.

Super Hi-Res graphics

The Apple IIGS has two graphics modes that are new to the Apple II family. These are the 320-pixel and 640-pixel Super Hi-Res graphics modes, which increase horizontal resolution to either 320 or 640 pixels and increase vertical resolution to 200 lines. The VGC is primarily responsible for implementing the Super Hi-Res video graphics, which provide these capabilities:

- 320- or 640-pixel horizontal resolution
- 200-line vertical resolution
- 12-bit color resolution that allows choices from 4096 available colors
- 16 colors for each of the 200 lines—up to 256 colors per frame
- Color Fill mode
- scan-line interrupts
- all new video mode features, programmable for each scan line
- linear display buffer
- pixels contained within byte boundaries

The New-Video register

When a standard Apple II video mode (Lo-Res, Hi-Res, or Double Hi-Res graphics) is enabled, the Mega II accesses the video memory buffers and generates video. When Super Hi-Res graphics is enabled, the Video Graphics Controller has sole access to the video buffers. The bit to enable this access, along with the memory map configuration switch, is in the New-Video register located at $C029. The bit descriptions for this register are shown in Figure 4-17. Table 4-17 gives a description of each bit.

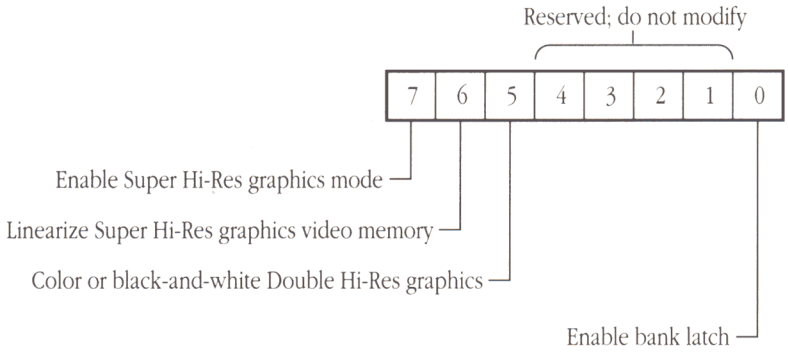
▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Table 4-17** Bits in the New-Video register

Bit	Value	Description
7	0	Selects Apple II video mode. If this bit is 0, all existing Apple II-compatible video modes are enabled. The Mega II alone reads the video memory during the video cycles and generates the video.
	1	Selects Super Hi-Res graphics video modes. If this bit is 1, all standard Apple II video modes are disabled; either 320-pixel resolution (and Color Fill mode) or 640-pixel resolution graphics are enabled. (The selection of 320 or 640 is made in the scan-line control byte for each line.) Also, when this bit is 1, bit 6 is overridden, and the memory map is changed to support the Super Hi-Res graphics video buffer, as described below. (See the description of bit 6.)
6*	0	If this bit is 0, the 128K memory map is the same as the Apple IIe.
	1	If this bit is 1, the memory map is reconfigured for use with Super Hi-Res graphics video mode: The video buffer becomes one contiguous, linear address space from $2000 through $9D00. (Figure 4-18 shows the Super Hi-Res graphics buffer.)
5	0	If this bit is 0, Double Hi-Res graphics is displayed in color (140 by 192, 16 colors).
	1	If this bit is 1, Double Hi-Res graphics is displayed in black-and-white (560 by 192).
4–1	–	Reserved; do not modify.
0	0	Enable bank latch. If this bit is 1, the 17th address bit is used to select either the main or auxiliary memory bank. If the address bit is 1, then the auxiliary bank is enabled. (Actually data bit 0 is used as the 17th address bit). If the address bit is 0, the state of the memory configuration soft switches determines which memory bank is enabled. See Chapter 3 for descriptions of the memory configuration soft switches. Table 4-18 shows how to use this bit to select a memory bank.
	1	The 17th address bit is ignored.

* Set bit 6 to 0 whenever using Double Hi-Res graphics mode. This is necessary to ensure that the video display will function properly.

■ **Figure 4-17** New-Video register



■ **Table 4-18** Memory bank selection using bit 0 of the New-Video register

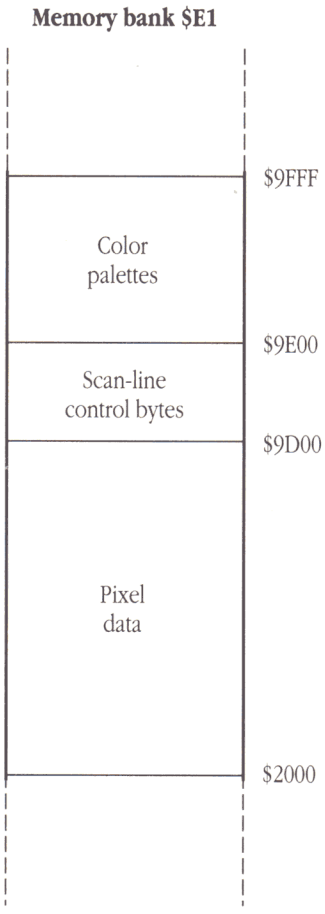
New-Video register Bit 0	Data bit 0	Memory bank enabled
0	1	Auxiliary
	0	Determined by state of memory configuration soft switches
1	Ignored	-

The Super Hi-Res graphics buffer

The Super Hi-Res graphics display buffer contains three types of data: scan-line control bytes, color palettes, and pixel data. Figure 4-18 shows a memory map of the display buffer. This buffer resides in contiguous bytes of the auxiliary 64K bank of the slow RAM ($E1) from $2000 through $9FFF. Note that this display buffer uses memory space used for the Apple II Double Hi-Res graphics buffers, but leaves the other graphics and text display buffers untouched.

The next three sections describe the scan-line control bytes, color palettes, and pixel data bytes used in Super Hi-Res graphics mode.

■ **Figure 4-18** Super Hi-Res graphics display buffer



Scan-line control bytes ($9D00–$9DC7)

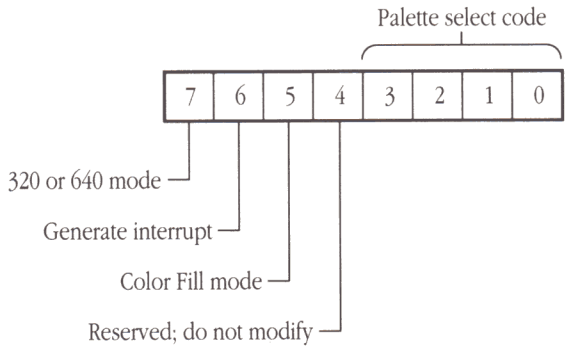
An added advantage of the new Apple IIGS video graphics is the ability to select the Super Hi-Res graphics horizontal resolution for each video scan line. The 200 scan-line control bytes (located from $9D00 through $9DC7 as shown in Figure 4-18) control the features for each scan line. There is one 8-bit control byte for each of the 200 scan lines. For each line, you can select

- the palette (16 colors) to be used on the scan line
- Color Fill mode on the scan line
- an interrupt to be generated on the scan line
- either 320-pixel or 640-pixel resolution for the scan line

The scan-line control byte bits and their functions are listed in Figure 4-19. Table 4-19 gives a description of each bit.

▲ **Warning** Be careful when changing bits within this byte. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 4-19** Scan-line control byte format



■ **Table 4-19** Bits in a scan-line control byte

Bit	Value	Description
7	1	Horizontal resolution = 640 pixels.
	0	Horizontal resolution = 320 pixels.
6	1	Interrupt enabled for this scan line. (When this bit is a 1, the scan-line interrupt status bit is set at the beginning of the scan line.)
	0	Scan-line interrupts disabled for this scan line.
5	1	Color Fill mode enabled. (This mode is available in Super Hi-Res graphics 320-pixel resolution mode only; in 640-pixel mode, Color Fill mode is disabled.)
	0	Color Fill mode disabled.
4	–	Reserved; write 0.
0–3	–	Palette chosen for this scan line.

The location of the scan-line control byte for each scan line is $9Dxx, where *xx* is the hexadecimal value of the line. For example, the control byte for the first scan line (line 0) is located in memory location $9D00; the control byte for the second scan line (line 1) is in location $9D01, and so forth.

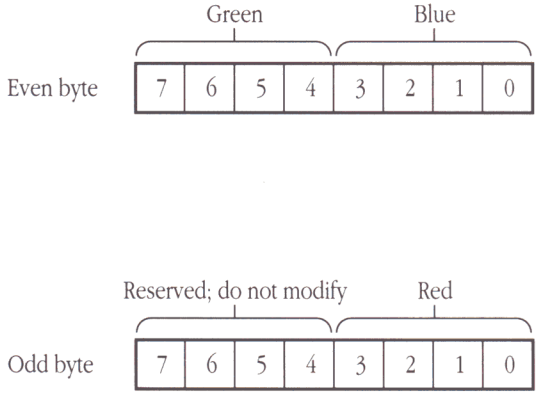
◆ *Note:* The first 200 bytes of the 256 bytes in the memory page beginning at $9D00 are scan-line control bytes, and the remaining 56 bytes are reserved for future expansion. For compatibility with future Apple products, do not modify these 56 bytes.

Color palettes ($9E00–$9FFF)

A color palette is a group of 16 colors to be displayed on the scan line. Each scan line can have one of 16 color palettes assigned to it. You can choose the 16 colors in each palette from any of the 4096 possible colors. You can draw each pixel on the scan line in any of these 16 colors.

These colors are determined by a 12-bit value made up of three separate 4-bit values. (12 bits allows 2^{12} or 4096 possible combinations for each palette color.) Each 4-bit quantity represents the intensity of each red, green, and blue. The combination of the magnitudes of each of the three primary colors determines the resulting color. Figure 4-20 shows the format of each of these 4-bit values that make up a palette color.

■ **Figure 4-20** Color palette format



The color palettes are located in video buffer locations $9E00 through $9FFF in bank $E0. There are 16 color palettes in this space, with 32 bytes per palette. Each color palette represents 16 colors, with 2 bytes per color. The palette indicated in the scan-line control byte is used to display the pixels in color on the scan line. The starting address for each of the color palettes and the colors within them are listed in Table 4-20. The 16 colors within a palette have numbers $0 through $F. Note that each color begins on an even address.

Once you have filled the palettes with the colors to be used and selected the display modes within each of the scan-line control bytes, you must choose which of the 16 colors you are going to display for each pixel.

■ **Table 4-20** Palette and color starting addresses

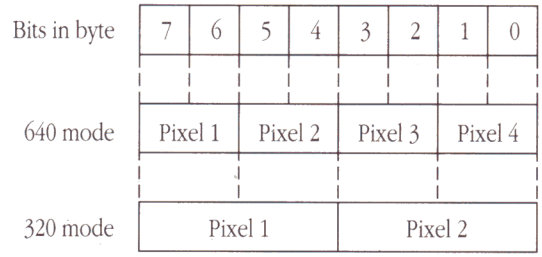
Palette number	Color$0	Color $1	...	Color $E	Color $F
$0	$9E00–01	$9E02–03	...	$9E1C–1D	$9E1E–1F
$1	$9E20–21	$9E22–23	...	$9E3C–3D	$9E3E–3F
$2	$9E40–41	$9E42–43	...	$9E5C–5D	$9E5E–5F
.	.	.		.	.
.	.	.		.	.
.	.	.		.	.
$F	$9FE0–E1	$9FE2–E3	...	$9FFC–FD	$9FFE–FF

Pixels

The Super Hi-Res graphics color information for each pixel is different for each of the two resolution modes: 4 bits represent each pixel color in 320-pixel mode; 2 bits represent the pixel color in 640-pixel mode. Higher resolution comes with a slight penalty, however: Although in 320 mode a pixel may be any of 16 colors chosen from the palette, a pixel may be one of only 4 colors in 640 mode.

The pixel data are located in the display buffer in a linear and contiguous manner; $2000 corresponds to the upper-left corner of the display, and $9CFF corresponds to the lower-right corner. Each scan line uses 160 ($A0) bytes. Figure 4-21 shows the format in which the pixel color data are stored in both the 320-pixel and 640-pixel modes.

■ **Figure 4-21** Pixel data byte format



In 320-pixel mode, four bits determine each pixel color, and data are stored two pixels to a byte of the display buffer. Since four bits determine the pixel color, in 320 mode each pixel can be any of the 16 colors from that palette.

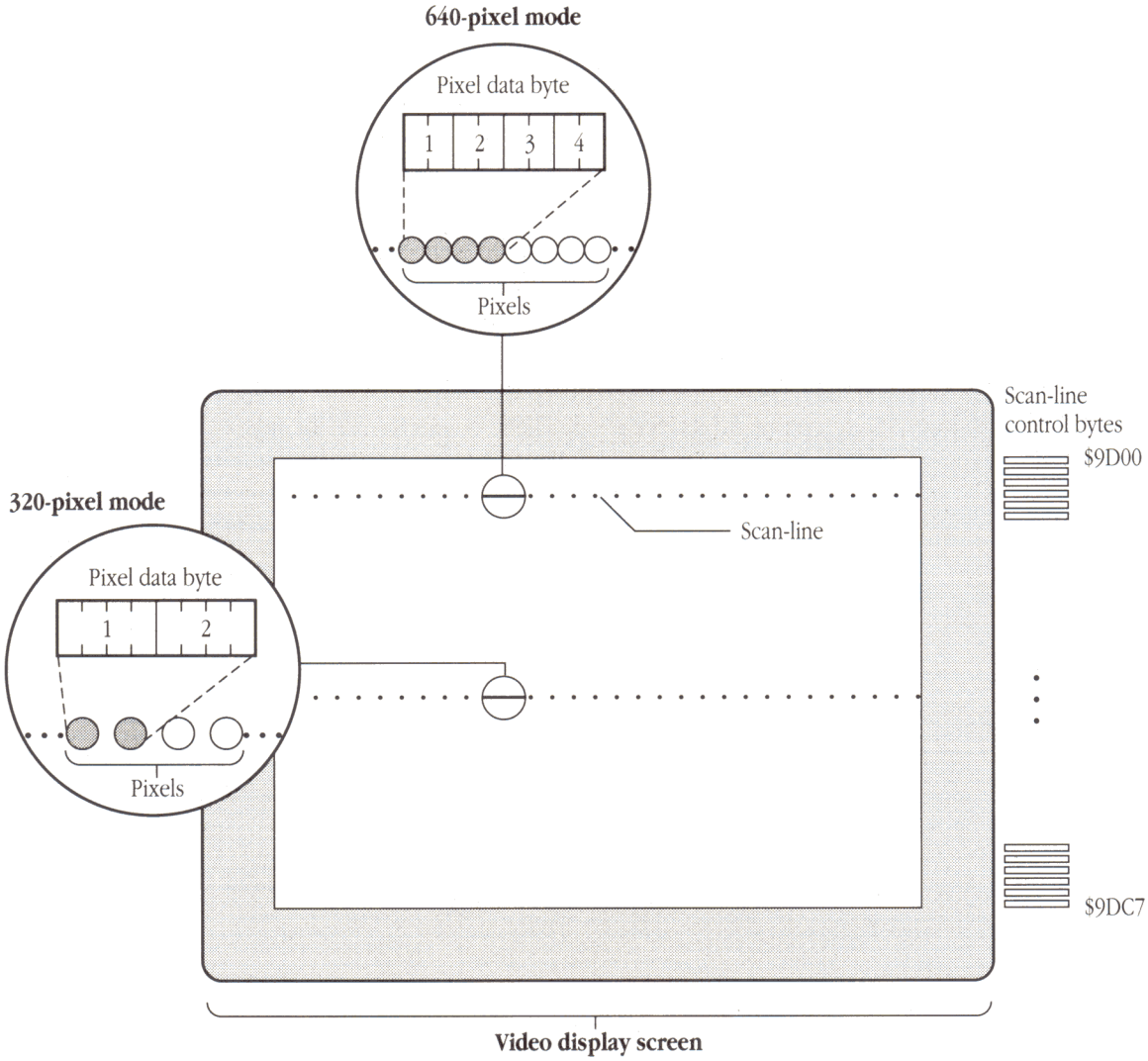
In 640-pixel mode, color selection is more complicated. The 640 pixels in each horizontal line occupy 160 adjacent bytes of memory, each byte representing 4 pixels that appear side-by-side on the screen. The 16 colors in the palette are divided into four groups of 4 colors each. The first pixel in each horizontal line can select one of 4 colors from the third group of 4 in the palette. The second pixel selects from the fourth group of 4 colors in the palette. The third pixel selects from the first group of 4 colors, and the fourth pixel selects from the second group, as shown in Table 4-21. The process repeats for each successive group of 4 pixels in a horizontal line. Thus, even though a given pixel can be one of 4 colors, different pixels in a line can take on any of the 16 colors in a palette.

■ **Table 4-21** Color selection in 640 mode

Pixel	Value	Palette color	Pixel	Value	Palette color
3	0	$0	1	0	$8
	1	$1		1	$9
	2	$2		2	$A
	3	$3		3	$B
4	0	$4	2	0	$C
	1	$5		1	$D
	2	$6		2	$E
	3	$7		3	$F

Figure 4-22 shows the display screen and the pixels that make up each scan line. Also shown are the pixel data bytes for both 640- and 320-pixel Super Hi-Res graphics mode. The scan-line control bytes, one for each scan line, are shown at the right.

■ **Figure 4-22** Drawing pixels on the screen



Dithering

In Super Hi-Res graphics mode using 640-pixel resolution, colors other than the available 4 palette colors may be displayed by a means called **dithering.** By choosing 2 adjacent pixel colors that mix to obtain a third desired color, you can increase the number of hues available. For example, in Figure 4-23, when red is selected from the available colors, the scan line appears as red. Alternating red and yellow results in orange, and so on. Through the use of dithering, as many as 16 colors can be generated.

Color Fill mode

Another feature of Apple IIGS video graphics is Color Fill, an option that simplifies the task of painting continuous color on any one line. Color Fill, which is available in 320-pixel mode only, is used to fill rapidly a large area of the video display with a single color. In this mode, color $0 in the palette takes on a unique definition. Any pixel data byte containing the color value $0 causes that pixel to take on the color of the previous pixel instead of displaying a palette color. This means that only 15 unique palette colors ($1 through $F) are available for each scan line, rather than 16 colors. For example, assume that A, B, and C represent 3 different palette colors, 4 bits per pixel. These colors do not include color $0. The desired color pattern for a series of pixels on a line might be as follows without Color Fill mode:

```
AAAAAAAAAAAABBBBBBBBBBBBCCCCCCCCCCCC
```

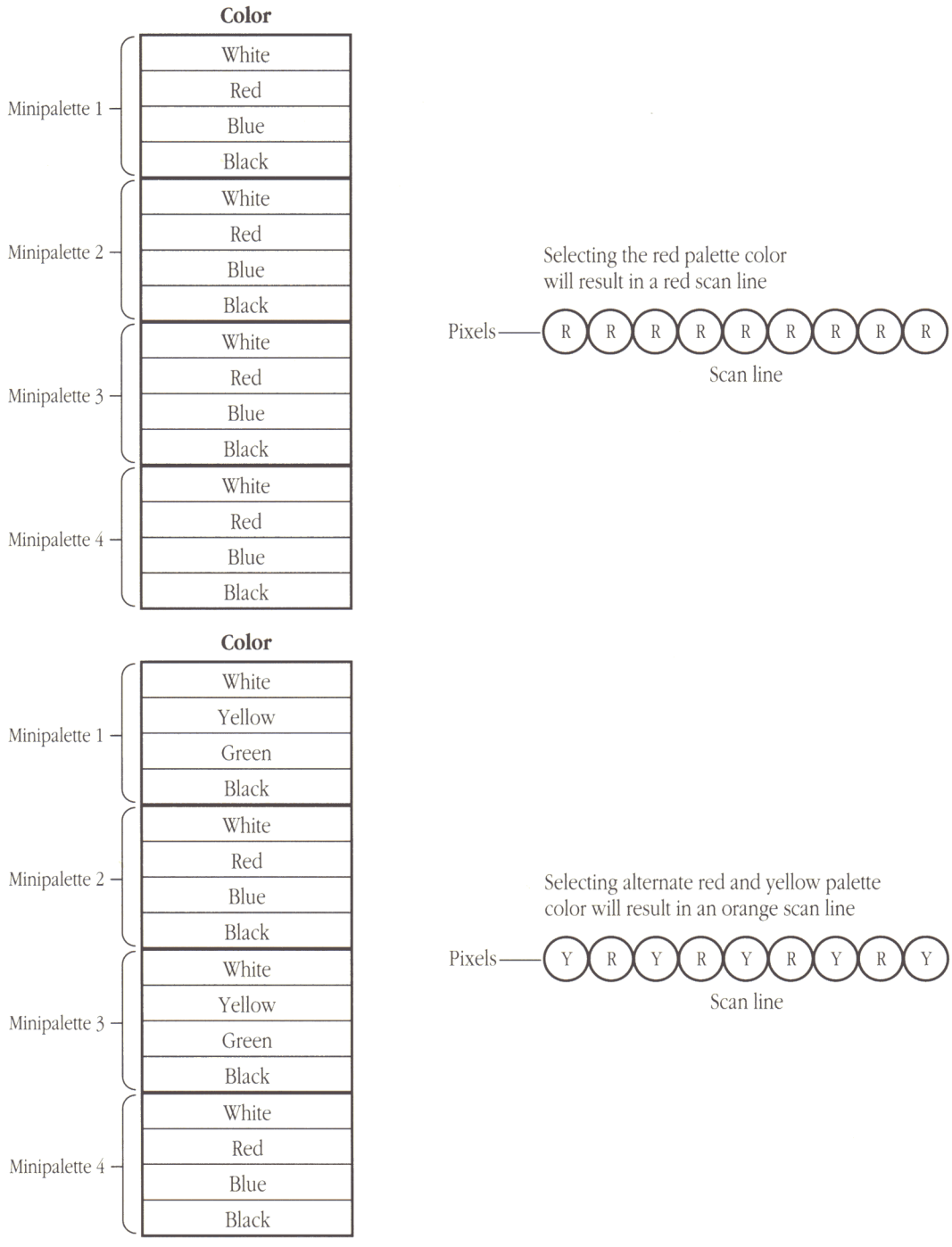
The same color pattern would be created by using Color Fill mode as follows:

```
A000000000000B000000000000C000000000000
```

Method 2 would save time: The program only needs to fill the pixel area of the scan line once with 0, and then to write a color value into those locations where a color should begin or change. In the example just given, only one byte needs to be written to implement each new color on the scan line using the Color Fill method, as opposed to six bytes per color without Color Fill.

The only restriction of the Color Fill mode is that the first pixel value on a scan line must not be 0; if the first pixel value is 0, then an undetermined color results.

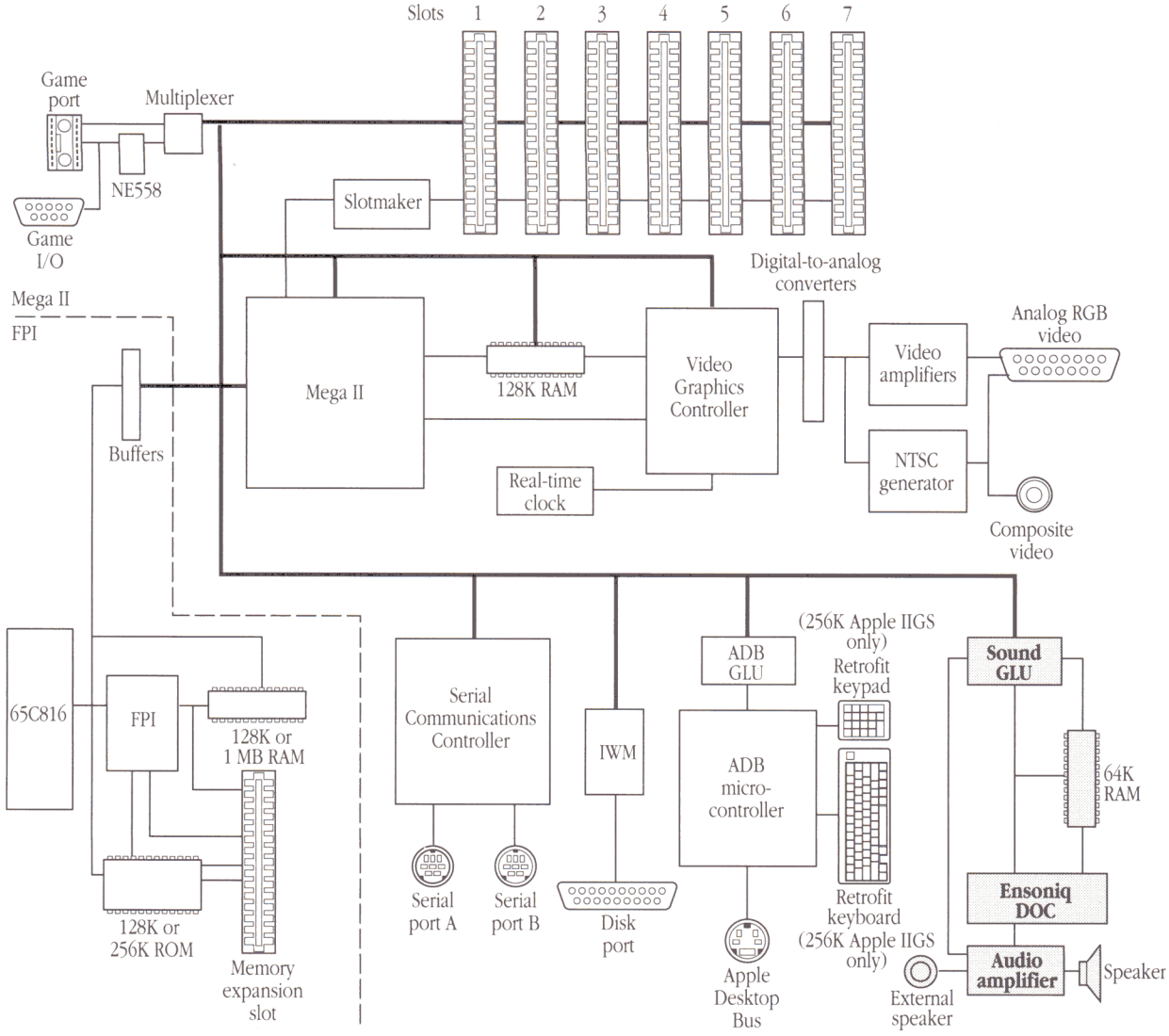
■ **Figure 4-23** Examples of dithering



Chapter 5 **Apple IIGS Sound**

One of the Apple IIGS computer's outstanding features is its sound capability. By programming the Apple IIGS, you can utilize this powerful sound-synthesizing ability; your ability to generate sounds is limited only by your imagination. This chapter covers the Digital Oscillator Chip (DOC), the individual oscillators, and the many registers associated with these oscillators. Also covered is the Sound General Logic Unit (GLU) and its associated registers. Figure 5-1 shows the relationship of the sound components to the rest of the computer.

■ **Figure 5-1** Sound components in the Apple IIGS



The built-in speaker

The built-in speaker is located on the left side of the baseplate directly below the keyboard. This sound output device is used as the primary device for standard Apple II sound output, and also for the output of the Digital Oscillator Chip.

One-bit sound

Quite complex sounds can be created simply by toggling a bit whose output is connected to the speaker. This bit can be set and cleared by accessing (either reading or writing to) location $C030. The faster you access this location, the faster the sound bit toggles and the higher the pitch of the sound. Using timing loops in the application program will allow you to toggle the speaker at any frequency starting at about 400 cycles per second. Below this frequency, the speaker clicks on every other access to $C030.

If you switch the speaker once, it emits a click; to make longer sounds, you access the speaker repeatedly.

- *Note:* If you access this soft switch by using an assembly-language indexed-mode command, it switches twice in rapid succession. The resulting pulse is so short that the speaker doesn't have time to respond; it doesn't make a sound.

With the inclusion of the DOC into the Apple IIGS, sound generation becomes much more sophisticated. The following sections describe the DOC and this computer's powerful sound-making ability.

Sound synthesis

Sound is synthesized by programming digital oscillators to produce waveforms that simulate ordinary sounds (musical, human, or other) or generate unique ones. These waveforms can be programmed manually (values placed in memory individually) or through digitization of an external analog-input signal. As stated earlier, the Apple IIGS uses a toolbox of utilities to perform many different functions: graphics, disk access, and sound. The following description of the DOC is meant to familiarize you with the general principles of Apple IIGS sound generation. When you program this computer for sound, using the toolbox utilities will result in the proper use of the DOC and ensure software compatibility with future Apple II products. (To find out how to use the sound tools, refer to the *Apple IIGS Toolbox Reference.*)

The Sound GLU

To program the DOC or build a wavetable in the Sound RAM, you must write command and data bytes to registers within the chip. This process is facilitated by the GLU, which serves as an interface between the microprocessor, the DOC, and the dedicated 64K–by–8-bit dynamic RAM. This interface allows the DOC chip to run independently of the rest of the system.

The Sound GLU contains

- a Sound Control register
- a data register
- a pair of Address Pointer (high and low address) registers

These registers and their addresses are listed in Table 5-1 and described in detail in the sections that follow.

■ **Table 5-1** GLU registers

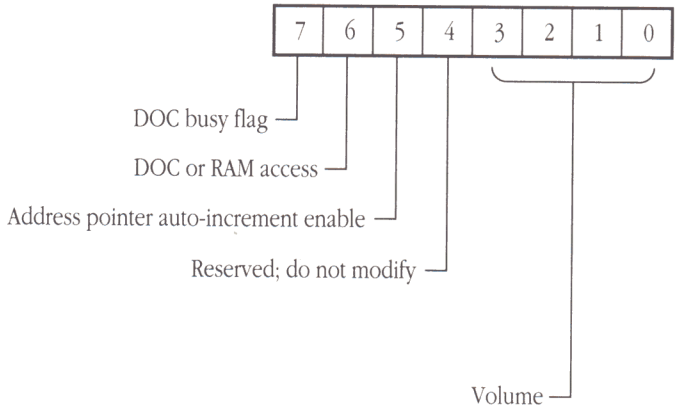
GLU registers	Address	Type
Sound Control register	$C03C	R/W
Data register	$C03D	R/W
Address Pointer register, low byte	$C03E	R/W
Address Pointer register, high byte	$C03F	R/W

The Sound Control register

The Sound Control register controls whether the microprocessor accesses the DOC internal registers or the Sound RAM. This register also controls whether or not the Address Pointer registers auto-increment, that is, increment automatically after every RAM read or write, thereby avoiding the necessity of reloading the pointers with addresses after each access. Figure 5-2 shows the format of the Sound Control register. Table 5-2 gives a description of each bit.

▲ **Warning** Do not use a read-modify-write command when altering bits in this register. ▲

■ **Figure 5-2** Sound Control register at $C03C



■ **Table 5-2** Bits in the Sound Control register

Bit	Value	Description
7	1	When this read-only bit is 1, the DOC is busy. Loop on this bit until it is clear.
	0	When this bit is 0, the DOC is free. The DOC will respond to register reads and writes.
6	1	All accesses are to the dedicated 64K RAM.
	0	All accesses are to the DOC.
5	1	Address auto-incrementing is enabled.
	0	Address auto-incrementing is disabled; Address Pointer registers hold the last value.
4	–	Reserved; do not modify.
3–0	–	Volume control: $0 is low volume, $F is high volume.

Data register

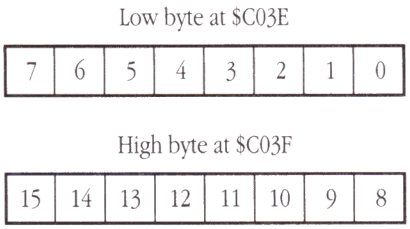
To load values into the DOC registers, write to the data register at $C03D. You also write to this register when you want to place values in Sound RAM. If the Sound Control register is set to access the DOC, writes to the data register will result in data bytes' being loaded into the DOC register indicated by the Address Pointer registers. If the Sound Control register is set to access the Sound RAM, writes to the data register will result in data bytes' being loaded into the Sound RAM address indicated by the Address Pointer registers.

Address Pointer registers

When accessing the Sound RAM (bit 6 set to 1 in the Sound Control register), the Address Pointer registers point to the address of the next byte in Sound RAM. The high-byte Address Pointer register contains the high 8 bits of the 16-bit address, and the low-byte register contains the low 8 bits.

When accessing the Sound DOC (bit 6 set to 0 in the Sound Control register), the high-byte Address Pointer register is ignored by the DOC, and the low-byte Address Pointer register points to the DOC register to be written or read from. Figure 5-3 shows the format of the Address Pointer registers.

■ **Figure 5-3** Address Pointer registers



Write operation

To write to the DOC or Sound RAM:

1. Set the Sound Control register
   - □ to point to either the RAM or the DOC
   - □ to enable or disable auto-incrementing in the Address Pointer registers
2. Then load the Address Pointer register with the beginning location into which data are to be written. Do this by writing the high byte of the address to the high-byte Address Pointer register at $C03F, and the low byte of the address to the low-byte Address Pointer register at $C03E.

Data now written to the data register will be transferred by the Sound GLU into the corresponding memory (if you are accessing RAM) or DOC register (if you are accessing the DOC).

If the auto-increment feature is enabled, the Address Pointer register is automatically incremented to the next higher location or the register with the next higher address after each write to the data register.

- *Note:* Do not use indexed addressing mode when reading data from or writing data to the data register. Indexed addressing mode generates a false read, which will cause the sound GLU to lose synchronization.

Read operation

The Sound RAM read operation is the same as the write operation with one exception: Reading from the data register lags by one read cycle. For example, if you want to read 10 bytes from the Sound RAM, select the RAM by setting the Sound Control register bit and enabling auto-incrementing. Then set the Address Pointer register to the starting address and read the data register 11 times, discarding the first byte read.

The Ensoniq DOC

The Apple IIGS uses the Ensoniq 5503 Digital Oscillator Chip, a programmable sound synthesizer chip with 32 independent oscillators, volume control, and digitizing capability. The synthesizer uses 64K of RAM dedicated to sound waveform storage, and interfaces with the 65C816 microprocessor via the Sound GLU. Commands and data are transferred to the DOC via the GLU, and sound is output via the built-in speaker, external speaker jack, or molex connector on the main logic board.

The DOC contains three registers common to all oscillators. These are

- the Oscillator Interrupt register ($E0)
- the Oscillator Enable register ($E1)
- the Analog-to-Digital (A/D) Converter register ($E2)

Also, each oscillator has one of each of the following registers dedicated to it:

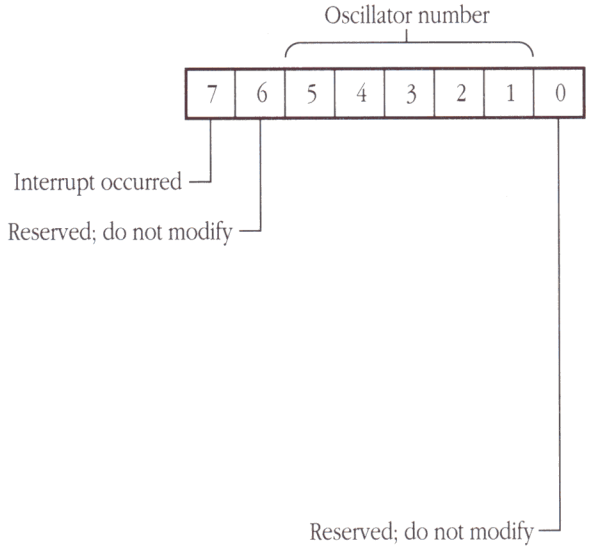
- an Oscillator Control register
- an Oscillator Data register
- a Volume register
- a Frequency Low register
- a Frequency High register
- a Wavetable Size register
- a Wavetable Pointer register

DOC registers common to all oscillators

The Oscillator Interrupt register ($E0)

This register contains the status of the DOC interrupt request (IRQ) pin and the number of the oscillator that generated the interrupt, if any. When an oscillator reaches the end of a wavetable and the enable interrupt (EI) bit for that oscillator has previously been set, the IRQ line and bit 7 of the Oscillator Interrupt register are then set, and the register number is entered in bits 1 through 5 of the Oscillator Interrupt register. Figure 5-4 shows the format of the Oscillator Interrupt register. Table 5-3 gives a description of each of the bits.

■ **Figure 5-4** Oscillator Interrupt register at $E0



The Oscillator Enable register ($E1)

The Oscillator Enable register controls the number of oscillators that are operating at a particular time. To enable 1 or more oscillators, multiply the desired quantity of oscillators (up to 32) by 2 and enter the number in this register at $E1. You may enter any number from 2 to 64, which will enable the corresponding oscillators in sequential order. (Low-numbered oscillators cannot be skipped in order to enable a higher-numbered one.) A minimum of 1 oscillator is always enabled, which is also the reset default.

■ **Table 5-3** Bits in the Oscillator Interrupt register

Bit	Value	Description
7	1	No oscillator has generated an interrupt.
	0	One of the 32 DOC oscillators has generated an interrupt; this bit reflects the status of the IRQ line.
6	–	Reserved; do not modify.
5–1	–	Interrupting oscillator number; when one of the 32 DOC oscillators generates an interrupt, the number of the oscillator is contained here.
0	–	Reserved; do not modify.

The A/D Converter register ($E2)

The A/D Converter register contains the output value of the successive-approximation analog-to-digital converter. An analog-input signal can be sampled at pin 1 of the 7-pin molex connector (J25). The result of the conversion resides in the A/D Converter register at the completion of the conversion. Reading this register at $E2 initiates the 31-microsecond conversion process. If this register is read before the end of the conversion process, the value will be lost and a new conversion will begin.

DOC registers for individual oscillators

Table 5-4 contains the addresses for the registers dedicated to the individual oscillators.

The Oscillator Control registers ($A0–$BF)

Each Oscillator Control register controls all functions of each oscillator, including

- which of eight optional external analog multiplexer channels an oscillator will use
- whether or not an oscillator may generate an interrupt
- the oscillator's mode of operation

■ **Table 5-4** DOC register addresses

Oscillator number	Frequency Low register	Frequency High register	Volume register	Data register	Wavetable Pointer register	Control register	Wavetable Size register
$00	$00	$20	$40	$60	$80	$A0	$C0
$01	$01	$21	$41	$61	$81	$A1	$C1
$02	$02	$22	$42	$62	$82	$A2	$C2
$03	$03	$23	$43	$63	$83	$A3	$C3
$04	$04	$24	$44	$64	$84	$A4	$C4
$05	$05	$25	$45	$65	$85	$A5	$C5
$06	$06	$26	$46	$66	$86	$A6	$C6
$07	$07	$27	$47	$67	$87	$A7	$C7
$08	$08	$28	$48	$68	$88	$A8	$C8
$09	$09	$29	$49	$69	$89	$A9	$C9
$0A	$0A	$2A	$4A	$6A	$8A	$AA	$CA
$0B	$0B	$2B	$4B	$6B	$8B	$AB	$CB
$0C	$0C	$2C	$4C	$6C	$8C	$AC	$CC
$0D	$0C	$2D	$4D	$6D	$8D	$AD	$CD
$0E	$0E	$2E	$4E	$6E	$8E	$AE	$CE
$0F	$0F	$2F	$4F	$6F	$8F	$AF	$CF
$10	$10	$30	$50	$70	$90	$B0	$D0
$11	$11	$31	$51	$71	$91	$B1	$D1
$12	$12	$32	$52	$72	$92	$B2	$D2
$13	$13	$33	$53	$73	$93	$B3	$D3
$14	$14	$34	$54	$74	$94	$B4	$D4
$15	$15	$35	$55	$75	$95	$B5	$D5
$16	$16	$36	$56	$76	$96	$B6	$D6
$17	$17	$37	$57	$77	$97	$B7	$D7
$18	$18	$38	$58	$78	$98	$B8	$D8
$19	$19	$39	$59	$79	$99	$B9	$D9
$1A	$1A	$3A	$5A	$7A	$9A	$BA	$DA
$1B	$1B	$3B	$5B	$7B	$9B	$BB	$DB
$1C	$1C	$3C	$5C	$7C	$9C	$BC	$DC
$1D	$1D	$3D	$5D	$7D	$9D	$BD	$DD
$1E*	$1E	$3E	$5E	$7E	$9E	$BE	$DE
$1F*	$1F	$3F	$5F	$7F	$9F	$BF	$DF

* These oscillators are reserved for system use. Use of these oscillators by the user may result in a system crash.

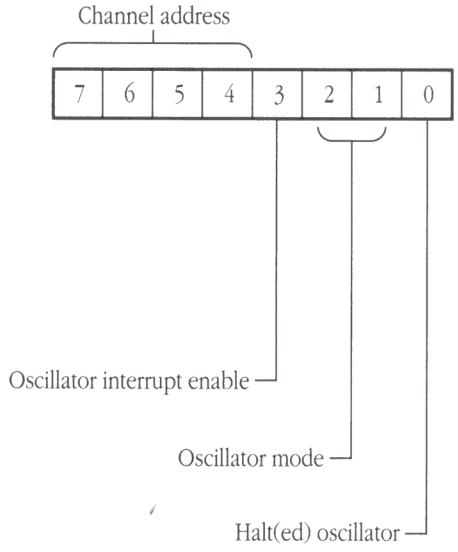
The oscillator may function in one of several modes:

- **Free Run mode:** The oscillator begins at the beginning of the wavetable and repeats the same wavetable. The oscillator will halt when the halt bit is set or when a 0 is encountered in the table data.

- **One Shot mode:** The oscillator begins at the beginning of the wavetable, stepping through it only once, and stopping at the end of the table.
- **Sync mode:** You enable Sync mode by selecting even/odd pairs of oscillators (a lower even-numbered oscillator paired with an adjacent higher-numbered oscillator). When the even-numbered oscillator begins its wavetable, the odd-mate oscillator will synchronize and begin its wavetable simultaneously.
- **Swap mode:** Uses even/odd pairs of oscillators (a lower even-numbered oscillator paired with an adjacent higher odd-numbered oscillator). The enabled oscillator pair runs in One Shot mode. When it reaches the end of its wavetable, it resets its **accumulator** to 0, sets its halt bit, and clears the halt bit of its mate.

Figure 5-5 shows the format of this register. Table 5-5 gives a description of each bit.

Figure 5-5 Oscillator Control register



The Oscillator Data registers ($60–$7F)

The Oscillator Data registers are read-only registers and contain the last byte read by an oscillator from the wavetable.

The Volume registers ($40–$5F)

The Volume registers contain an oscillator's volume value. The current wavetable data byte is multiplied by the 8-bit volume value to obtain the oscillator final output level.

■ **Table 5-5** Bits in the Oscillator Control register

Bit	Value	Description
7–4	–	Channel address bits: Only the low three bits are supported by the hardware. Bits 7–4 determine to which demultiplexer output channel (provided by optional external demultiplexer hardware) this oscillator will be directed. Connecting a demultiplexer to the 7-pin molex connector will allow you to use up to eight separate sound channels. Figure 5-10 shows an example of how the external demultiplexer circuitry may be implemented.
3	1	Interrupts enabled: An interrupt flag will be set in the DOC's Oscillator Interrupt register and the DOC will assert the IRQ signal when an oscillator generates an interrupt.
	0	Interrupts disabled: The interrupt flag will not be set in the Oscillator Interrupt register when an oscillator generates an interrupt.
2–1	–	Oscillator mode: Select the mode desired by setting these two bits as follows:

Bit 2	Bit 1	Mode
0	0	Free Run
0	1	One Shot
1	0	Sync
1	1	Swap

0	1	Halted oscillator: This is a read/write bit. To halt the oscillator, set this bit. Certain modes (Sync, Swap) will halt the oscillator and set this bit automatically after completion.
	0	Running oscillator: This bit is cleared if the corresponding oscillator is currently enabled.

The Frequency High and Frequency Low registers ($00–$3F)

The Frequency High registers and Frequency Low registers are concatenated to create a 16-bit value for each oscillator. This frequency value determines the speed at which the wavetable is read from memory. This speed indirectly determines the frequency of the output signal at the speaker. The relationship between output signal frequency, wavetable scan rate, and the Frequency High and Frequency Low register values is

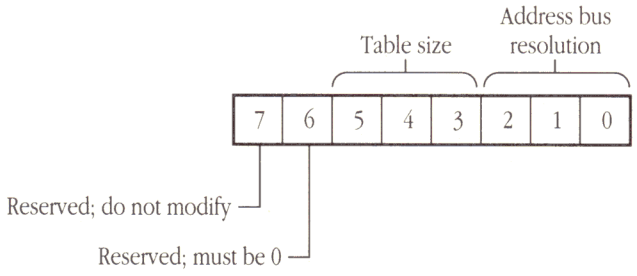
$$\text{Output frequency} = \frac{SR}{2^{(17+RES)}} * F_{HL}$$

where the scan rate $SR = \frac{894.886KHz}{OSC + 2}$ and where *RES* is the resolution value in the Wavetable register, F_{HL} is the 16-bit frequency value concatenated from the Frequency High and Frequency Low registers, and OSC is the number of enabled oscillators.

The Wavetable Size registers ($C0–$DF)

The Wavetable Size registers control the size of the individual wavetable each oscillator will access. Figure 5-6 shows the format of the Wavetable Size register. Table 5-6 gives a description of each bit. The size of the wavetable for each oscillator can vary from 256 bytes to 32K in eight discrete steps, as shown in Table 5-7.

■ **Figure 5-6** Wavetable Size register



■ **Table 5-6** Bits in the Wavetable Size register

Bit	Value	Description
7	–	Reserved; do not modify.
6	0	Extended addressing: The Apple IIGS uses only 64K for the Sound RAM and has no high memory bank available. Therefore, this bit must always be set to 0.
5–3	–	Table size: The wavetables may extend up to 32K in size, but in discrete steps only. Wavetables must begin on a page boundary ($0C00, $0D00, and so forth). Table 5-7 shows the possible sizes of wavetables. Unused locations within a wavetable should begin with a minimum of eight zeros. Otherwise, the oscillator will not halt when it encounters these bytes and will interpret them as data.
2–0	–	Address bus resolution: The wavetable is played back by using every byte as data, or only intermittent bytes, as desired. The address resolution bits determine whether or not every byte is used during playback. Figure 5-7 shows how these bits effect wavetable scanning.

■ **Table 5-7** Wavetable size determination

Bit			
5	4	3	Table size
0	0	0	256
0	0	1	512
0	1	0	1024
0	1	1	2048
1	0	0	4096
1	0	1	8192
1	1	0	16384
1	1	1	32768

The Wavetable Pointer registers ($80–$9F)

This set of registers contains the beginning page number of the oscillator's wavetable. All wavetables must begin on a page boundary (that is, the first byte of the page) and cannot wrap around to low memory addresses. The value in this register is used in the final address calculation.

Figure 5-7 shows how the final address that is used to obtain the next wavetable data byte is calculated. The bits from the accumulator and the Wavetable Pointer register are concatenated in a varying ratio depending on the table size and address bus resolution parameters in the Wavetable Size register. The final address is used by the DOC to fetch the next wavetable data byte for conversion to an analog value by the digital-to-analog converter.

Making sound with the DOC

The oscillators in the DOC are not oscillators in the traditional sense; they do not generate a signal or tone. Instead, each oscillator acts as an address generator, pointing to a data byte in Sound RAM that, together with others, represents a signal that is to be generated. The data byte is then converted to an analog voltage by the digital-to-analog converter. These sequential series of voltages make up the output signal, which is then filtered and amplified and used to drive a speaker.

The data residing in the Sound RAM can be placed there either byte by byte, by manually building the wavetable (trial-and-error method works best here; try a sound, and modify it as you prefer), or by filling memory with a digitized input signal.

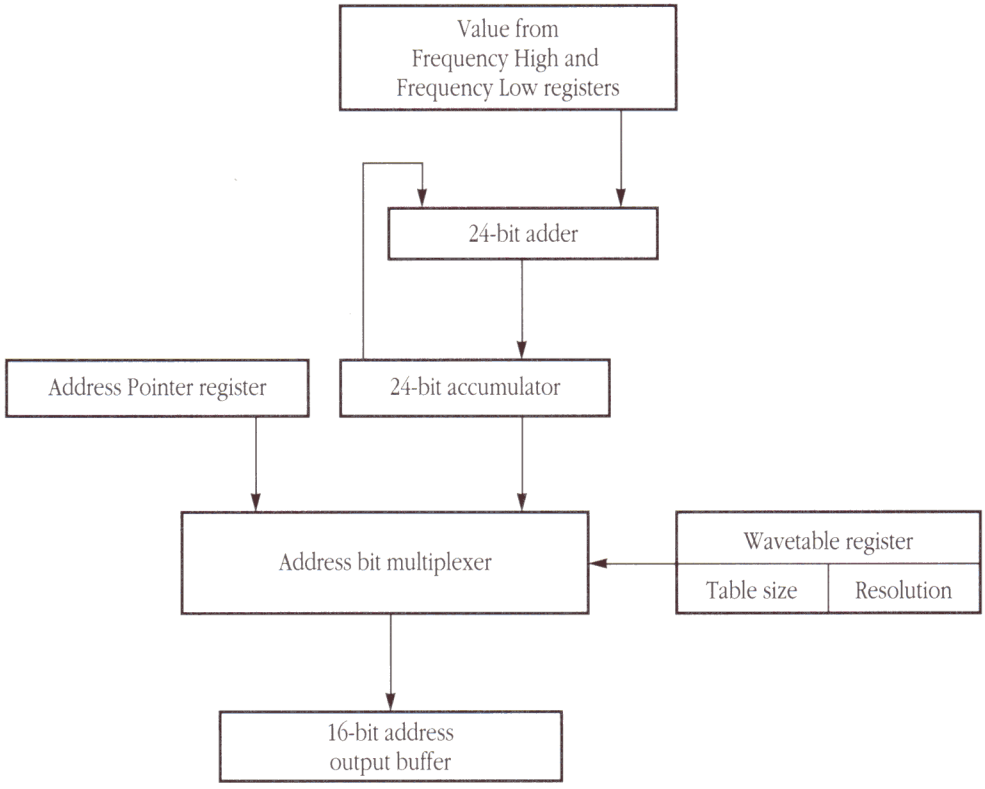
The waveshape of the signal is determined by the actual values of the data bytes that make up the wavetable. The pitch of the signal is determined by the speed with which the wavetable is scanned by the DOC. This scan rate is the value contained in the Wavetable Size register of each oscillator, and is arrived at by using several factors. Figure 5-8 shows the process used to calculate the scan rate. As shown in Figure 5-7 and Figure 5-8, each address's calculation is determined by the Frequency High and Frequency Low registers and the Wavetable Size register (made up of table size and address bus resolution data) associated with each oscillator.

■ **Figure 5-7** Final address calculation in the Wavetable Size register

Final address

	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Resolution R2	R1	R0
Table size	Pointer register bits								Accumulator bits										
256	P7	P6	P5	P4	P3	P2	P1	P0	A23	A22	A21	A20	A19	A18	A17	A16	1	1	1
									A22	----------		through		----------		A15	1	1	0
									A21	----------		through		----------		A14	1	0	1
									A20	----------		through		----------		A13	1	0	0
									A19	----------		through		----------		A12	0	1	1
									A18	----------		through		----------		A11	0	1	0
									A17	----------		through		----------		A10	0	0	1
									A16	----------		through		----------		A9	0	0	0
512	P7	------	through	-------	P1				A23	----------		through		----------		A15	1	1	1
									•			•				•	•	•	•
									•			•				•	•	•	•
									•			•				•	•	•	•
									A16	----------		through		----------		AB	0	0	0
1024	P7	----	through	-----	P2				A23	----------		through		----------		A14	1	1	1
									•			•				•	•	•	•
									•			•				•	•	•	•
									•			•				•	•	•	•
									A16	----------		through		----------		A7	0	0	0
2048	P7	---	through	---	P3				A23	----------		through		----------		A13	1	1	1
									•			•				•	•	•	•
									•			•				•	•	•	•
									•			•				•	•	•	•
									A16	----------		through		----------		A6	0	0	0
4096	P7	through		P4					A23	----------		through		----------		A12	1	1	1
									•			•				•	•	•	•
									•			•				•	•	•	•
									•			•				•	•	•	•
									A16	----------		through		----------		A5	0	0	0
8192	P7	P6	P5						A23	----------		through		----------		A11	1	1	1
									•			•				•	•	•	•
									•			•				•	•	•	•
									•			•				•	•	•	•
									A16	----------		through		----------		A4	0	0	0
16384	P7	P6							A23	----------		through		----------		A10	1	1	1
									•			•				•	•	•	•
									•			•				•	•	•	•
									•			•				•	•	•	•
									A16	----------		through		----------		A3	0	0	0
32768	P7								A23	----------		through		----------		A9	1	1	1
									•			•				•	•	•	•
									•			•				•	•	•	•
									•			•				•	•	•	•
									A16	----------		through		----------		A2	0	0	0

■ **Figure 5-8** Generating the sound addresses

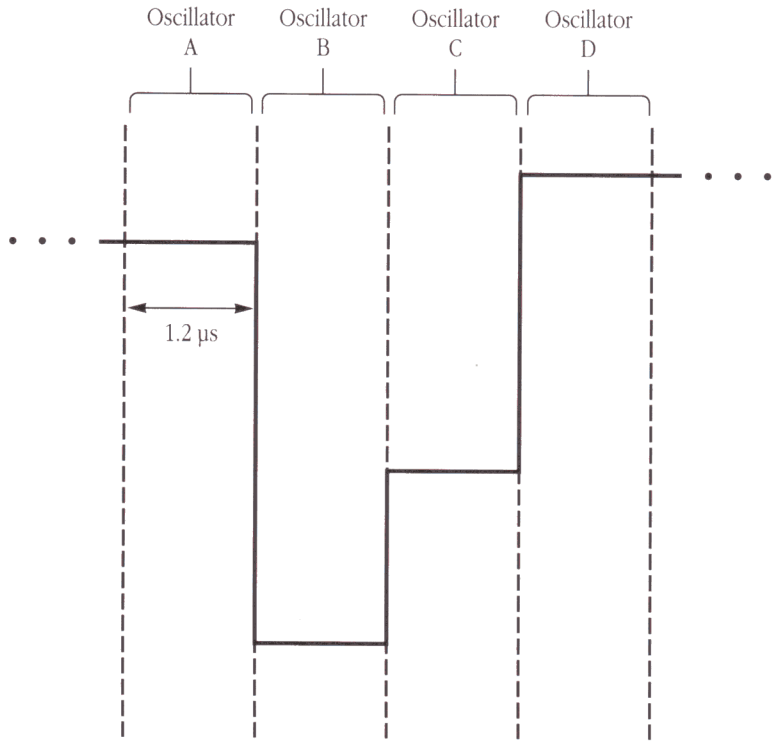


The address calculation is performed like this: Each time an oscillator is updated with a new address, the 16-bit value from the Frequency High and Frequency Low registers is added to the 24-bit accumulator by the adder. The 24-bit result is also passed to the address bit multiplexer and is used to form the final 16-bit Sound RAM address. The resolution value in the Wavetable Size register determines which of the 16 bits within the accumulator is used in the address calculation. The table-size value in the Wavetable Size register also determines how many accumulator bits will be used to calculate the final address.

The 32 oscillators are time-domain multiplexed, that is, the DOC services each oscillator in its turn. With all oscillators enabled, the DOC takes approximately 38 microseconds to service all 32. Figure 5-9 shows an example of a signal resulting from the combined output of 4 oscillators. This signal is then filtered, or smoothed, to remove the high-frequency components, leaving only waveforms in the audio-frequency range.

Digitized soundwaves are built by using consecutive data bytes (known collectively as a wavetable) in dedicated Sound RAM. Each of the 32 oscillators reads these bytes in sequential order at a speed that is programmable. This speed determines the frequency at which the waveform is reproduced, while the actual data in RAM determine the shape of the output waveform. The volume for each oscillator is also programmable.

■ **Figure 5-9** Combined output of time-domain multiplexed oscillators



Sound input and output specifications

The 7-pin molex connector is used for sound input and output to and from the Apple IIGS. The electrical specifications for these inputs and outputs are listed in Table 5-8.

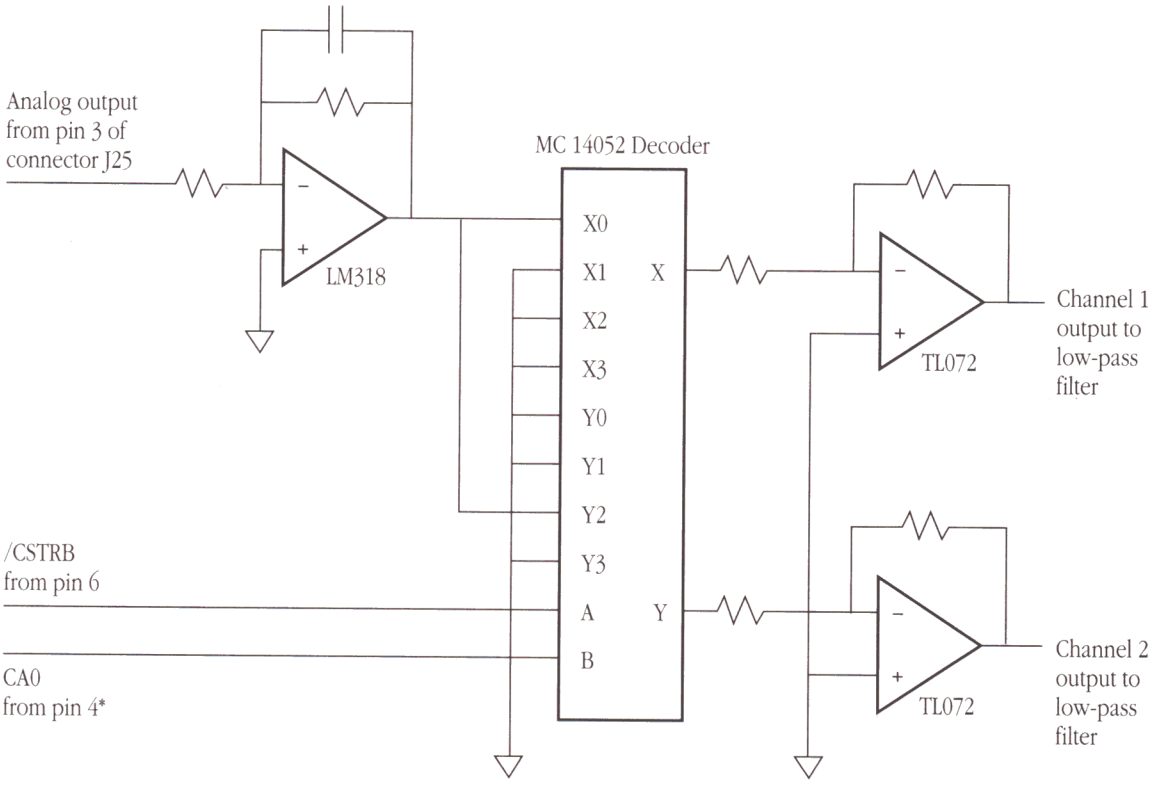
■ **Table 5-8** Sound input and output electrical specifications for connector J-25

Signal	Pin	Maximum	Units
A/D converter input	1	2.5	Volts peak-to-peak, full-scale conversion
Input impedance	1	3,000	ohms
Analog GND	2	–	–
Analog output	3	–5 to +5	Volts peak-to-peak
Output load	3	10,000	ohms, minimum
Channel address 0	4	1	LS TTL load
Channel address 1	5	1	LS TTL load
Channel strobe*	6	1	LS TTL load
Channel address 2	7	1	LS TTL load

* Channel strobe goes low when the channel address is valid.

Figure 5-10 shows an example of a demultiplexer circuit that can be designed to produce stereo (two-channel) sound, using the analog output from the DOC. The sound output at pin 3 will be demultiplexed according to addresses supplied by pins 4, 5, and 7. These addresses are those specified by each Oscillator Control register. A more complex circuit could handle as many as eight unique sound channels. For correct sound-channel demultiplexing, be sure to set the channel bits in each Oscillator Control register to the appropriate value.

■ **Figure 5-10** A two-channel demultiplexer circuit



Further reading

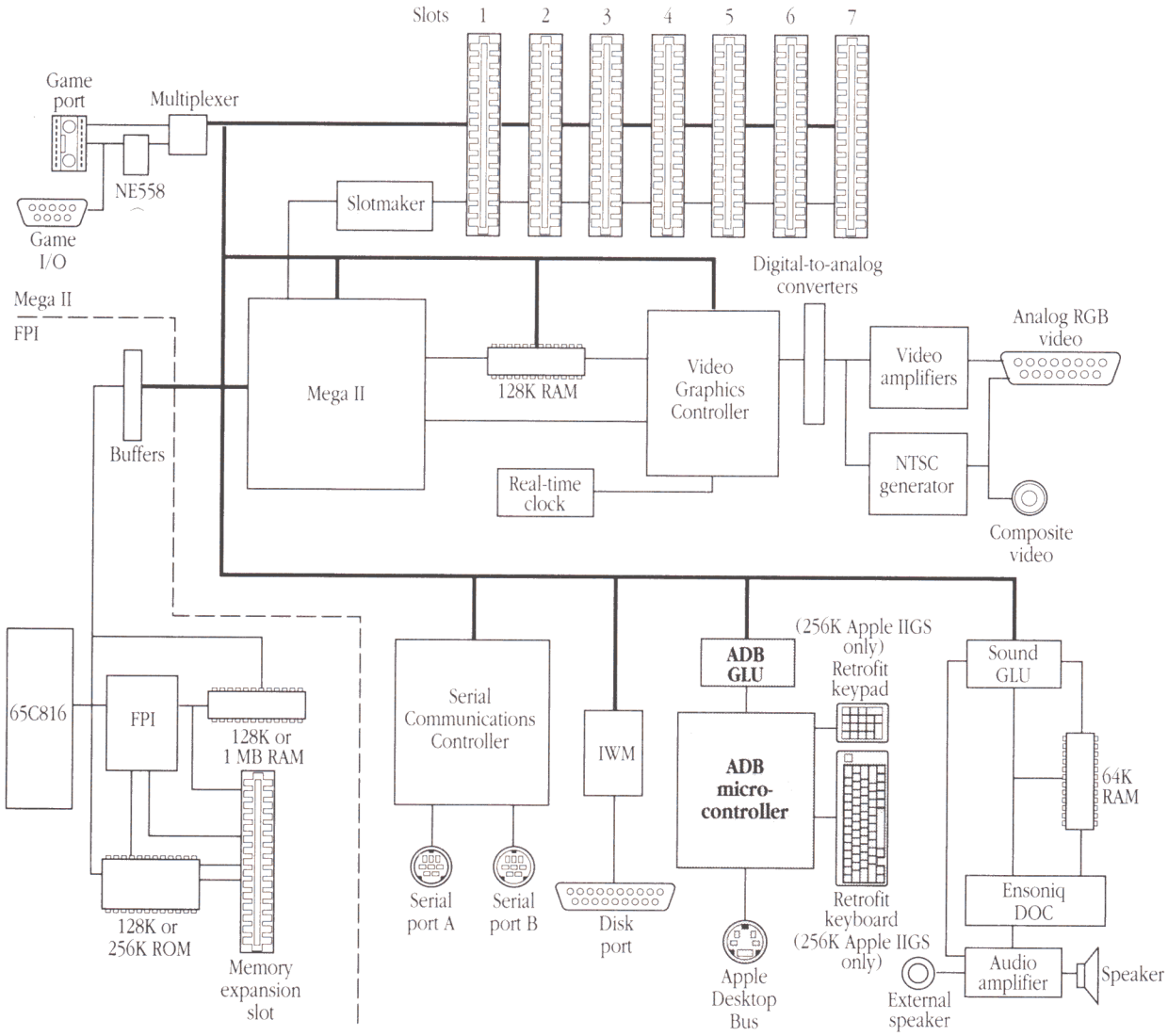
To learn more about synthesized music and sound, you may want to read this book:

Chamberlin, Hal. *Musical Applications of Microprocessors.* Hasbrouck Heights, NJ, Berkeley, CA: Hayden Books, 1985.

Chapter 6 **The Apple Desktop Bus**

The Apple Desktop Bus (ADB) is a method for connecting input devices (such as a keyboard or a mouse) with the Apple IIGS computer. The Apple Desktop Bus consists of the ADB GLU, the ADB microcontroller chip, and the Apple Desktop Bus cables. Figure 6-1 shows the relationship of the ADB components in the Apple IIGS computer.

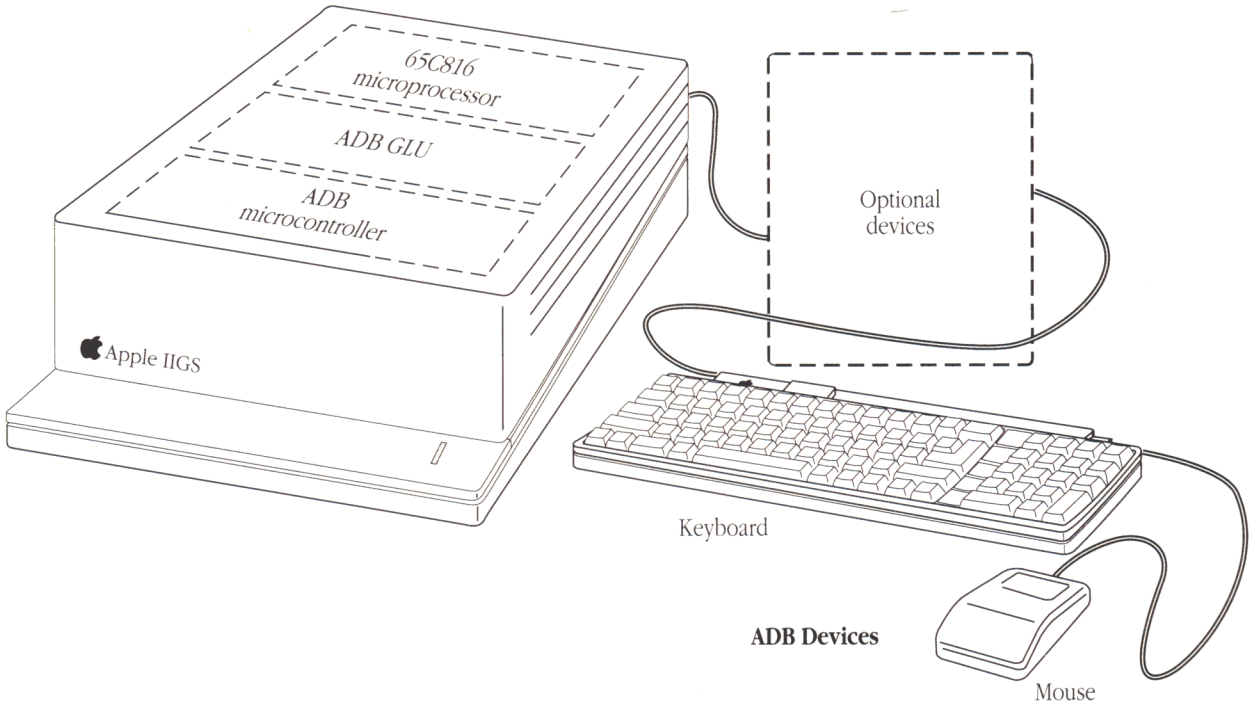
■ **Figure 6-1** ADB components in the Apple IIGS



The ADB microcontroller controls devices on the bus by receiving commands from the 65C816 microprocessor, and then sending appropriate ADB commands to and receiving data from the input devices on the bus. Microcontroller commands (those received from the 65C816) are located in ROM. Figure 6-2 shows the relationship of ADB components in the host computer to the bus devices (keyboards, mouse devices, and so on).

For the remainder of this chapter, the computer will be referred to as the host, and the input devices (for example, a keyboard or a mouse) connected to the bus as devices.

■ **Figure 6-2** ADB components



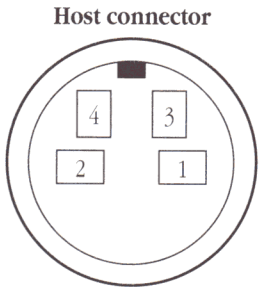
◆ *Note:* To keep compatibility with future Apple II products using the ADB, use the Apple Desktop Bus Tool Set in ROM. Directly accessing some of the ADB registers may cause the system to crash. (For more information on the ADB Tool Set, refer to the *Apple IIGS Toolbox Reference*.)

The input bus

All input devices share the input bus with the host. This bus consists of a 4-wire cable and uses 4-pin mini-DIN jacks at the host and at each device. Figure 6-3 shows the pin configuration of the ADB connectors and Table 6-1 lists the description of each pin. ADB devices may use the +5-volt power supplied by the bus, but must not draw more than 500 mA

total for all devices. All devices are connected in **parallel,** using the signal, power, and ground wires. Cables should be no longer than 5 meters, and total cable capacitance should not exceed 100 picoFarads per meter.

■ **Figure 6-3** Mini-DIN connector pin configuration used in the ADB



■ **Table 6-1** Pin assignments of the ADB connectors

Pin	Description
1	Data
2	Reserved
3	+5-volt **power supply** at 500 mA for all devices
4	Ground

The keyboard

The keyboard uses the Apple Desktop Bus to communicate with the processor. All input devices are connected to the ADB and are controlled by the keyboard microcontroller. This controller supports reading of the keyboard by standard Apple II application programs.

The Apple IIGS keyboard has automatic repeat, which means that if you press any key longer than you would during normal typing, the character code for that key will be sent continuously until you release the key. The speed at which the key repeats, and the delay before it repeats, may be set from the Control Panel. (See the section "Modifier Key Register," later in this chapter, for more information.)

Any number of modifier keys may be held down and an additional key pressed will be recognized. This is called *n-key rollover.* The alphanumeric keys do not have this ability. Any two alphanumeric keys pressed simultaneously will be recognized. A third key pressed will not be recognized. This is called *2-key lockout.*

Apple IIGS computers manufactured for sale outside the United States have slightly different standard keyboard arrangements. The different keyboards are shown in Appendix B.

ADB and the upgraded Apple IIe

The Apple IIGS can be used with a standard Apple IIe keyboard when an Apple IIe is upgraded with an Apple IIGS logic board. The Apple IIe keyboard is then used as the primary input device, rather than the ADB keyboard. The Apple IIe keyboard is read just like the standard keyboard in an Apple IIe, and is not subject to the Apple Desktop Bus protocols.

Reading the keyboard

The ADB microcontroller generates all 128 ASCII codes, so all the special character codes in the ASCII character set are available. Application programs obtain character codes from the keyboard by reading a byte from the keyboard data location shown in Table 6-2 (the Keyboard Data register).

■ **Table 6-2** Keyboard Data locations

Location		
Hex	**Dec**	**Description**
$C000	49152	Keyboard Data register and strobe
$C010	49168	Any-key-down flag and clear-strobe switch

Your programs can get the code for the last key pressed by reading the Keyboard Data register located at $C000. Table 6-2 gives this location in two different forms: The hexadecimal value, indicated by a preceding dollar sign ($), is used in assembly language; the decimal value is used in Applesoft BASIC.

The low-order seven bits of the byte at the keyboard data location contain the character code; the high-order bit of this byte is the strobe bit, described below.

Your program can find out whether any key is down, except the Reset, Control, Shift, Caps Lock, Command, and Option keys, by reading from location 49152 ($C000). The high-order bit (bit 7) of the byte you read at this location is called *any-key-down;* it is 1 if a key is down and 0 if no key is down. The value of this bit is 128; if a BASIC program gets this information with a PEEK, the value is 128 or greater if any key is down, and less than 128 if no key is down.

The strobe bit is the high-order bit of the Keyboard Data register. After any key has been pressed, the strobe bit is high. It remains high until you reset it by reading or writing at location $C010. This location is a combination flag and soft switch; the flag tells whether any key is down, and the switch clears the strobe bit. In this case, it doesn't matter whether the program reads or writes, and it doesn't matter what data the program writes: The only action that occurs is the resetting of the keyboard strobe. (See the *Apple IIGS Firmware Reference* for information on firmware for reading the keyboard.)

- *Note:* Any time you read the any-key-down flag, you also clear the keyboard strobe. If your program needs to read both the flag and the strobe, it must read the strobe bit first.

After the keyboard strobe has been cleared, it remains low until another key is pressed. Even after you have cleared the strobe, you can still read the character code at the keyboard location. The data byte has a different value, because the high-order bit is no longer set, but the ASCII code in the seven low-order bits is the same until another key is pressed. Appendix D contains the ASCII codes for the keys on the keyboard.

There are several special function keys that do not generate ASCII codes. For example, pressing the Control, Shift, or Caps Lock key directly alters the character codes produced by the other keys. In the Apple IIGS, you can determine the state of these modifier keys by reading the modifier key register within the ADB microcontroller, described later in this chapter.

The Control-Command-Reset key combination is different from all other key combinations on the ADB keyboard in that it generates a special key code. When the ADB microcontroller detects the reset key combination, the program currently running in memory is halted and the system asserts the RESET line and restarts the computer. This restarting process is called the *reset routine.* (To read about the reset routine, see the *Apple IIGS Firmware Reference.*)

The ADB microcontroller

The ADB microcontroller is an intelligent controller IC that oversees the Apple Desktop Bus. The M50740 microcontroller uses a superset of the 6502 instruction set, and contains 96 bytes of RAM and 3K of ROM. The ADB microcontroller operates asynchronously, transmitting commands and data to and receiving data from the bus devices. To ensure compatibility with future versions of ADB, use the ADB commands in the ROM toolbox to communicate with the ADB. To find out how to use the toolbox in the system ROM, see the *Apple IIGS Toolbox Reference.*

The ADB GLU

The ADB General Logic Unit (GLU) works together with the ADB microcontroller to form an intelligent input-device interface. The ADB GLU, located on the main logic board, uses two independent data buses that serve as a communications interface between the ADB microcontroller and the system bus. This interface is accomplished by using multiple internal read/write registers to store keyboard data, key modifiers, mouse X and Y coordinates, command data, and status information.

The ADB GLU registers

The ADB GLU contains five data and control registers. These registers are used for storing keyboard data and commands, key modifiers, mouse X and Y coordinates, and status information. The registers are

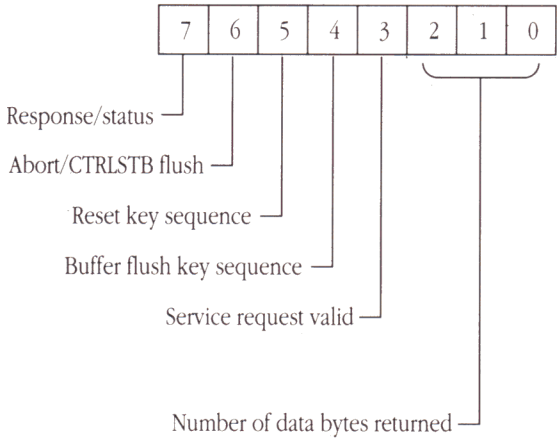
- ADB Command/Data register ($C026)
- Keyboard Data register ($C000)
- Modifier Key register ($C025)
- Mouse Data register ($C024)
- ADB Status register ($C027)

All registers except the status registers have a status flag that is set to 1 when the register is written to, and cleared to 0 when the register is read. Each of the data registers also has an interrupt flag that generates system interrupts, if interrupts are enabled. These status and interrupt flags are located in the status register. These registers are described in the following sections.

ADB Command/Data register

The ADB Command/Data register is a dual-function register used to communicate with ADB devices. To send a command to a device on the bus, write the command byte to this register at address $C026. To check the status of an ADB device, read this register at the same address. Figure 6-4 shows the format of the ADB Command/Data register when it is read. Table 6-3 gives a description of each bit.

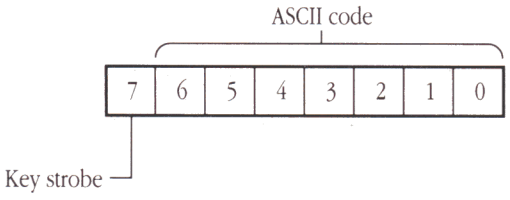
■ **Figure 6-4** ADB Command/Data register at $C026



Keyboard Data register

The Keyboard Data register contains the ASCII value of the last key pressed on the keyboard. The high bit is set when a new key has been pressed. Figure 6-5 shows the format of this register. Table 6-4 gives a description of each bit.

■ **Figure 6-5** Keyboard Data register at $C000



■ **Table 6-3** Bits in the ADB Command/Data register

Bit	Value	Description
7	1	When this bit is 1, the ADB microcontroller has received a response from an ADB device previously addressed.
	0	No response.
6	1	When this bit is 1, and only this bit in the register is 1, the ADB microcontroller has encountered an error and has reset itself. When this bit is 1 and bit 4 is also 1, the ADB microcontroller should clear the key strobe (bit 7 in the Keyboard Data register at $C000).
	0	–
5	1	When this bit is 1, the Control, Command, and Reset keys have been pressed simultaneously. This condition is usually used to initiate a cold **start up.**
	0	Reset key sequence has not been pressed.
4	1	When this bit is 1, the Control, Command, and **Delete keys** have been pressed simultaneously. This condition will result in the ADB microcontroller's flushing all internally buffered commands.
	0	Buffer flush key sequence has not been pressed.
3	1	When this bit is 1, a valid service request is pending. The ADB microcontroller will then poll the ADB devices and determine which has initiated the request.
	0	No service request pending.
2–0	–	The number of data bytes to be returned from the device is listed here.

■ **Table 6-4** Bits in the Keyboard Data register

Bit	Value	Description
7	–	This bit is 1 when a key has been pressed, and indicates that the ASCII value in bits 6 through 0 are valid. This bit must be cleared after reading the data by reading or writing to any address from $C010 through $C01F.
6–0	–	ASCII data from the keyboard.

Modifier Key register

The Modifier Key register contains bits that reflect the status of the modifier keys. These keys include the standard Shift, Control, Command, and Caps Lock keys, as well as keys on the numeric keypad. Figure 6-6 shows the format of this register. Table 6-5 gives a description of each bit.

■ **Table 6-5** Bits in the Modifier Key register

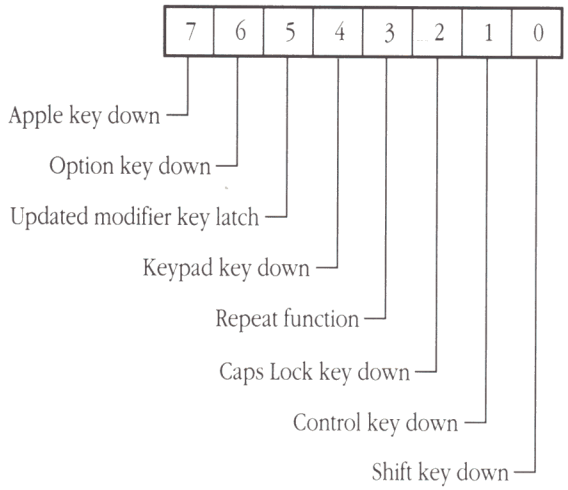
Bit	Value	Description
7	1	When this bit is 1, the Command key has been pressed.
	0	When this bit is 0, the Command key has not been pressed.
6	1	When this bit is 1, the Option key has been pressed.
	0	When this bit is 0, the Option key has not been pressed.
5	1	When this bit is 1, the modifier key latch has been updated, but no key has been pressed.
	0	–
4	1	When this bit is 1, a numeric keypad key has been pressed.
	0	When this bit is 0, a numeric keypad key has not been pressed.
3	1	When this bit is 1, a key is being held down.
	0	When this bit is 0, no key is being held down.
2	1	When this bit is 1, the Caps Lock key has been pressed.
	0	When this bit is 0, the Caps Lock key has not been pressed.
1	1	When this bit is 1, the Control key has been pressed.
	0	When this bit is 0, the Control key has not been pressed.
0	1	When this bit is 1, the Shift key has been pressed.
	0	When this bit is 0, the Shift key has not been pressed.

Mouse Data register

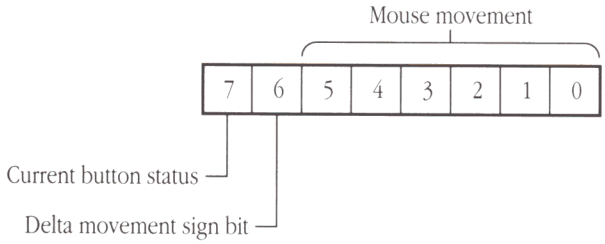
The ADB mouse, when moved, generates movement data that are transmitted to the host. These data, along with the mouse button status, are available in the Mouse Data register. Figure 6-7 shows the format of this register. Table 6-6 gives a description of each bit.

◆ *Note:* Read this register only twice in succession. The first read returns Y-coordinate data, and the second read returns X-coordinate data. Reading this register an odd number of times will result in a random movement of the cursor.

■ **Figure 6-6** Modifier Key register at $C025



■ **Figure 6-7** Mouse Data register at $C024



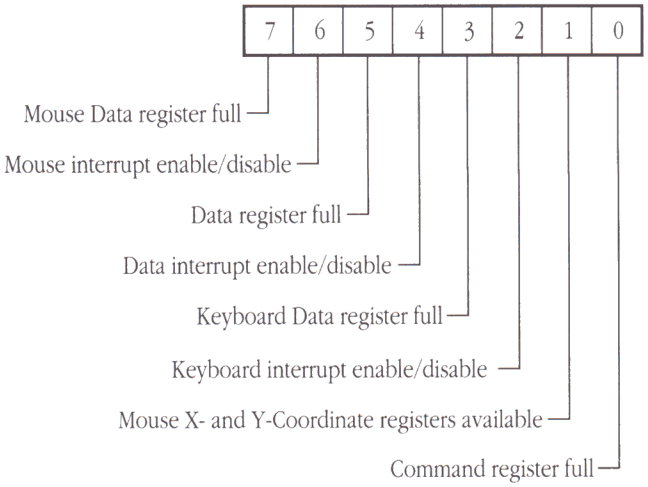
■ **Table 6-6** Bits in the Mouse Data register

Bit	Value	Description
7	1	Current mouse status: When this bit is 1, the mouse button is up.
	0	When this bit is 0, the mouse button is down.
6	1	Delta sign bit: if this bit is 1, the delta value is negative.
	0	If this bit is 0, the delta value is positive.
5–0	–	The relative mouse movement data are returned here.

ADB Status register

The ADB Status register, located at $C027, contains flags that relate to mouse and keyboard data and status. Figure 6-8 shows the format of the ADB Status register. Table 6-7 gives a description of each bit.

■ **Figure 6-8** ADB Status register at $C027



■ **Table 6-7** Bits in the ADB Status register

Bit	Value	Description
7	1	When this bit is 1, the Mouse Data register at $C024 is full (read-only bit)
	0	When this bit is 0, the Mouse Data register is empty.
6	1	When this bit is 1, the mouse interrupt is enabled (read/write bit), and an interrupt is generated when the Mouse Data register contains valid data.
	0	When this bit is 0, the mouse interrupt is disabled.
5	1	When this bit is 1, the Command/Data register contains valid data (read-only bit)
	0	When this bit is 0, the Command/Data register contains no valid data.
4	1	When this bit is 1, the Command/Data interrupt is enabled (read/write bit), and an interrupt is generated when the Command/Data register contains valid data.

■ **Table 6-7** Bits in the ADB Status register (Continued)

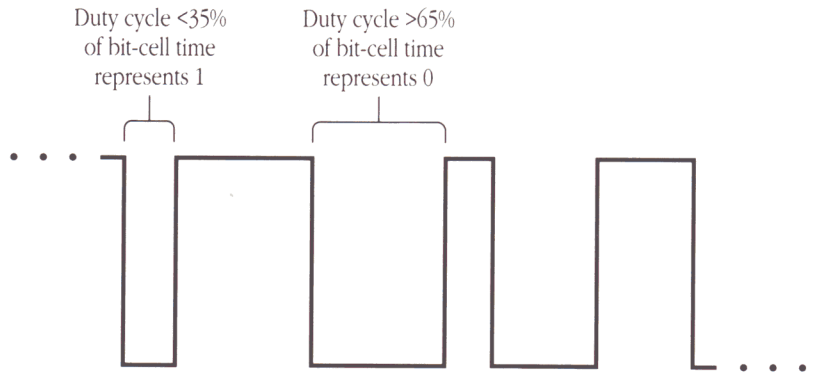
Bit	Value	Description
	0	When this bit is 0, the data interrupt is disabled.
3	1	When this bit is 1, the Keyboard Data register is full (read-only bit). This bit is set when the Keyboard Data register is written to.
	0	When this bit is 0, the Keyboard Data register is empty. This bit is cleared when the Keyboard Data register is read or the ADB Status register is read.
2	1	When this bit is 1, the keyboard data interrupt is enabled (read/write bit), and an interrupt is generated when the Keyboard Data register contains valid data
	0	When this bit is 0, the keyboard data interrupt is disabled.
1	1	When this bit is a 1, mouse X-coordinate data is available in the Mouse Data register (read-only bit)
	0	When this bit is a 0, mouse Y-coordinate data is available in the Mouse Data register.
0	1	Command register full (read-only bit). This bit is set when the Command/Data register is written to.
	0	Command register empty. This bit is cleared when the Command/Data register is read.

Bus communication

The host communicates with the devices on the bus by sending commands and/or data to a device. A device may respond to commands by sending data back to the host. This form of communication uses strings of bits, each making up a packet. A data transfer or transaction consists of a complete communication between the host and a device; for example, it may be a command packet sent by the host to request data from a device, followed by a data packet sent from the device to the host. Depending upon the type of command sent by the host, the device will send two to eight data bytes back.

Figure 6-9 shows how duty-cycle modulation represents bits on the bus. A low period of less than 35 percent of the bit-cell time is interpreted as a 1. A low period of greater than 65 percent of the bit-cell time is interpreted as a 0.

■ **Figure 6-9** Bit representation via duty-cycle modulation



Commands and global signals

Communications on the bus are of two types: Commands that are sent by the host to a specific device, and global signals that instruct all devices to perform a predefined function. The four commands in the command set are

- Listen
- Talk
- Device Reset
- Flush

Certain signals, however, are neither commands nor data transactions. These are special signals that the host or a device uses to broadcast status globally to all devices on the bus. There are four signals in the global signal group:

- Attention
- Sync
- Global Reset
- Service Request

The following paragraphs describe each of these device commands and global signals.

Transactions

A transaction is a bus communication between the host and a device, or vice versa. A command (Listen, Talk, Device Reset, Flush) initiates a transaction. A transaction consists

of a command packet sent by the host followed by a data packet of several bytes by either the host or a device. A command packet consists of

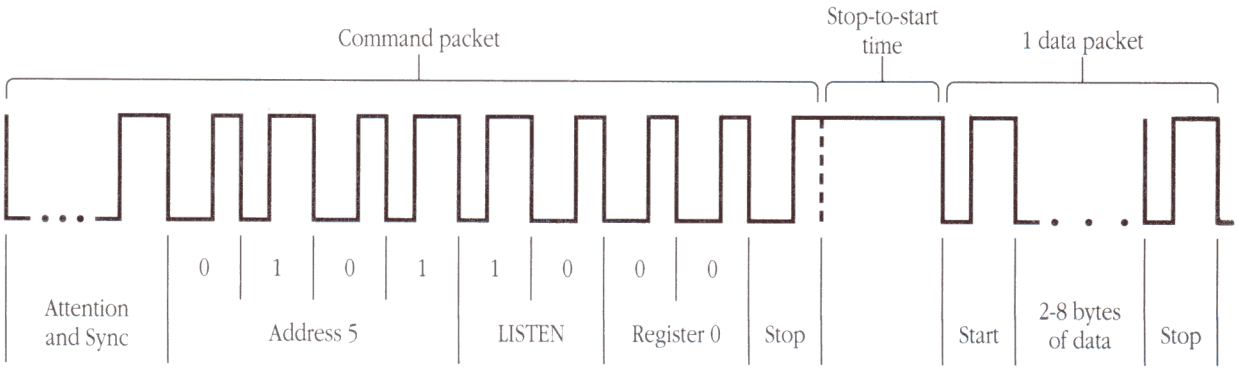
- an attention/sync signal
- one command byte
- one stop bit

A data packet consists of

- a start bit
- two to eight (8-bit) data bytes
- one stop bit

Figure 6-10 shows a typical transaction on the Apple Desktop Bus, consisting of a command packet followed by a data packet.

- **Figure 6-10** A typical transaction



To indicate the end of the command packet, the command ends with a stop bit after the last command bit-cell. Then the transaction is complete and the host releases its control of the bus. (The bus is always floating in a high state until a device or the host initiates a transaction.) The identical scheme is used for the data packet.

Note that the stop-bit–to–start-bit time is critical; the host requires this minimum turnaround delay to allow for internal overhead. Table 6-8 lists the timing maximum and minimum parameters for ADB signals.

■ **Table 6-8** ADB timing specifications

Parameter	Minimum	Maximum	Unit
Bit-cell time	70	130	microseconds
"0" low time	60	70	percent of bit-cell time
"1" low time	30	40	percent of bit-cell time
Attention	560	1040	microseconds
Global Reset	2.8	5.2	milliseconds
Sync	60	70	percent of bit-cell time
Service request	140	260	microseconds
Stop bit to start bit time	140	260	microseconds

Commands

A command is sent by the host to one specific device address. Only the host can send commands. There are four commands: The Talk command is used to acquire data from a device; the Listen command is used to place data in a device register or to get a device to perform some new function; the Device Reset command reinitializes the device; and the Flush command is device specific.

A command is an 8-bit word that has a specific syntax (as shown in Table 6-9):

- A 4-bit field that specifies the address of the desired device. (The addresses range from 0 through 15 [address bits 3 through 0].)
- A 4-bit command and register address code.

◆ *Note:* To allow for future expansion of the command structure, Apple Computer, Inc. has reserved a group of instructions that are currently treated as no-ops (no operation performed). Use of commands not listed will result in possible incompatibility with future Apple products.

Talk

The Talk command is a request of the device to transmit the contents of one of the device's registers (0 through 3). When the host addresses a device to Talk, the device must respond with data before the host times out (does not receive data within the specified time). The selected device performs its data transaction and releases the bus.

■ **Table 6-9** Command byte syntax

	Device address				Command code		Register code		
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Command**
	x	x	x	x	0	0	0	0	Send Reset
	A_3	A_2	A_1	A_0	0	0	0	1	Flush
	x	x	x	x	0	0	1	0	Reserved
	x	x	x	x	0	0	1	1	Reserved
	x	x	x	x	0	1	x	x	Reserved
	A_3	A_2	A_1	A_0	1	0	r_1	r_0	Listen
	A_3	A_2	A_1	A_0	1	1	r_1	r_0	Talk

Note: x = ignored; r = register number; A3 through A0 = bits 11 through 8 of register 3.

Listen

The Listen command is a request of the device to store the data being transmitted in one of the internal registers (0 through 3). When the host addresses a device to Listen, the device receives the next data packet from the host and places it in the appropriate register. After the stop bit following the data is received, the transaction is complete and the host releases the bus. If the addressed device detects another command on the bus before it receives any data, the original transaction is immediately considered complete.

Send Reset

When a device receives a Send Reset command, it will clear all pending operations and data, and will initialize to the power-on state. The Send Reset is not device specific; it is sent to all devices on the bus simultaneously.

Flush

This is a device-specific command that will clear all pending commands from the device.

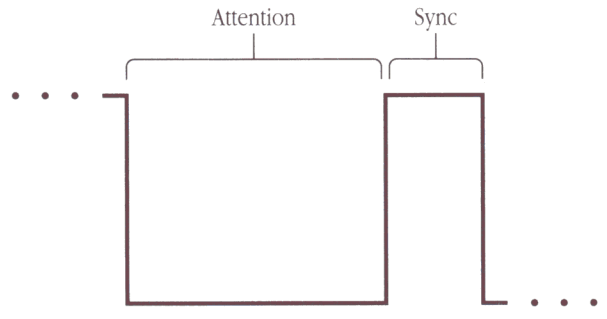
Broadcast signals

Broadcast signals are transmitted on the bus but do not address any specific device. This way, all devices receive the signals and respond simultaneously. There are four broadcast signals: Attention, Sync, Global Reset, and Service Request.

Attention and Sync

The start of every command is indicated by a long low Attention signal that the host sends on the bus. This signal is followed by a short high Sync pulse that signals the beginning of the initial bus timing. The falling edge of the sync pulse is used as a timing reference, after which the first command bit follows. Figure 6-11 shows the format of the Attention and Sync signals.

■ **Figure 6-11** Attention and Sync signals



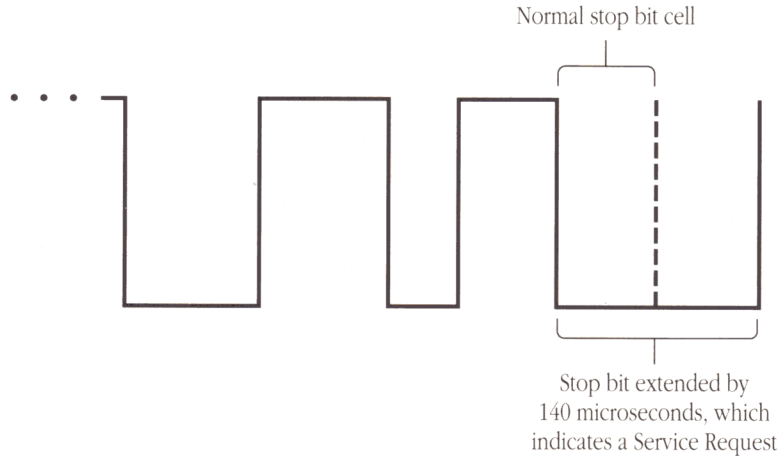
Global Reset

When the bus is held low for a minimum of 2.8 milliseconds, a Global Reset is initiated. Only the host may issue this signal, which forces all bus devices to reset. Note that the Global Reset signal differs from the Device Reset command: The Device Reset command addresses one specific device, and resets that device. The Global Reset signal is received by all devices and forces all devices to reset.

Service Request

A Service Request signal is used to inform the host that a device requires service, as, for example, when there are data to send to the host. Only a device can issue a Service Request. Following any command packet, a requesting device can signal a Service Request by holding the bus low during the low portion of the stop bit of the last command transaction. Holding the bus low in this manner lengthens the stop by a minimum of 140 microseconds beyond its normal bit-cell boundary. Figure 6-12 shows the format of the Service Request signal.

■ **Figure 6-12** Service Request



A device will signal a Service Request repeatedly until it is served. When a device has requested service (at which point the host does not know *which* device sent the request), the host will poll each of the devices by sending a Talk register 0 command (discussed later in this chapter), beginning with the last active device. Only the device that has data to send (the device that sent the Service Request) will respond to the Talk command.

When the host commands the requesting device to Talk, the device is considered served and does not send a Service Request signal again until it needs to be served again. The host can enable and disable the ability of a device to send a Service Request at any time. ADB mouse devices are prohibited by the Apple IIGS from issuing Service Requests. All other ADB devices may issue Service Requests, as long as the device has not been prohibited from sending a Service Request. (See description of register 3, later in this chapter.)

Error conditions

If the bus level remains low for a significant time period, all devices reset themselves and output a 1. If a command transaction is incomplete by staying high beyond the maximum bit-cell time, all devices ignore the command and wait for an Attention signal.

Apple Desktop Bus peripheral devices

All devices on the Apple Desktop Bus are slaves; only the host (the computer) may send commands. Devices transmit on the bus only after they have been requested by the host.

A device that receives a Talk command (and has data to send) sends the data and then releases control of the bus. If a device has been addressed but has no data to send, it does not respond and allows the host to time out (waiting for data, none arrives). The host may also send a Listen command to the addressed device followed by data in a separate packet.

Device registers

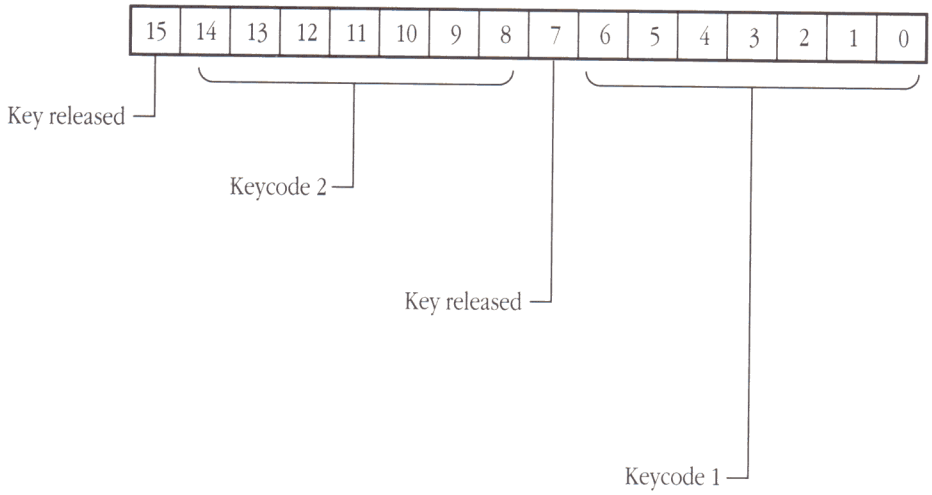
All devices have four locations to receive data. These are

- **Register 0**
  Talk: Data register, device specific
  Listen: Data register, device specific
- **Register 1**
  Talk: Data register, device specific
  Listen: Data register, device specific
- **Register 2**
  Talk: Data register, device specific
  Listen: Status or data, device specific
- **Register 3**
  Talk: Status information, including the device address handler
  Listen: Status information, including the device address handler

Register 0

Register 0 is a data register and contains data that will be sent to the host in response to a Talk register 0 command. Figures 6-13 and 6-14 show the format of register 0 as used in a keyboard and a mouse device, and Tables 6-10 and 6-11 describe each bit in these registers.

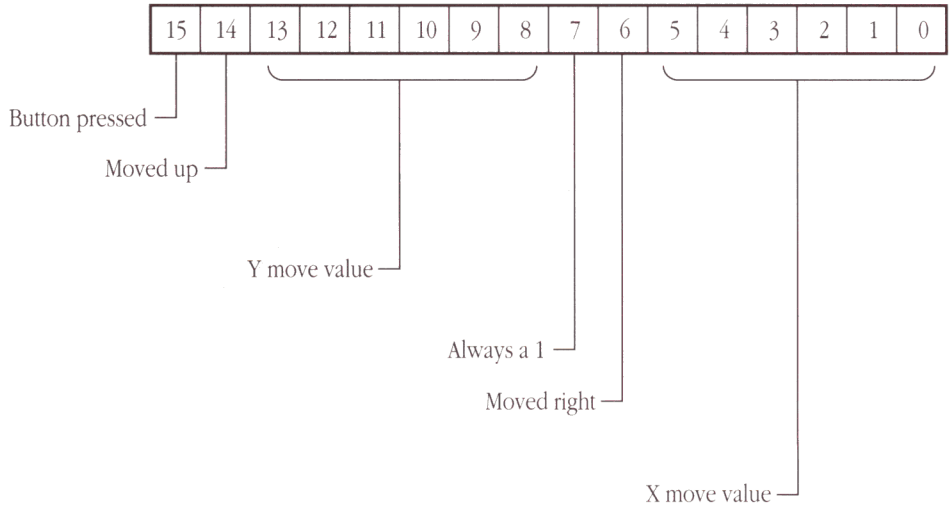
■ **Figure 6-13** Keyboard register 0



■ **Table 6-10** Bits in keyboard register 0

Bit	Value	Description
15	1	Key released indicator: When this bit is 1, the key represented by keycode 2 is up.
	0	When this bit is 0, the key represented by keycode 2 is down.
14–8	–	Keycode 2: The 7-bit ASCII value of the last key pressed.
7	1	Key released indicator: When this bit is 1, the key represented by keycode 1 is up.
	0	When this bit is 0, the key represented by keycode 1 is down.
6–0	–	Keycode 1: The 7-bit ASCII value of the last key pressed.

■ **Figure 6-14** Mouse register 0



■ **Table 6-11** Bits in mouse register 0

Bit	Value	Description
15	1	Button pressed indicator: When this bit is 1, the mouse button is up.
	0	When this bit is 0, the mouse button is down.
14	1	Movement indicator: When this bit is 1, the mouse has moved up along the Y axis.
	0	When this bit is 0, the mouse has moved down along the Y axis.
13–8	–	Y movement value: This field contains the value of the relative mouse movement along the Y (vertical) axis.
7	–	Always a 1.
6	1	Movement indicator: When this bit is 1, the mouse has moved left along the X axis.
	0	When this bit is 0, the mouse has moved right along the X axis.
5–0	–	X movement value: This field contains the value of the relative mouse movement along the X (horizontal) axis.

Register 1

Register 1, like register 0, is a data register, but it is device specific. The function of this register is not defined for use by the ADB protocol, but it may be used by the application program for any data function.

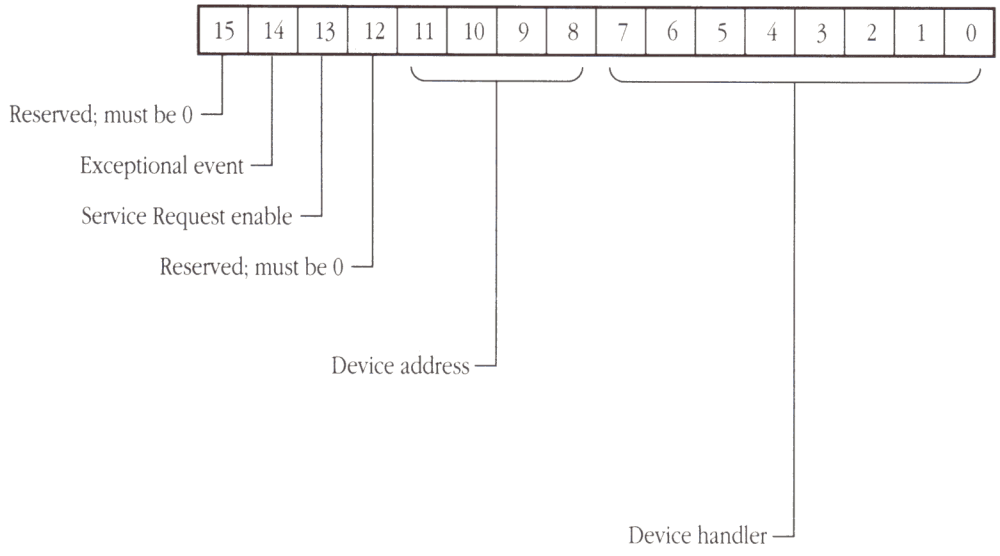
Register 2

This register is a data register and is device specific. In response to the Talk register 2 command, the device will send the contents of this register onto the bus. In response to the Listen register 2 command, the device will store the data sent to the device, as defined by the device's specification.

Register 3

This register is a status and command register that contains a handler code and the device address. The host may change the contents of this register with a Listen register 3 command. Figure 6-15 shows the format of register 3, and a description of each bit follows in Table 6-12.

■ **Figure 6-15** Device register 3



■ **Table 6-12** Bits in device register 3

Bit	Value	Description
15	–	Reserved; must be 0.
14	1	Exceptional event, device specific. In the ADB keyboard, this bit indicates that the Reset key has been pressed.
	0	No exceptional event has occurred.
13	1	Service Request enable: When this bit is 1, the device may transmit a Service Request. See the section "Service Request Enable/Disable," later in this chapter, for more details on this function.
	0	When this bit is 0, the device may not transmit a Service Request.
12	–	Reserved; must be 0.
11–8	–	Device address: This field contains the device's unique bus address. The possible addresses are listed in Table 6-14.
7–0	–	Device handler: This field contains the special handler code that defines the function of the device. See the next section, "Device Handlers," for a list of unique handlers.

Device handlers

Device handlers provide a means for a device to function in more than one manner. By sending a new handler ID, the device can be instructed to perform a new function.

There are two kinds of handlers: reserved, and all others. There are four reserved device handlers, listed in Table 6-13. Other handlers may be used for special device functions, but new device handlers defined by third-party developers must first be registered with Apple Computer, Inc.

Upon receiving a reserved device handler, the device will immediately perform the new function. The device will not store the reserved handler in register 3; only device-defined handler codes are stored. All unrecognized handlers are ignored.

Device addresses

Each peripheral device is preassigned a 4-bit command address, which identifies its device type. For example, all relative devices, such as a mouse, power up at address 3. Most devices have movable addresses. That is, the host can assign a new address to the device. The host

■ **Table 6-13** Reserved device handlers

Handler	Definition
$FF	Initiates a self-test in the device.
$FE	As Listen register 3 data, instructs the device to change the address field to the new address sent by the host if no collision has been detected.
$FD	As Listen register 3 data, instructs the device to change the address field to the new address sent by the host if the activator is pressed.
$00	As Listen register 3 data, instructs the device to change the address and enable fields to the new values sent by the host.
$00	As data sent in response to a Talk register 3 command, indicates that the device failed a self-test.

must assign a new address when two devices have the same default address (such as two mouse devices); one must be moved to a new address. A device will always default to its assigned address upon power-on or after it detects an ADB reset. Currently, eight addresses are predefined or reserved. The other eight addresses are available for movable devices. This means that ADB can support up to nine mouse devices, keyboards, or graphics tablets at the same time, each one with a unique address. Table 6-14 lists all the possible device addresses.

■ **Table 6-14** Device addresses

Address	Device class	Device type	Example
$00	Reserved	–	–
$01	Reserved	–	–
$02	Encoded devices	Movable	Keyboard
$03	Relative devices	Movable	Mouse
$04	Absolute devices	Movable	Graphics tablet
$05	Reserved	–	–
$06	Reserved	–	–
$07	Reserved	–	–
$08	Soft address	Movable	Any
$09	Soft address	Movable	Any
$0A	Soft address	Movable	Any
$0B	Soft address	Movable	Any
$0C	Soft address	Movable	Any
$0D	Soft address	Movable	Any
$0E	Soft address	Movable	Any
$0F	Soft address	Movable	Any

Collision detection

All devices must be able to detect collisions. If a device is attempting to output a bit and the data line is forced low by another device, it has lost a bit in collision with the other device. If another device sends data before the device is able to assert its start bit, it has lost a collision. The losing device should immediately stop transmitting and preserve the data that were being sent. A device sets an internal flag if it loses a collision.

- *Note:* Devices using internal clocks that operate within ±1 percent should attempt to assert their start bit at a random time within the limits of the bus turnaround time.

To aid in collision detection, the address field of register 3 is replaced with a random number in response to a Talk register 3 command. A device will change its device address to this new address as long as it has not detected a collision. A device that has detected a collision will not change its address during the next Listen register 3 command.

At the systems level, a host can change the addresses of normal devices by using this technique. By issuing a Talk register 3 command and following it with a Listen R3 command with a new address in bits 8 to 11 of the data packet, the host moves all devices that did not detect a collision to the new address. Typically, only one device will not detect a collision. This technique can be repeated at new addresses until the response to the Talk register 3 command is a time-out (no response). This process can be used to identify and relocate multiple devices of the same type after system initialization.

A normal device may have an optional *activator* on it. The activator can be a special key on a keyboard or a mouse button. At the application level, addresses can be changed by the host's displaying a message requesting a user to use the activator (hold down a key). By using the Listen register 3 command, the host can move the device with the activator pressed to a new address. This method can be used by an application program to identify and locate individual devices in multi-user applications. Also, certain reserved handlers are used to facilitate both address-changing methods.

Service Request enable/disable

It is possible to control the ability of a device to transmit a Service Request. To disable a device's ability to send a Service Request, set bit 12 in register 3 to 0; to enable it, set this bit to 1. This feature is useful in an application where the Service Request response time in a polled system is longer than desired. When only specific devices are required for an application, the others can be disabled.

1 MB Apple IIGS

The 1 MB Apple IIGS main logic board includes a redesigned ADB microcontroller. This IC has RAM expanded to 96 bytes, and ROM expanded to 4K. The new ROM code supports sticky keys and the ADB mouse functions. This information is provided here for developers who may have need to provide these functions in a third-party ADB device. For complete instructions on using sticky keys and ADB mouse, see the *Apple IIGS Owner's Guide*.

Sticky keys

The new ADB microcontroller provides a new feature useful to anyone who is limited to pressing only one key at a time. By enabling sticky keys, any combination of modifier keys (Shift, Command, Option, Control) can be achieved by pressing the keys sequentially rather than simultaneously. Table 6-15 lists these functions and the action required to implement each.

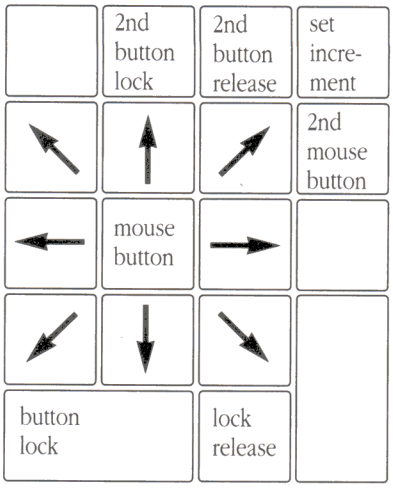
■ **Table 6-15** Sticky keys functions

Function	Action
Enable sticky keys	Press Shift key five times
Enable a modifier key	Press the modifier key once
Lock down modifier key	Press the modifier key twice
Disable a modifier key	Press the modifier key a total of three times
Disable sticky keys	Press Shift key five times

ADB mouse

The ADB microcontroller provided with the 1 MB Apple IIGS includes the ADB mouse feature. This feature allows users to use the numeric keypad on the ADB keyboard to control mouse functions. Figure 6-16 shows the key functions, and Table 6-16 lists the mouse functions and the keys that implement these functions.

■ **Figure 6-16** ADB mouse keypad



■ **Table 6-16** ADB mouse functions

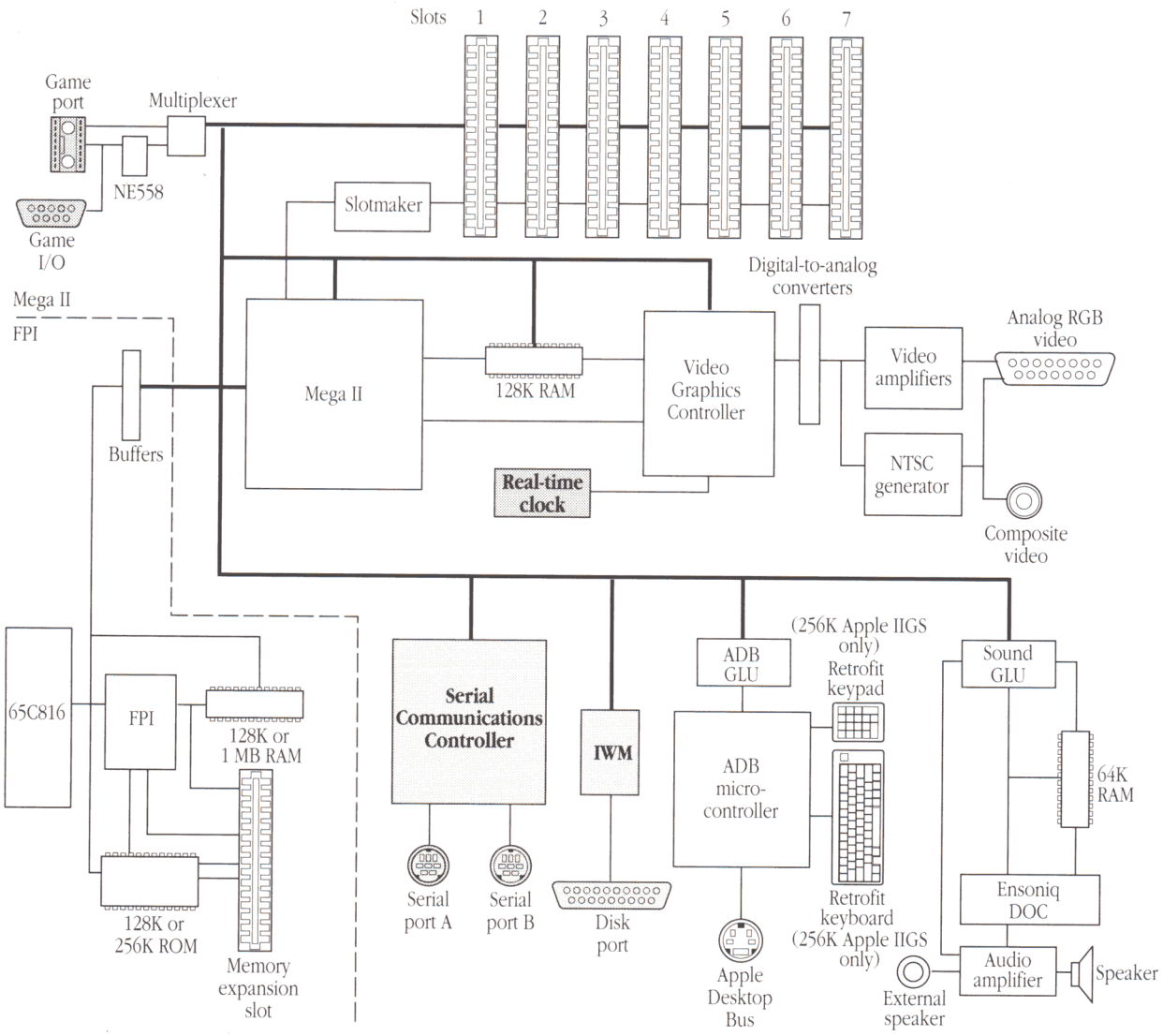
Function	Action
Enable ADB mouse.	Press Shift-Command-Clear key sequence.
Click mouse button.	Press keypad 5.
Lock down mouse button.	Press keypad 0.
Release mouse button.	Press keypad decimal.
Click second mouse button.	Press keypad "–".
Lock down second button.	Press keypad "=".
Release second button.	Press keypad "/".
Set cursor increment.	Press keypad "*" followed by 0–9.
Set cursor default.	Press keypad "*" twice.
Disable ADB mouse.	Press the Clear key.

Chapter 7 **Built-in I/O Ports and Clock**

The Apple IIGS has several means for data input and output. The primary output device is the video output, covered in Chapter 4. Keyboards and mouse devices provide input. Another means of I/O available is the I/O expansion slots, covered in Chapter 8.

The disk port, the two **serial** ports, and the game port provide additional I/O. Another I/O device, although internal to the Apple IIGS, is the real-time clock (RTC). This chapter describes the disk-port connector, serial ports, the game port, and the real-time clock in the Apple IIGS. Figure 7-1 shows the Apple IIGS block diagram and position of these I/O devices within the system.

■ **Figure 7-1** I/O components of the Apple IIGS described in this chapter



The disk port

The Apple IIGS uses a disk-port connector, located on the back of the computer, which is compatible with all 3.5-inch Apple II disk drives and most 5.25-inch Apple II disk drives. The firmware routines within the ROM make communicating with the disk drives reliable and consistent.

▲ **Warning** Using means other than documented entry points and Apple IIGS ROM firmware routines to communicate with the disk drives is extremely dangerous. Not only do you run the risk of crashing the operating system, but the potential for damaging data on your system disk is high. It is recommended that you use firmware calls when accessing all disk devices connected to your Apple IIGS. ▲

Apple II compatibility

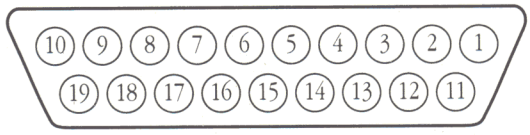
The Apple IIGS uses the same disk drive interface as the Apple IIc and IIe. Programs written for both of these earlier computers will run on the Apple IIGS. The firmware recognizes ProDOS **block device** calls and **SmartPort** interface calls to both the Apple UniDisk™ 3.5-inch and Apple DuoDisk® 5.25-inch disk drives.

To find out how to use the ProDOS block device calls, see the *ProDOS 8 Technical Reference Manual*. To find out how to use the SmartPort interface calls, see the *Apple IIGS Firmware Reference*.

The disk-port connector

The disk-port connector is located at the rear of the Apple IIGS case. It is a 19-pin connector. Figure 7-2 shows the connector. Table 7-1 gives a description of each pin.

■ **Figure 7-2** Disk-port connector



■ **Table 7-1** Pins on the disk-port connector

Pin	Signal	Description
1,2,3	GND	Ground reference and supply
4	3.5DISK	3.5- or 5.25-inch drive select
5	–12V	–12-volt supply
6	+5V	+5-volt supply
7,8	+12V	+12-volt supply
9	DR2	Drive 2 select
10	WRPROTECT	Write-protect input
11	**Phase** 0	Motor phase 0 output
12	Phase 1	Motor phase 1 output
13	Phase 2	Motor phase 2 output
14	Phase 3	Motor phase 3 output
15	WREQ	Write request
16	HDSEL	Head select
17	DR1	Drive 1 select
18	RDDATA	Read data input
19	WDATA	Write data output

▲ **Warning** The power connections on this disk port are for use by the disk drive only. Do not use them for any other purpose. Any other use of these connections may damage the computer's voltage regulator. ▲

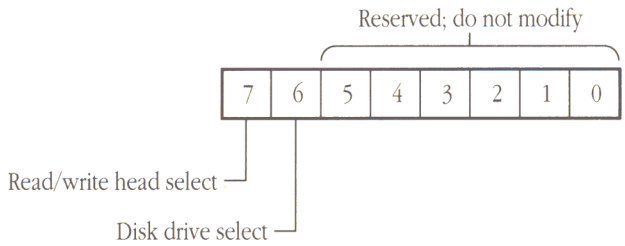
The Disk Interface register

The Disk Interface register ($C031) serves as a control register for the disk drive. By writing to this register, you select the type of disk drive being used and the side of the disk to be accessed.

This register uses only two bits, which are both cleared on reset. When the Disk Interface register is read, 0's are returned in the unused positions (bits 5 through 0). Figure 7-3 shows the format for this register. Descriptions of each bit are listed in Table 7-2.

▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 7-3** Disk Interface register at $C031



■ **Table 7-2** Bits in the Disk Interface register

Bit	Value	Description
7	1	Read/write head select bit: A 1 in this position selects head 1.
	0	A 0 selects head 0.
6	1	Disk drive select bit: A 1 in this position selects 3.5-inch disks.
	0	A 0 selects 5.25-inch disks.
5–0	–	Reserved; do not modify.

The IWM

The disk-port interface is enhanced by the Integrated Woz Machine **(IWM),** which simplifies the microprocessor's task of reading and writing serial group-code recording (GCR) encoded data to and from the disk drives. To perform disk operations, the microprocessor simply reads or writes control and data bytes to or from the IWM.

▲ **Warning** Writing directly to the IWM is extremely dangerous. Not only do you run the risk of crashing the operating system, but the potential for damaging data on your system disk is high. It is recommended that you use firmware calls when accessing all disk devices connected to your Apple IIGS. ▲

The IWM contains several typical disk support circuits, which make writing data to the disk possible. These are the discriminator, the phase-locked loop, the data separator, and the write current circuitry.

The IWM contains several registers that allow you to control disk access:

- the Mode register
- the Status register
- the Handshake register
- the Data register

The IWM is mapped as an internal device with soft switches at addresses $C0E0 through $C0EF. These are the same addresses as in the Apple IIc. Table 7-3 shows these locations and their functions.

Table 7-3 Disk-port soft switches

Address	Description
$C0E0	Stepper motor phase 0 low
$C0E1	Stepper motor phase 0 high
$C0E2	Stepper motor phase 1 low
$C0E3	Stepper motor phase 1 high
$C0E4	Stepper motor phase 2 low
$C0E5	Stepper motor phase 2 high
$C0E6	Stepper motor phase 3 low
$C0E7	Stepper motor phase 3 high
$C0E8	Drive disabled
$C0E9	Drive enabled
$C0EA	Drive 0 select
$C0EB	Drive 1 select
$C0EC	Q6 select bit low
$C0ED	Q6 select bit high
$C0EE	Q7 select bit low
$C0EF	Q7 select bit high

Soft switches Q6 and Q7 are select bits for accessing registers within the IWM. By setting or clearing the Q6, Q7, and spindle motor switches, you may read or write to one of the internal IWM registers, as listed in Table 7-4.

■ **Table 7-4** IWM states

Q7	Q6	Spindle motor	Operation
0	0	1	Read Data register
0	1	x	Read Status register
1	0	x	Read Handshake register
1	1	0	Write Mode register
1	1	1	Write Data register

The drive-enable soft switches and the drive-select switches control the state of the disk-select signals DR1 and DR2 located at the disk-port connector. Table 7-5 shows how these soft switches determine the state of the disk-select signals.

■ **Table 7-5** Controlling the disk select signals

Soft switches				Disk port signals	
$C0E8	$C0E9	$C0EA	$C0EB	DR1	DR2
1	–	–	–	0	0
–	1	1	–	1	0
–	1	–	1	0	1

The Mode register

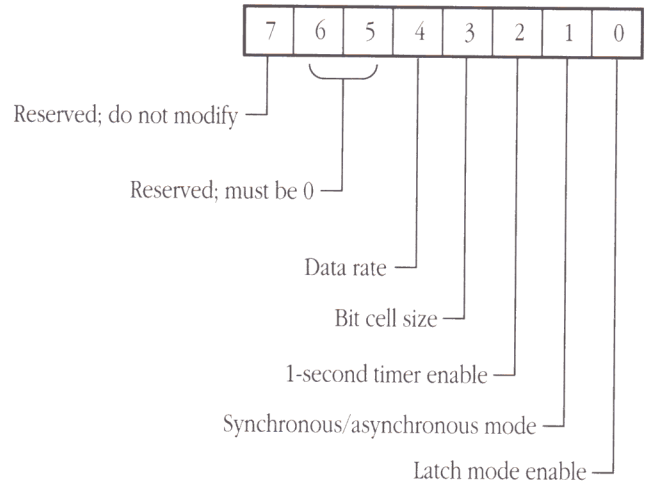
The Mode register is a write-only register and contains bits that control the state of the IWM. These bits are shown in Figure 7-4. Table 7-6 gives a description of these bits. To write to the Mode register, set the appropriate soft switches required to access the Mode register. (See Table 7-4.) Writing to any odd IWM address ($C0E0 through $C0EF) will write to this register.

◆ *Note:* Writing to the Mode register will succeed only after the one-second timer has timed out.

■ **Table 7-6** Bits in the Mode register

Bit	Value	Description
7	–	Reserved; do not modify.
6–5	–	Reserved; always write 0.
4	1	8-MHz read-clock speed selected.
	0	7-MHz read-clock speed selected. Set this bit to 0 for all Apple IIGS disk accesses.
3	1	Bit-cells are 2 microseconds; used in accesses to Apple 3.5-inch drives.
	0	Bit-cells are 4 microseconds; used in accesses to SmartPort devices and all Apple 5.25-inch disk drives.
2	1	One-second timer is disabled.
	0	One-second timer is enabled. When the current disk drive is deselected, the drive will remain enabled for 1 second if this bit is set.
1	1	Asynchronous handshake protocol selected; for all except Apple 5.25-inch Apple disk drives.
	0	Synchronous handshake protocol selected; for Apple 5.25-inch disk drives.
0	1	Latch mode is enabled; read-data byte remains valid for full byte time (16 microseconds if using 2-microsecond bit-cells; 32 microseconds if using 4-microsecond bit-cells).
	0	Latch mode is disabled; read-data byte remains valid for approximately 7 microseconds.

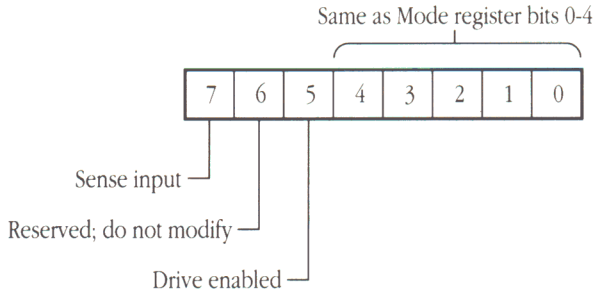
■ **Figure 7-4** Mode register



The Status register

The Status register is a read-only register and contains bits that reflect the current state of the disk interface. These bits are shown in Figure 7-5. Table 7-7 gives a description of each bit. To read from the Status register, set the appropriate soft switches required to access the Status register. (See Table 7-4). Reading from any even IWM address ($C0E0 through $C0EF) will read from this register.

■ **Figure 7-5** Status register



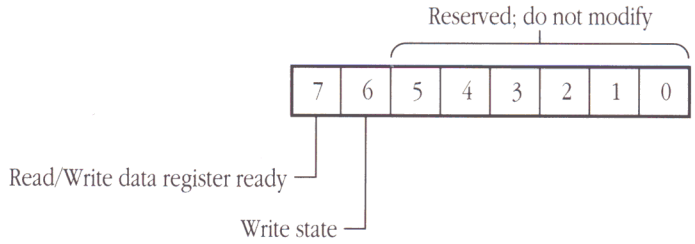
■ **Table 7-7** Bits in the Status register

Bit	Value	Description
7	–	Sense input line from disk device. Multifunction input; use determined by disk device. (Used as a write-protect sense in some Apple disk drives.)
6	–	Reserved; do not modify.
5	1	Either drive 1 or drive 2 is selected and the drive motor is on.
	0	No drive is currently selected.
4–0	1	Same as Mode register bits 4–0. (See Figure 7-4 and Table 7-6).

The Handshake register

The Handshake register is a read-only register that contains the status of the IWM when writing out the data to the disk drive. The format of this register is shown in Figure 7-6. Table 7-8 gives a description of the bits. To read from the Handshake register, set the appropriate soft switches required to access the Handshake register. (See Table 7-4.) Reading from any even IWM address ($C0E0 through $C0EF) will read from this register.

■ **Figure 7-6** Handshake register



■ **Table 7-8** Bits in the Handshake register

Bit	Value	Description
7	1	Read/write data register is ready for data.
	0	Read/write data register is full.
6	1	No write underrun has occurred; the last write to the disk drive was successful.
	0	A write underrun has occurred; a recent data byte was missed and not written to the disk.
5–0	1	Reserved; do not modify.

The data register

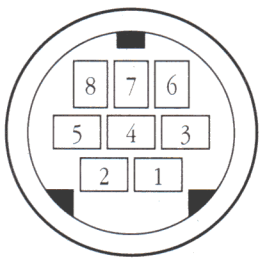
The Data register is a dual-function register. Depending on the state of soft switches Q6 and Q7 (Table 7-3), this register functions as a Read-Data register and a Write-Data register. See Table 7-4 for the state of these bits when reading from and writing to this register. To read from the Data register, set the appropriate soft switches required to read the Data register. (See Table 7-4.) Reading from any even IWM address ($C0E0 through $C0EF) will read from this register. To write to the Data register, set the appropriate soft switches required to write to the Data register. (See Table 7-4.) Writing to any odd IWM address ($C0E0 through $C0EF) will write to this register.

The serial ports

The Apple IIGS has two RS-232-C serial ports located at the back of the computer, which provide synchronous and asynchronous serial communications. Each of these ports may be used to drive a modem, printer, plotter, or other serial device, or as an **AppleTalk local area network** port. These serial ports are called channel A and channel B, and are virtually identical except for the different addresses assigned to each. Only the firmware differs in the way the routines utilize the hardware to provide RS-232 or AppleTalk protocol. Figure 7-7 shows the pin organization of the serial-port connectors. Table 7-9 gives a description of the signals.

◆ *Note:* Remember that firmware for serial ports A and B is located in the ROM space for slots 1 and 2. Because the AppleTalk firmware operates through either port A or port B, one of the slots (1 or 2) must be available to the AppleTalk firmware. See the *Apple IIGS Owner's Guide* for details on choosing serial-port functions from the Control Panel.

■ **Figure 7-7** Pin configuration of a serial-port connector



■ **Table 7-9** Pins on a serial-port connector

Pin	Signal	Description
1	DTR	Data terminal ready
2	HSKI	Handshake in
3	TX Data –	Transmit data –
4	GND	Ground reference and supply
5	RX Data –	Receive data –
6	TX Data +	Transmit data +
7	GPI	General purpose input
8	RX Data +	Receive data +

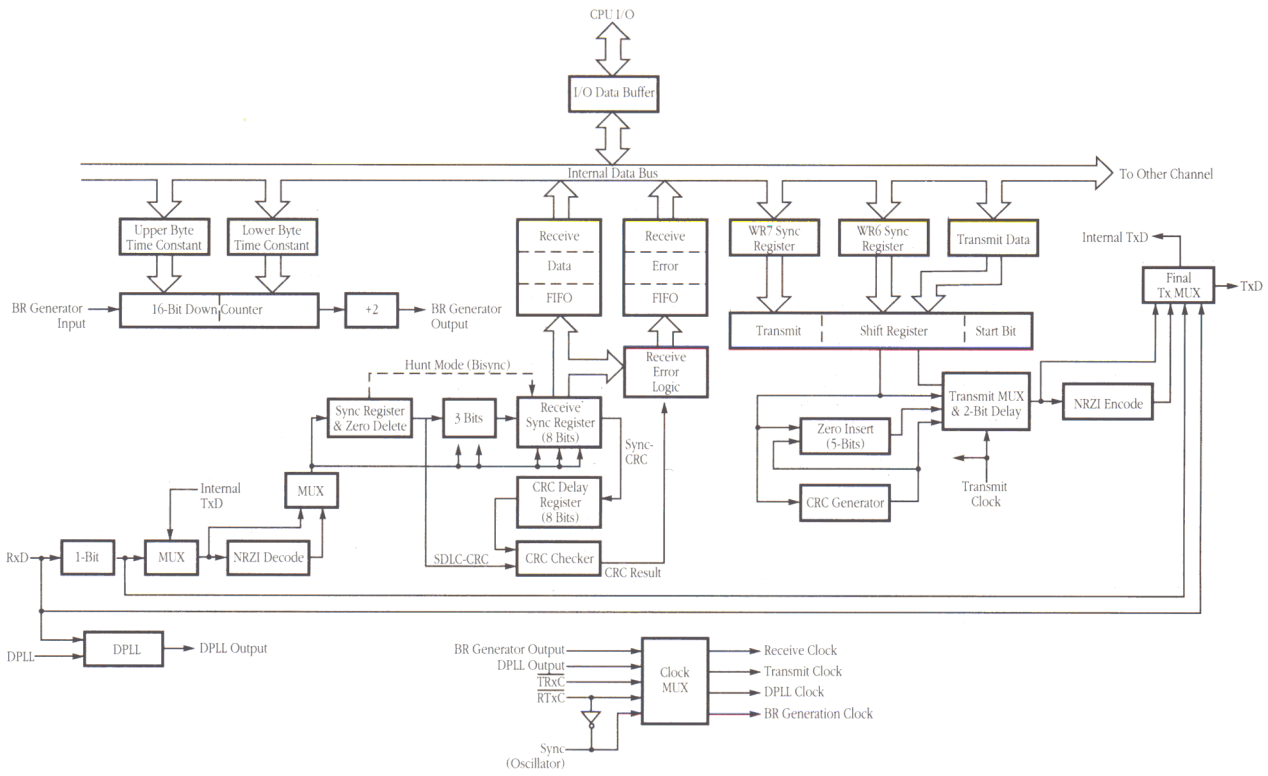
Noncompatibility with ACIA

Previous Apple II computers use an asynchronous communications interface adapter **(ACIA)** chip, either built into the computer (as in the Apple IIc), or on a peripheral card (as used in the Apple IIe), to control the serial ports in the computer. Due to the great difference in internal architecture of the ACIA and the **Serial Communications Controller (SCC) chip,** previous Apple II programs that do not use the serial-port firmware calls but rather communicate directly to the ACIA will be incompatible with the Apple IIGS serial ports. Existing Apple II programs not using the serial-port firmware calls must be rewritten, using firmware routines or SCC commands.

The Serial Communications Controller

The Apple IIGS uses a Zilog 8530 Serial Communications Controller (SCC) chip to control the two serial ports. The SCC is a programmable, dual-channel, multiprotocol data communications chip as well as a parallel-to-serial/serial-to-parallel converter and controller. The SCC has on-chip baud-rate generators and phase-locked loops, which reduce the need for additional support circuitry. Figure 7-8 is a block diagram showing major functional segments of the Zilog SCC.

■ **Figure 7-8** Zilog Serial Communications Controller chip (Reproduced by permission. © 1986 Zilog, Inc. This material may not be reproduced without the consent of Zilog, Inc.)



To communicate with the SCC, you must address one SCC Command register and one SCC Data register for each of the two serial ports. These register addresses are listed in Table 7-10.

■ **Table 7-10** SCC Command and SCC Data register addresses

Register	Channel A	Channel B
SCC Command	$C039	$C038
SCC data	$C03B	$C03A

Through these two registers, you can access the 9 SCC read registers and the 15 SCC write registers for each channel. These registers and their functions are listed in Table 7-11 and Table 7-12. Figure 7-9 is a diagram showing the major data paths within the Zilog SCC.

■ **Table 7-11** SCC read register functions

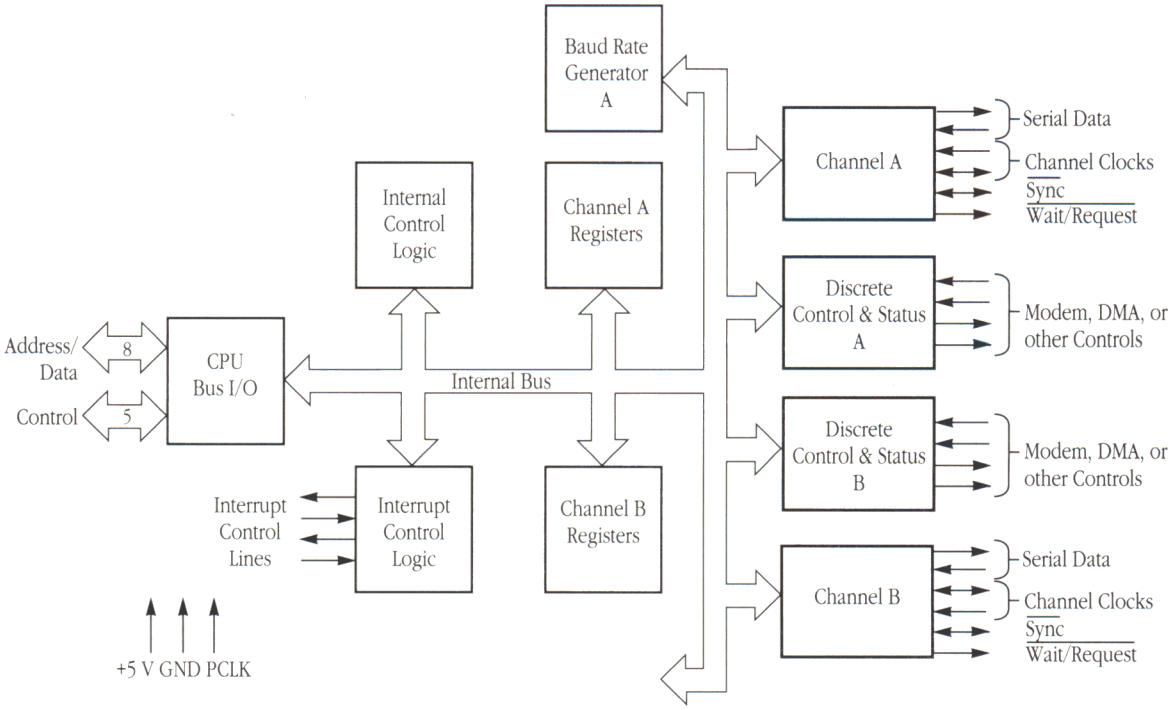
Read register	Functions
0	Transmit/receive buffer status External status
1	Receive status Residue codes Error conditions
2	Interrupt vectors
3	Interrupt pending bits (channel A)
8	Receive buffer
10	Transmit and receive status
12	Baud-rate generator time constant, low byte
13	Baud-rate generator time constant, high byte
15	External status Interrupt control

◆ *Note:* If you wish to use the SCC without utilizing the firmware routines, you must initialize and communicate with the SCC in proper sequence. Details of how to program the SCC may be found in the *Z8530 SCC Serial Communications Controller Technical Manual* (September, 1986), from Zilog Corporation.

■ **Table 7-12** SCC write register functions

Write register	Functions
0	Register pointers CRC initialization Mode resets
1	Interrupt conditions Wait/DMA request control
2	Interrupt vector
3	Receive byte format Receive CRC enable
4	Transmit/receive clock rate, sync byte format
5	Transmit byte format Transmit CRC enable
6	Sync/SDLC byte format
8	Transmit buffer
9	Master interrupt bits Reset bits Interrupt daisy chain
10	Transmit/receive control Data encoding format
11	Receive and transmit clock control
12	Baud-rate generator time constant, low byte
13	Baud-rate generator time constant, high byte
14	Baud-rate generator control Phase-locked loop control Echo and loopback
15	External interrupt control status

■ **Figure 7-9** Data paths in the Zilog SCC (Reproduced by permission. © 1986 Zilog, Inc. This material may not be reproduced without the consent of Zilog, Inc.)



The game I/O port

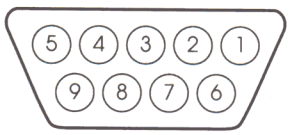
All Apple II computers have a game I/O port to which joysticks or hand-controls can connect. These controls allow users to provide mechanical input to a game program, which analyzes these inputs and responds accordingly.

Four digital switch inputs (SW0 through SW3) are provided, as well as four analog hand control inputs (PDL0 through PDL3) and four digital annunciator outputs (AN0 through AN3). The following sections describe these inputs and outputs in detail.

Game I/O

The Mega II supports hand-control inputs PDL0 through PDL3 and switch inputs SW0 through SW3. These inputs are available through the 16-pin DIP game connector (J21) located below slot 4, and through the 9-pin connector (J9) that is located at the rear panel. Annunciator outputs AN0 through AN3 are provided by the Slotmaker IC and are available only through the 16-pin DIP connector. Unlike previous Apple II computers, the STROBE output is not available on the game I/O port. Figure 7-10 shows the two Apple IIGS game connectors. Table 7-13 lists the locations of the game I/O signals.

■ **Figure 7-10** Game I/O connectors



■ **Table 7-13** Game I/O signals

Pin number			
J21	J9	Signal	Description
1	2	+5V	+5 volts
2	7	SW0	Switch input 0
3	1	SW1	Switch input 1
4	6	SW2	Switch input 2
5	–	+5V	+5-volt pullup
6	5	PDL0	Analog input 0
7	4	PDL2	Analog input 2
8	3	GND	Power and signal ground
9	–	SW3	Switch input 3
10	8	PDL1	Analog input 1
11	9	PDL3	Analog input 3
12	–	AN3	Digital output 3
13	–	AN2	Digital output 2
14	–	AN1	Digital output 1
15	–	AN0	Digital output 0
16	–	N.C.	No connection

The hand-control signals

Several inputs and outputs are available at the 16-pin IC connector on the main logic board: four 1-bit inputs, or switches (SW0 through SW3); four analog inputs (PDL0 through PDL3); and four 1-bit outputs (AN0 through AN3). You can access all these inputs and outputs from your application program. Note that the SW3 input is new to the Apple IIGS.

Ordinarily, you connect a pair of hand controls to the 16-pin connector. The rotary controls use two analog inputs, and the push buttons use two 1-bit inputs. But you can also use these inputs and outputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick. Figure 7-10 shows the connector pin numbers.

The Apple Desktop Bus will accept ADB hand controls, joysticks, and graphics tablets as well as those keyboards and mouse devices specifically designed for the ADB. The ADB microcontroller handles mouse and keyboard input devices transparently; that is, simply reading the standard locations will return the current values of these devices. See Chapter 6 for more information.

Annunciator outputs: The four 1-bit outputs (AN0 through AN3) are called *annunciators.* Each annunciator can be used to turn a lamp, a relay, or some similar electronic device on and off.

▲ **Warning** When driving a device with the annunciator outputs, be sure not to load any one output with more than one standard TTL load. ▲

Each annunciator is controlled by a soft switch, and each switch uses a pair of memory locations. These memory locations are shown in Table 7-14. Any reference to the lower address of an address pair turns the corresponding annunciator off; a reference to the higher address turns the annunciator on. You can determine the state of only one annunciator, AN3. To do this, read the RDDHIRES switch at location $C064 and test bit 5. If this bit is a 0, then AN3 is cleared. If this bit is a 1, then AN3 is set.

Annunciator 3 serves a dual purpose in the Apple IIGS: It also serves as a switch, allowing you to toggle between two display modes. Refer to Chapter 4 for more information about the role of annunciator 3 in video. Table 7-14 shows the annunciator memory locations.

Switch inputs: The four 1-bit inputs (SW0 through SW3) can be connected to the output of another electronic device or to a push button. When you read a byte from one of these locations, only the high-order bit—bit 7—is valid information; the rest of the byte is undefined. The soft switch locations that reflect the state of these switch inputs are 49249 through 49251 ($C060 through $C063), as shown in Table 7-15.

Table 7-14 Annunciator memory locations

Annunciator			Address	
Number	Pin*	State	Hex	Dec
0	15	Off	$C058	49240
		On	$C059	49241
1	14	Off	$C05A	49242
		On	$C05B	49243
2	13	Off	$C05C	49244
		On	$C05D	49245
3	12	Off	$C05E	49246
		On	$C05F	49247

* Pin numbers given are for the 16-pin IC connector on the circuit board.

Analog inputs: The four analog inputs (PDL0 through PDL3) are designed for use with 150,000-ohm variable resistors or potentiometers. The variable resistance is connected between the +5-volt supply and each input, so that it makes up part of a timing circuit. The circuit changes state when its time constant has elapsed, and the time constant varies as the resistance varies. Your program can measure this time by counting in a loop until the circuit changes state, or times out.

Before a program can read the analog inputs, it must first reset the timing circuits. Accessing memory location 49264 ($C070) does reset these circuits. As soon as you reset the timing circuits, the high bits of the bytes at locations 49252 through 49255 ($C064 through $C067) are set to 1. Within about 3 milliseconds, these bits will change back to 0 and remain there until you reset the timing circuits again. The exact time each of the four bits remains high is directly proportional to the resistance connected to the corresponding input. If these inputs are open—no resistances are connected—the corresponding bits may remain high indefinitely.

To read the analog inputs, use a program loop that resets the timers and then increments a counter until the bit at the appropriate memory location changes to 0. High-level languages, such as BASIC, also include convenient means of reading the analog inputs: Refer to your language manuals.

Summary of secondary I/O locations

Table 7-15 shows the memory locations for all of the built-in I/O devices except the keyboard and the video display and other primary I/O locations. As explained earlier, some soft switches should be accessed only by means of read operations; those switches are marked.

■ **Table 7-15** Secondary I/O memory locations

Soft switch	Address		Definition
	Hex	Dec	
SPKR	$C030	49200	Toggle speaker (read only).
CLRAN0	$C058	49240	Clear annunciator 0.
SETAN0	$C059	49241	Set annunciator 0.
CLRAN1	$C05A	49242	Clear annunciator 1.
SETAN1	$C05B	49243	Set annunciator 1.
CLRAN2	$C05C	49244	Clear annunciator 2.
SETAN2	$C05D	49245	Set annunciator 2.
CLRAN3	$C05E	49246	Clear annunciator 3.
SETAN3	$C05F	49247	Set annunciator 3.
BUTN3	$C060	49248	Read switch 3 (read only).
BUTN0	$C061	49249	Read switch 0 (read only).
BUTN1	$C062	49250	Read switch 1 (read only).
BUTN2	$C063	49251	Read switch 2 (read only).
PADDL0	$C064	49252	Read analog-input 0.
PADDL1	$C065	49253	Read analog-input 1.
PADDL2	$C066	49254	Read analog-input 2.
PADDL3	$C067	49255	Read analog-input 3.
PTRIG	$C070	49264	Analog-input reset.

Built-in real-time clock

The real-time clock (RTC) chip provides the system with calendar and clock information as well as parameter RAM preserved by battery power. These functions are performed through two read/write registers: the control and data registers.

◆ *Note:* The parameter RAM in the RTC is used for system parameters, and is not available to, nor should it be used by, programs other than the system.

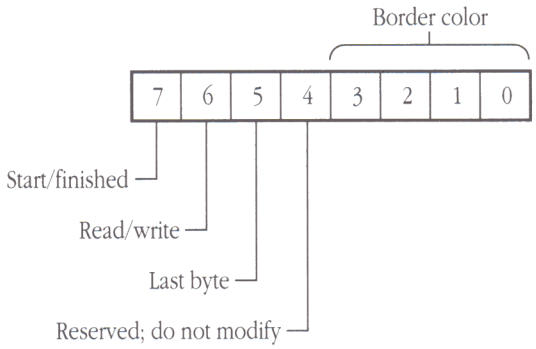
The control register (located at $C034), shown in Figure 7-11, serves a dual function: as the control register for the RTC and as the Border Color register. Refer to "Border Color" in Chapter 4 for more information on controlling the color of the display border.

Serial data communication to and from the RTC is carried out one byte at a time. (The terms *read* and *write* are used in perspective of the system: A read transfers data from the clock chip, while a write transfers data to the clock chip.) To write to the clock chip, the program must first write the data into the Data register ($C033), then set the appropriate bits in the control register ($C034). To read from the clock chip, set the appropriate control register bits, and then read the data from the Data register.

◆ *Note:* To remain compatible with future Apple II products, use the firmware calls to read and write data to the RTC. See the *Apple IIGS Firmware Reference* for how to use the firmware.

▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 7-11** Control register at $C034



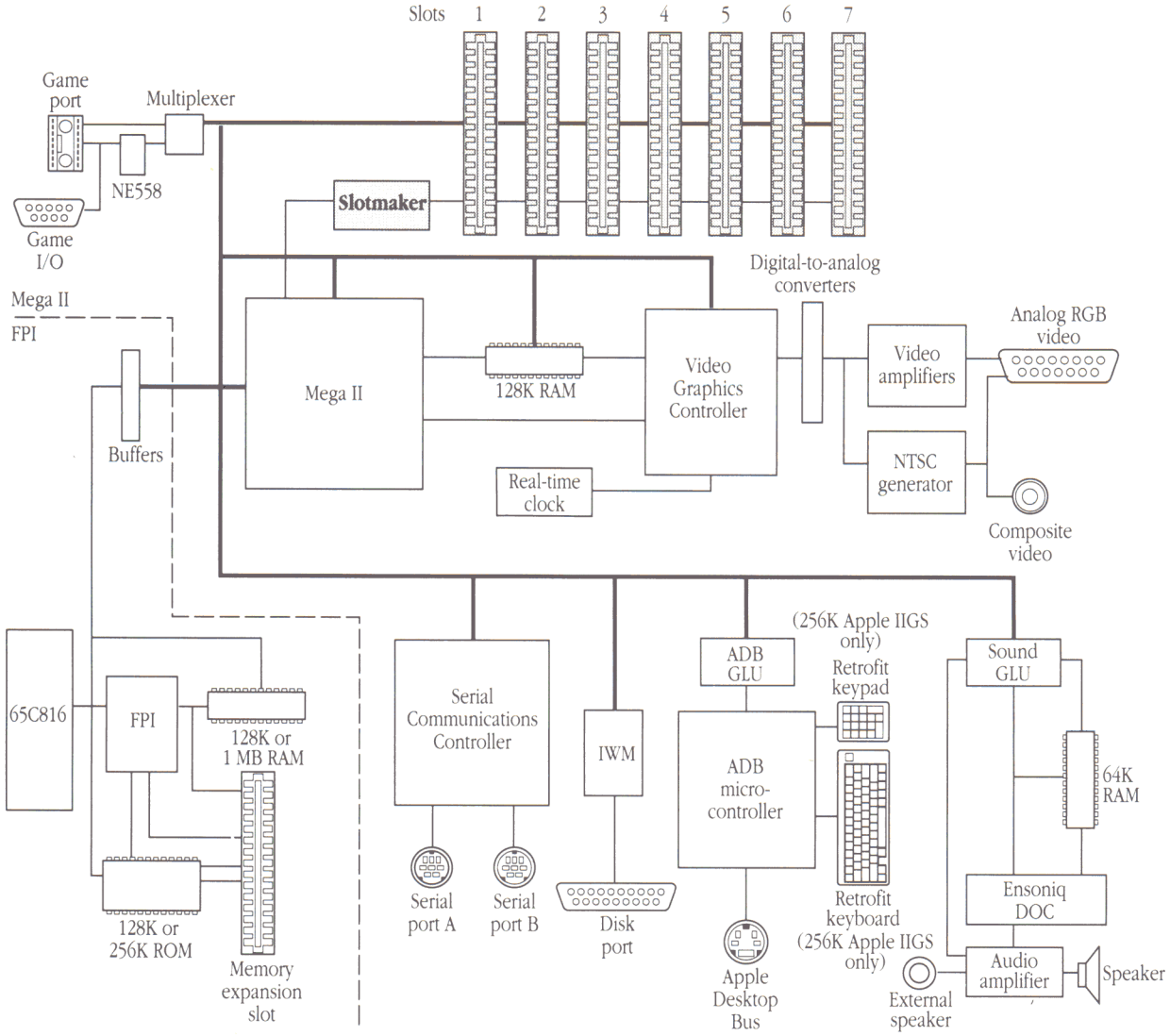
■ **Table 7-16** Bits in the control register

Bit	Value	Description
7	1	A read or write to the the clock chip begins by setting this bit to 1.
	0	This bit is set to 0 automatically by the RTC when the data exchange is complete. The program can detect that the exchange has been completed by polling bit 7 for a 0.
6	1	The read/write bit: Set this bit to 1 prior to a read from the RTC.
	0	Set this bit to 0 prior to a write to the RTC.
5	1	The last-byte control bit: After the last byte has been read or written, this bit must be set to 1. This last step is necessary to avoid corrupting the data in the clock chip after the transactions are completed.
	0	A data transfer typically involves an exchange of two or three bytes. Set this bit to 0 before transferring any bytes to or from the RTC.
4	–	Reserved; do not modify.
3–0	–	Border Color register: See "Border Color" in Chapter 4 for details on selecting the video display border color.

Chapter 8 I/O Expansion Slots

The main logic board of the Apple IIGS has seven empty peripheral-card connectors or slots on it. These slots make it possible to add features by plugging in peripheral cards with additional hardware. This chapter describes the hardware that supports these slots, including the signals available at the expansion slots. Figure 8-1 is a block diagram of the Apple IIGS that shows the relationship of the slots in the computer.

■ **Figure 8-1** Expansion slots and other components in the Apple IIGS

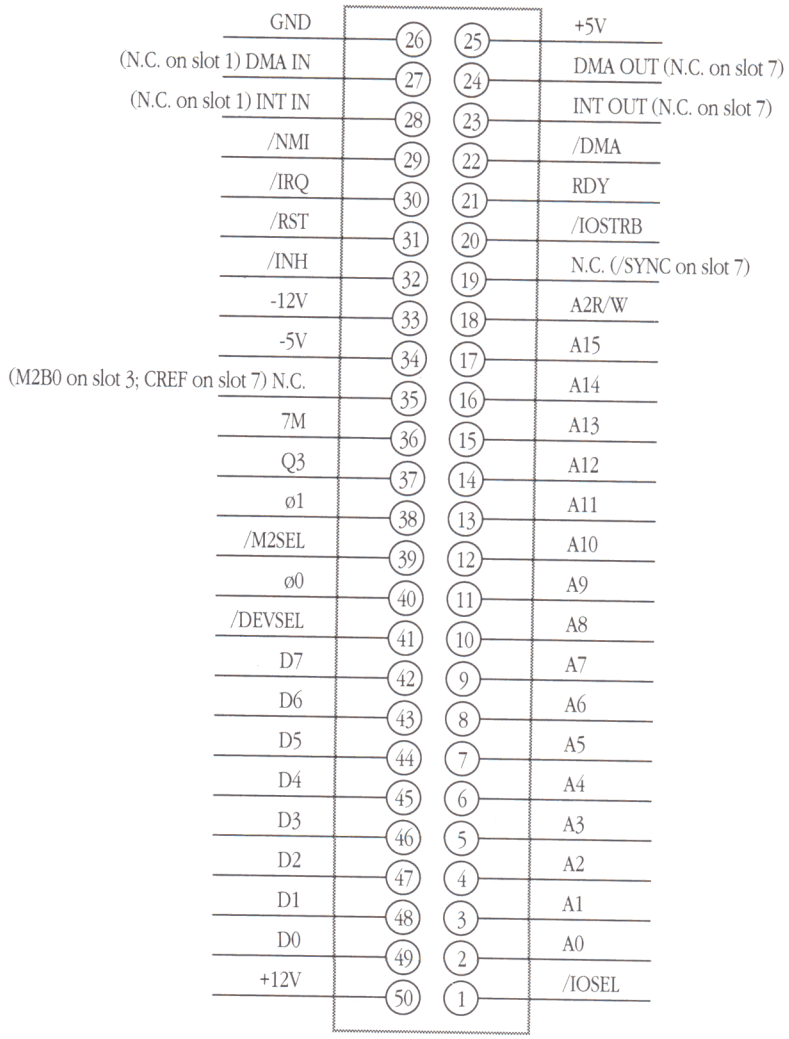


◆ *Note:* The Apple IIGS has seven expansion slots plus a memory expansion slot. This memory expansion slot is not the same as the seven expansion slots, nor should it be used as such. Also, the memory expansion slot is not the same as the auxiliary slot in the Apple IIe, nor should it be used as such. The memory expansion slot is to be used for memory expansion cards designed specifically for this slot. See "Memory Expansion," in Chapter 3, for a description of this slot.

The expansion slots

The seven connectors lined up across the back part of the Apple IIGS main circuit card are the expansion slots (also called *peripheral slots* or simply *slots*), numbered from 1 to 7. They are 50-pin card-edge connectors with pins on 0.10-inch centers. A circuit card plugged into one of these connectors has access to all the signals necessary to perform input and output and to execute programs in RAM or ROM on the card. These signals are described in Table 8-1 and are shown in Figure 8-2.

■ **Figure 8-2** Peripheral-expansion slot pins



■ **Table 8-1** Expansion slot signals

Pin	Signal	Description
1	/IOSEL	Normally high; goes low during ø0 when the 65C816 addresses location $Cnxx, where *n* is the connector number. This line can drive 10 LS TTL loads.*
2–17	A0–A15	Three-state address bus: The address becomes valid during ø1 and remains valid during ø0. Each address line can drive 2 LS TTL loads.*
18	A2R/W	Three-state read/write line: Valid at the same time as the address bus; high during a read cycle, low during a write cycle. It can drive 2 LS TTL loads.*
19	/SYNC	Composite horizontal and vertical sync, on expansion slot 7 only. (This pin has no connection on the other 6 slots.) This line can drive 2 LS TTL loads.*
20	/IOSTRB	Normally high; goes low during ø0 when the 65C816 addresses a location between $C800 and $CFFF. This line can drive 4 LS TTL loads.
21	RDY	Input to the 65C816: Pulling this line low during ø1 halts the 65C816 with the address bus holding the address of the location currently being fetched. This line has a 4700-ohm pullup resistor to +5 volts.
22	/DMA	Input to the address bus buffers: Pulling this line low during ø1 disconnects the 65C816 from the address bus. This line has a 3300-ohm pullup resistor to +5 volts.
23	INT OUT	Interrupt priority daisy-chain output: Usually connected to pin 28 (INT IN). On slot 7 only, this pin has no connection.
24	DMA OUT	DMA priority daisy-chain output: Usually connected to pin 27 (DMA IN). On slot 7 only, this pin has no connection.
25	+5V	+5-volt power supply: A total of 500 mA is available for all peripheral cards.
26	GND	System common ground.
27	DMA IN	DMA priority daisy-chain input: Usually connected to pin 24 (DMA OUT). On slot 1 only, this pin has no connection.
28	INT IN	Interrupt priority daisy-chain input: Usually connected to pin 23 (INT OUT). On slot 1 only, this pin has no connection.
29	/NMI	Nonmaskable interrupt to 65C816: Pulling this line low starts an interrupt cycle with the interrupt-handling routine at location $03FB. This line has a 3300-ohm pullup resistor to +5 volts.

■ **Table 8-1** Expansion slot signals (Continued)

Pin	Signal	Description
30	/IRQ	Interrupt request to 65C816: Pulling this line low starts an interrupt cycle only if the interrupt-disable (I) flag in the 65C816 is not set. Uses the interrupt-handling routine at location $03FE. This line has a 3300-ohm pullup resistor to +5 volts.
31	/RST	Pulling this line low initiates a reset routine.
32	/INH	Pulling this line low during ø1 inhibits (disables) the memory on the main circuit board. This line has a 3300-ohm pullup resistor to +5 volts.
33	–12V	–12-volt power supply: A total of 200 mA is available for all peripheral cards.
34	–5V	–5-volt power supply: A total of 200 mA is available for all peripheral cards.
35	CREF	3.58-MHz color-reference signal.: slot 7 only. This line can drive 2 LS TTL loads.*
35	M2B0	Mega II bank 0 signal. 256K Apple IIGS: slot 3 only; 1 MB Apple IIGS: slots 1—6. This signal is the bank address bit and is valid only during Mega II accesses.
36	7M	System 7-MHz clock: This line can drive 2 LS TTL loads.*
37	Q 3	System 2-MHz asymmetrical clock: This line can drive 2 LS TTL loads.*
38	ø1	ø1 clock: This line can drive 2 LS TTL loads.*
39	/M2SEL	The Mega II select signal: This signal goes low whenever the Mega II is addressing a location within the 128K of Mega II RAM.
40	ø0	ø0 clock: This line can drive 2 LS TTL loads.*
41	/DEVSEL	Normally high; goes low during ø0 when the 65C816 addresses location $C0nx, where *n* is the connector number plus 8. This line can drive 10 LS TTL loads.*
42–49	D7–D0	Three-state buffered bidirectional data bus: Data become valid during ø0 high and remain valid until ø0 goes low. Each data line can drive 1 LS TTL load.*
50	+12V	+12-volt power supply: A total of 250 mA is available for all peripheral cards.

* Loading limits are for each card.

Apple II compatibility

The seven I/O slots in the Apple IIGS are almost identical to the slots in the Apple IIe, the only exceptions being signals /M2SEL and M2B0. /M2SEL replaces μPSYNC on pin 39, and M2B0 is available at pin 35, only at slot 3; CREF is still available at pin 35, at slot 7.

The slots behave like their counterparts in the Apple II with only a few differences, the most important being the behavior of the address bus. Since the Apple IIGS computer can operate at 2.8 MHz and has a 24-bit address, the address bus to the slots is not always valid as it was in the Apple II. The signal /M2SEL indicates when a valid address for banks 224 or 225 ($E0 or $E1) is present on the address bus and so should be used to qualify any address decoding that does not use /IOSEL. Since these memory spaces contain video buffers and I/O addresses, peripheral video cards can make extensive use of these two signals.

Direct memory access

Direct memory access (DMA) supports the address range $00 through $4F. This means that any peripheral card using DMA may have direct address control of all memory (main and expansion memory). Be sure to load the DMA bank register, located at address $C037, with the 8 most-significant bits of the address before performing DMA.

During DMA cycles (memory access cycles that are controlled by a DMA peripheral card), the address bus is turned off until the bank address has been latched. When that happens, the address bus is enabled, pointing "in" toward the FPI and 65C816. The FPI decodes the address and stored DMA bank address to determine whether the cycle is to RAM, ROM, or Mega II. If the cycle is a DMA to the Mega II (or slots), the Mega II select line is asserted by the FPI, and the FPI data buffers are turned off if the R/W line is high. If the access is to the high-speed RAM, the data buffers are enabled while ø0 is high.

- *Note:* To increase read/write data timing margins to the high-speed RAMs, the FPI generates an early CAS (card address strobe) signal for read cycles and a late CAS signal for write cycles. This makes read data available earlier and requires less write data setup time.

I/O in the Apple IIGS

The input and output functions are made possible by built-in I/O devices and the use of peripheral-slot I/O and DMA cards. The following sections describe these cards.

Slot I/O cards

Most I/O cards used in the Apple II also work in the Apple IIGS. Cards that use the /IOSEL and /DEVSEL bus signals will work especially well, because they do not have to deal with the larger address range of the Apple IIGS.

The 65C816 processor operates with a 24-bit address; however, the I/O slots receive only a 16-bit address. Therefore, cards that use the 16-bit address decode select method rather than the /DEVSEL and /IOSEL signals will not work properly. These cards include the multifunction I/O cards that emulate multiple I/O cards and most add-on RAM cards. In general, these types of cards will not be needed because of the extensive built-in I/O and high-speed RAM expansion already provided.

Cards that use /INH will work properly if

- the system is running at 1.024 MHz
- they assert /INH within 200 nanoseconds of the ø0 falling edge

However, compatibility with this type of card must be determined on an individual basis, because many Monitor firmware calls execute code in bank $FF, and many cards are not designed to decode bank information.

The FPI will ignore any occurrence of /INH when the system is running fast (2.8 MHz), or when it is not in a bank where I/O and language-card operation are enabled. By ignoring /INH, compatibility with existing cards is improved.

DMA cards

Many DMA cards that work successfully in previous Apple II models will work in the Apple IIGS, but may require changes in their firmware or associated software to function properly with the DMA bank register. In general, DMA cards that assert and remove the /DMA signal within the first 120 nanoseconds of the ø0 rising edge will probably work properly; this allows sufficient time for /M2SEL to be activated by the FPI when video and I/O accesses are required.

- *Note:* Normally the system should be running at 1.024 MHz when performing DMA; otherwise, DMA to I/O or Mega II video areas will not work properly. However, DMA can be performed while the system is running fast as long as the following warnings are heeded:
  - Only high-speed RAM or ROM can be accessed (access to I/O, video, or the Mega II banks does not work properly).
  - Fast DMA may cause a repeated cycle to occur to the location currently being accessed by the processor. This repetition could cause a malfunction if the processor is accessing I/O when the DMA occurs; however, a repeated access to a RAM or ROM location will have no effect. The 65C816 can be stopped indefinitely for DMA and does not require any processor refresh cycles from a DMA card.

Expansion-slot signals

Many of the expansion-slot signals can be grouped into three general categories:

- those that constitute and support the address bus
- those that constitute and support the data bus
- those that support the functions of DMA and interrupts

These signals are described in the following paragraphs. For additional information, refer to the **schematic diagrams** in the addendum at the back of the book.

The buffered address bus

The microprocessor's address bus is buffered by two 74HCT245 octal three-state bidirectional buffers. The 65C816 R/W line is also buffered. The FPI disables these buffers when requested by any peripheral card so that peripheral DMA circuitry can control the address bus. The DMA address and A2R/W signals supplied by a peripheral card must be stable all during ø0 of the instruction cycle. (Refer to the timing diagram in Figure 8-11 shown later in this chapter.)

Another signal that can be used to disable normal operation of the Apple IIGS is /INH. Pulling /INH low disables all the memory in the Apple IIGS except the part in the I/O space from $C000 to $CFFF. A peripheral card that uses either /INH or /DMA must observe proper timing; in order to disable RAM and ROM properly, the disabling signal must be stable all during ø0 of the instruction cycle. (Refer to the timing diagram in Figure 8-10 shown later in this chapter).

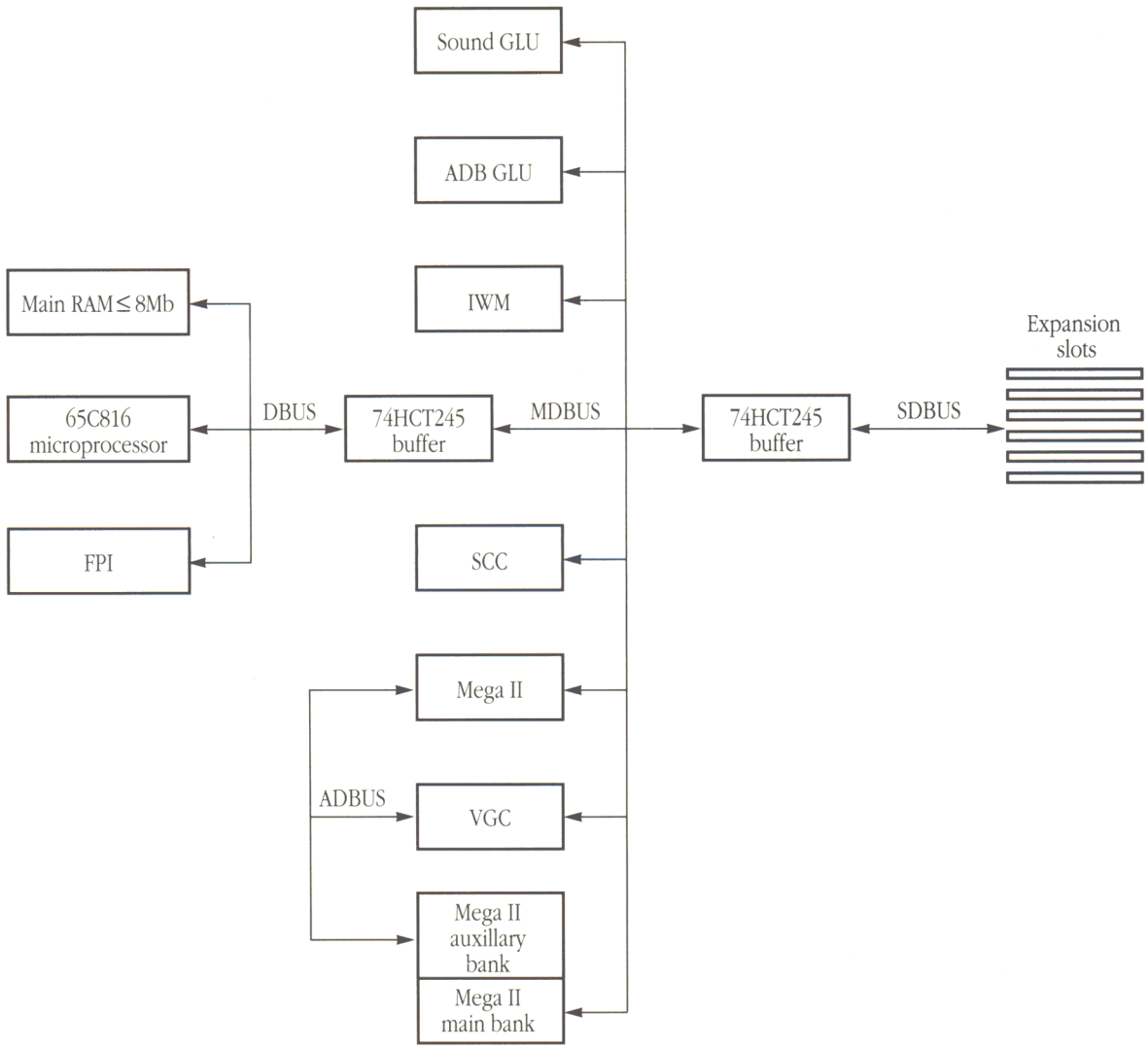
The peripheral devices should use /IOSEL and /DEVSEL as enables. Most peripheral-card ICs require their enable signals to be present for a certain length of time before data are strobed into or out of the device. Remember that /IOSEL and /DEVSEL are asserted only during ø0 high.

The slot data bus

The Apple IIGS has three versions of the microprocessor data bus (shown in Figure 8-3):

- the internal data bus, DBUS, connected directly to the microprocessor and the FPI chip and all main RAM
- the Mega II data bus, MDBUS, connecting the Mega II, VGC, Serial Communications Controller (SCC), Integrated Woz Machine (IWM), ADB and Sound General Logic Units (GLUs), and the Mega II RAM main bank
- the slot data bus, SDBUS, common to all expansion slots

- **Figure 8-3** Data buses within the Apple IIGS



The 65C816 is fabricated with **MOS** (Metal Oxide **Semiconductor**) circuitry, so it can drive capacitive loads of up to about 130 picoFarads. If peripheral cards are installed in all seven slots, the loading on the data bus can be as high as 500 pF, so the 74HCT245 buffer is used to drive the data bus peripheral-card loads. The same situation occurs if you use MOS devices on peripheral cards: They can't provide enough drive current for the fully loaded bus, so you should add buffers. A peripheral card must have the capacity to drive two LS TTL loads per slot pin, plus additional capacitance for the Apple IIGS data bus.

Interrupt and DMA daisy chains

The interrupt requests (/IRQ and /NMI) and the direct memory access (/DMA) signal are available at all seven expansion slots. A peripheral card requests an interrupt or a DMA transfer by pulling the appropriate output line (pin 24) low. If two peripheral cards request an interrupt or a DMA transfer at the same time, they will contend for the data and address buses. To prevent this contention, two pairs of pins on each connector are wired as a priority daisy chain. The daisy-chain pins for interrupts are INT IN (pin 28) and INT OUT (pin 23), and the pins for DMA are DMA IN (pin 27) and DMA OUT (pin 24), as shown in Figure 8-2.

Each daisy chain works like this: The output from each connector goes to the input of the next higher numbered one. For these signals to be useful for cards in lower numbered connectors, all the higher numbered connectors must have cards in them, and all those cards must connect DMA IN to DMA OUT and INT IN to INT OUT. Whenever a peripheral card uses pin /DMA, it must do so only if its DMA IN line is active, and it must disable its DMA OUT line while it is using /DMA. The INT IN and INT OUT lines must be used the same way: Enable the card's interrupt circuits with INT IN, and disable INT OUT whenever /IRQ or /NMI is being used.

Loading and driving rules

Do not overload any pin on the expansion slots; the driving capability of each pin is listed under each signal description in Table 8-1. The address bus, the data bus, and the A2R/W line should be driven by three-state buffers; remember that there is considerable distributed capacitance on these buses and that you should plan on tolerating the added load of up to six additional peripheral cards. MOS devices such as PIAs (peripheral interface adapters) and ACIAs (asynchronous communications interface adapters) cannot switch such heavy capacitive loads; connecting such devices directly to the bus will lead to possible timing and level errors. Buffer all MOS output signals.

The total power-supply current available for all seven expansion slots is

- 500 mA at +5 volts
- 250 mA at +12 volts
- 200 mA at –5 volts
- 200 mA at –12 volts

The support circuitry for the slots is designed to handle a DC load of two LS TTL loads per slot pin and an AC load of no more than 15 pF per slot pin.

Peripheral programming

The seven expansion slots on the main logic board are used for installing circuit cards containing the hardware and firmware needed to interface peripheral devices to the Apple IIGS. These slots are not simple I/O ports; peripheral cards can access the computer's address and control lines via these slots. The expansion slots are numbered from 1 to 7, and certain signals, described below, are used to select a specific slot.

Selecting a device

The Apple IIGS supports several built-in devices and traditional slot devices, with each device taking up one logical slot. Each built-in device is assigned to a slot, and peripheral cards are plugged into any of the seven peripheral slots. However, only one device (either the built-in device or the peripheral device) can be selected at a time for each slot. For example, you can choose either the peripheral device in slot 2 or the internal device that is associated with slot 2 but built into the main logic board, the serial port.

The Slot register

The Slot register, located at $C02D, is used to select which device is enabled for each of the seven slots. That device can be either the internal or a peripheral-card device. If the enable bit for a slot is 1, accesses for that slot's ROM space ($Cnxx) are directed to the ROM on the peripheral card. If the enable bit is cleared, the built-in I/O device is selected, and the

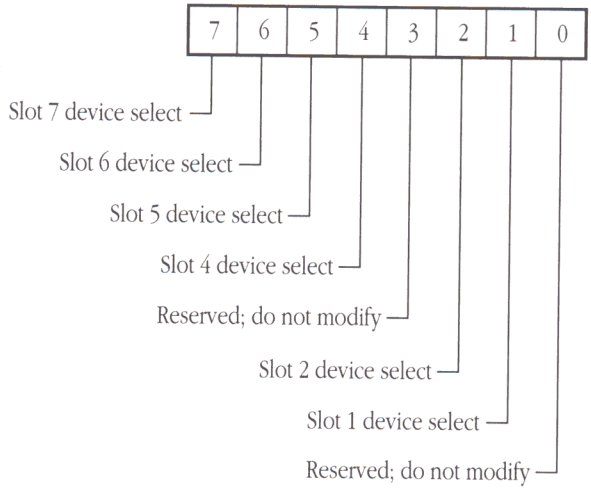
system ROM code associated with the slot is executed. The user can select the appropriate slot device through the Control Panel. The user can access the Control Panel by pressing the Command-Control-Esc keys simultaneously. The Slot register format is given in Figure 8-4. Table 8-2 gives a description of each bit.

◆ *Note:* Slot 3 device hardware addresses are always available. However, the slot 3 ROM space is controlled by the SETSLOTC3ROM and SETINTC3ROM soft switches to maintain compatibility with existing Apple II products.

▲ **Warning** You are encouraged not to manipulate the Slot register bits under software control; you run a great risk of crashing the operating system. ▲

▲ **Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in the preface. ▲

■ **Figure 8-4** Slot register at $C02D



■ **Table 8-2** Bits in the Slot register

Bit	Value	Description
7	0	Selects the internal-device (AppleTalk) ROM code for slot 7.
	1	Enables both the slot-card ROM space (location $C700 to $C7FF) and I/O space $C0F0 to $C0FF.
6	0	Selects the internal-device (5.25-inch disk drive) ROM code for slot 6.
	1	Enables both the slot-card ROM space (location $C600 to $C6FF) and I/O space $C0E0 to $C0EF.
5	0	Selects the internal-device (3.5-inch disk drive) ROM code for slot 5.
	1	Enables both the slot-card ROM space (location $C500 to $C5FF) and I/O space $C0D0 to $C0DF.
4	0	Selects the internal-device (mouse) ROM code for slot 4.
	1	Enables the slot-card ROM space (location $C400 to $C4FF).
3	–	Reserved; do not modify.
2	0	Selects the internal-device (serial port B, the modem port) ROM code for slot 2.
	1	Enables both the slot-card ROM space (location $C200 to $C2FF) and I/O space $C0A0 to $C0AF.
1	0	Selects the internal-device (serial port A, the printer port) ROM code for slot 1.
	1	Enables both the slot-card ROM space (location $C100 to $C1FF) and I/O space $C090 to $C09F.
0	–	Reserved; do not modify.

Note: I/O space for slots 3 ($C0B0 to $C0BF) and 4 ($C0C0 to $C0CF) is always enabled.

Peripheral-card memory spaces

Because the Apple IIGS microprocessor does all its I/O through memory locations, portions of the memory space have been allocated for the exclusive use of the cards in the expansion slots. In addition to the memory locations used for actual I/O, there are memory spaces available for programmable memory (RAM) in the main memory and for read-only memory (ROM or **PROM**) on the peripheral cards themselves.

The memory spaces allocated for the peripheral cards are described below. These memory spaces are used for small dedicated programs such as I/O drivers. Peripheral cards that contain their own driver routines in firmware are called *intelligent peripherals*. They make it possible for you to add peripheral hardware to your Apple IIGS without having to change your programs, provided that your programs follow normal practice for data input and output.

Peripheral-card I/O space

Each expansion slot has the exclusive use of 16 memory locations for data input and output in the memory space beginning at location $C090. Slot 1 uses locations $C090 through $C09F, slot 2 uses locations $C0A0 through $C0AF, and so on through location $C0FF, as shown in Table 8-3.

These memory locations are used for different I/O functions, depending on the design of each peripheral card. Whenever the Apple IIGS addresses one of the 16 I/O locations allocated to a particular slot, the signal on pin 41 of that slot, called /DEVSEL, switches to the active (low) state. This signal can be used to enable logic on the peripheral card that uses the four low-order address lines (A0 through A3) to determine which of its 16 I/O locations is being accessed.

Table 8-3 Peripheral-card I/O memory locations enabled by /DEVSEL

Slot	Locations	Slot	Locations
1	$C090–$C09F	5	$C0D0–$C0DF
2	$C0A0–$C0AF	6	$C0E0–$C0EF
3	$C0B0–$C0BF	7	$C0F0–$C0FF
4	$C0C0–$C0CF		

Peripheral-card ROM space

One 256-byte page of memory space is allocated to each accessory card. This space is normally used for read-only memory (ROM or PROM) on the card, and contains driver programs that control the operation of the peripheral device connected to the card.

The page of memory allocated to each expansion slot begins at location $Cn00, where *n* is the slot number, as shown in Table 8-3 and Table 8-4. Whenever the Apple IIGS addresses one of the 256 ROM memory locations allocated to a particular slot, the signal on pin 1 of that slot, called /IOSEL, switches to the active (low) state. This signal enables the ROM or PROM devices on the card, and the eight low-order address lines determine which of the 256 memory locations is being accessed.

■ **Table 8-4** Peripheral-card I/O memory locations enabled by /IOSEL

Slot	Locations	Slot	Locations
1	$C100–$C1FF	5	$C500–$C5FF
2	$C200–$C2FF	6	$C600–$C6FF
3	$C300–$C3FF	7	$C700–$C7FF
4	$C400–$C4FF		

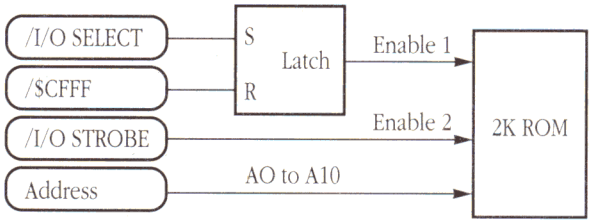
Expansion ROM space

In addition to the small areas of ROM memory allocated to each expansion slot, peripheral cards can use the 2K memory space from $C800 to $CFFE for larger programs in ROM or PROM. This memory space is called expansion ROM space. (See the memory map in Figure 8-7, shown later in this chapter.) Besides being larger, the expansion ROM memory space is always at the same locations, regardless of which slot is occupied by the card, making programs that occupy this memory space easier to write.

This memory space is available to any peripheral card that needs it. More than one peripheral card can use the expansion ROM space, but only one of them can be active at a time.

Each peripheral card that uses expansion ROM must have a circuit on it to enable the ROM. The circuit does this by a two-stage process: First, it sets a flip-flop when the /IOSEL signal, pin 1 on the slot, becomes active (low); the /IOSEL signal on a particular slot becomes active whenever the Apple IIGS microprocessor addresses a location in the 256-byte ROM address space allocated to that slot. Second, the circuit enables the expansion ROM devices when the /IOSTRB signal, pin 20 on the slot, becomes active (low); the /IOSTRB signal on all the expansion slots becomes active (low) when the microprocessor addresses a location in the expansion ROM memory space, $C800 to $CFFE. The /IOSTRB signal is then used to enable the expansion ROM devices on a peripheral card. Figure 8-5 shows a typical ROM enable circuit.

■ **Figure 8-5** Expansion ROM enable circuit

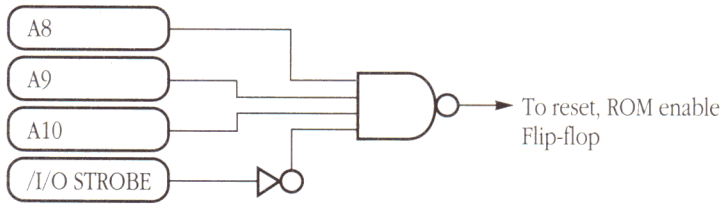


A program on a peripheral card can get exclusive use of the expansion ROM memory space by referring to location $CFFF in its initialization phase. This location is special: All peripheral cards that use expansion ROM must recognize a reference to $CFFF as a signal to disable their expansion ROMs. Of course, doing so also disables the expansion ROM on the card that is about to use it, but the next instruction in the initialization code sets the expansion ROM enable circuit on the card.

A card that needs to use the expansion ROM space must first insert its slot address ($Cn) in location $07F8 (known as MSLOT) before it refers to $CFFF. This allows interrupting devices to re-enable the card's expansion ROM after interrupt handling is finished. Once its slot address has been written in MSLOT, the peripheral card has exclusive use of the expansion memory space and its program can jump directly into the expansion ROM.

As described earlier, the expansion ROM disable circuit resets the enable flip-flop whenever the microprocessor addresses location $CFFF. To do this, the peripheral card must detect the presence of $CFFF on the address bus. You can use the /IOSTRB signal for part of the address decoding, since it is active for addresses from $C800 through $CFFF. If you can afford to sacrifice some ROM space, you can simplify the address decoding even further and save circuitry on the card. For example, if you give up the last 256 bytes of expansion ROM space, your disable circuit needs to detect only addresses of the form $CFxx, and you can use the minimal disable decoding circuitry shown in Figure 8-6.

■ **Figure 8-6** ROM disable address decoding



Peripheral-card RAM space

There are 56 bytes of main memory allocated to the peripheral cards, 8 bytes per card, as shown in Table 8-5. These 56 locations are actually in the RAM memory space reserved for the text and Lo-Res graphics displays, but these particular locations (called *screen holes*) are not displayed on the screen and their contents are not changed by the built-in output routine COUT1. Programs in ROM on peripheral cards use these locations for temporary data storage.

■ **Table 8-5** Peripheral-card RAM memory locations

Base address	Slot number						
	1	2	3	4	5	6	7
$0478	$0479	$047A	$047B	$047C	$047D	$047E	$047F
$04F8	$04F9	$04FA	$04FB	$04FC	$04FD	$04FE	$04FF
$0578	$0579	$057A	$057B	$057C	$057D	$057E	$057F
$05F8	$05F9	$05FA	$05FB	$05FC	$05FD	$05FE	$05FF
$0678	$0679	$067A	$067B	$067C	$067D	$067E	$067F
$06F8	$06F9	$06FA	$06FB	$06FC	$06FD	$06FE	$06FF
$0778	$0779	$077A	$077B	$077C	$077D	$077E	$077F
$07F8	$07F9	$07FA	$07FB	$07FC	$07FD	$07FE	$07FF

A program on a peripheral card can use the eight base addresses shown in the table to access the eight RAM locations allocated for its use, as shown in the next section, "I/O Programming Suggestions."

I/O programming suggestions

A program in ROM on a peripheral card should work no matter which slot the card occupies, excepting any hardware restrictions (such as a signal not available at some slots). If the program includes a jump to an absolute location in one of the 256-byte memory spaces, then the card will work only when it is plugged into the slot that uses that memory space. If you are writing the program for a peripheral card that will be used by many people, you should avoid placing such a restriction on the use of the card.

△ **Important** To function properly no matter which slot a peripheral card is installed in, the program in the card's 256-byte memory space must not make any absolute references to itself. Instead of using jump instructions, you should force conditions on branch instructions, which use relative addressing.

The first thing a peripheral card used as an I/O device must do when called is to save the contents of the microprocessor's registers. (Peripheral cards not being used as I/O devices do not need to save the registers.) The device should save the registers' contents on the stack, and restore them just before returning control to the calling program. If there is RAM on the peripheral card, the information may be stored there.

Finding the slot number with ROM switched in

The memory addresses used by a program on a peripheral card differ depending on which expansion slot the card is installed in. Before it can refer to any of those addresses, the program must somehow determine the correct slot number. One way to do this is to execute a JSR (jump to subroutine) to a location with an RTS (return from subroutine) instruction in it, and then derive the slot number from the return address saved on the stack, as shown in the following example.

△ **Important** Make sure the return address is located in Apple IIGS RAM, not the memory on the peripheral card.

```
PHP             ; save status
SEI             ; inhibit interrupts
JSR KNOWNRTS    ; ->a known RTS instruction...
                ;...that you set up
TSX             ; get high byte of the...
LDA $0100,X     ; ...return address from stack
AND #$0F        ; low-order digit is slot no.
PLP             ; restore status
```

The slot number can now be used in addressing the memory allocated to the peripheral card, as shown in the next section.

I/O addressing

Once your peripheral-card program has the slot number, the card can use the number to address the I/O locations allocated to the slot. Table 8-6 shows how these locations are related to 16 base addresses starting with $C080. Notice that the difference between the base address and the desired I/O location has the form $n0, where *n* is the slot number. Starting with the slot number in the accumulator, the following example computes this difference by four left shifts, then loads it into an **index register** and uses the base address to specify one of 16 I/O locations.

```
ASL             ; get n into...
ASL             ;
ASL             ;
ASL             ; ...high-order nibble...
TAX             ; ...of index register.
LDA  $C080,X    ; load from first I/O location
```

Table 8-6 Peripheral-card I/O base addresses

Base address	Slot number						
	1	2	3	4	5	6	7
$C080	$C090	$C0A0	$C0B0	$C0C0	$C0D0	$C0E0	$C0F0
$C081	$C091	$C0A1	$C0B1	$C0C1	$C0D1	$C0E1	$C0F1
$C082	$C092	$C0A2	$C0B2	$C0C2	$C0D2	$C0E2	$C0F2
$C083	$C093	$C0A3	$C0B3	$C0C3	$C0D3	$C0E3	$C0F3
$C084	$C094	$C0A4	$C0B4	$C0C4	$C0D4	$C0E4	$C0F4
$C085	$C095	$C0A5	$C0B5	$C0C5	$C0D5	$C0E5	$C0F5
$C086	$C096	$C0A6	$C0B6	$C0C6	$C0D6	$C0E6	$C0F6
$C087	$C097	$C0A7	$C0B7	$C0C7	$C0D7	$C0E7	$C0F7
$C088	$C098	$C0A8	$C0B8	$C0C8	$C0D8	$C0E8	$C0F8
$C089	$C099	$C0A9	$C0B9	$C0C9	$C0D9	$C0E9	$C0F9
$C08A	$C09A	$C0AA	$C0BA	$C0CA	$C0DA	$C0EA	$C0FA
$C08B	$C09B	$C0AB	$C0BB	$C0CB	$C0DB	$C0EB	$C0FB
$C08C	$C09C	$C0AC	$C0BC	$C0CC	$C0DC	$C0EC	$C0FC
$C08D	$C09D	$C0AD	$C0BD	$C0CD	$C0DD	$C0ED	$C0FD
$C08E	$C09E	$C0AE	$C0BE	$C0CE	$C0DE	$C0EE	$C0FE
$C08F	$C09F	$C0AF	$C0BF	$C0CF	$C0DF	$C0EF	$C0FF

- *Selecting your target*: You must make sure that you get an appropriate value into the index register when you address I/O locations this way. For example, starting with 1 in the accumulator, the instructions in the above example perform an LDA from location $C090, the first I/O location allocated to slot 1. If the value in the accumulator (discussed in detail in Chapter 10) had been 0, the LDA would have accessed location $C080, thereby setting the soft switch that selects the second bank of RAM at location $D000 and enables it for reading.

RAM addressing

A program on a peripheral card can use the eight base addresses shown in Table 8-5 to access the eight RAM locations allocated for its use. The program does this by putting its slot number into the Y Index register (discussed in detail in Chapter 10) and using indexed addressing mode with the base addresses. The base addresses can be defined as constants because they are the same no matter which slot the peripheral card occupies.

If you start with the correct slot number in the accumulator (by using the example shown earlier in this chapter, in the section "Finding the Slot Number With ROM Switched In"), then the following example uses all eight RAM locations allocated to the slot:

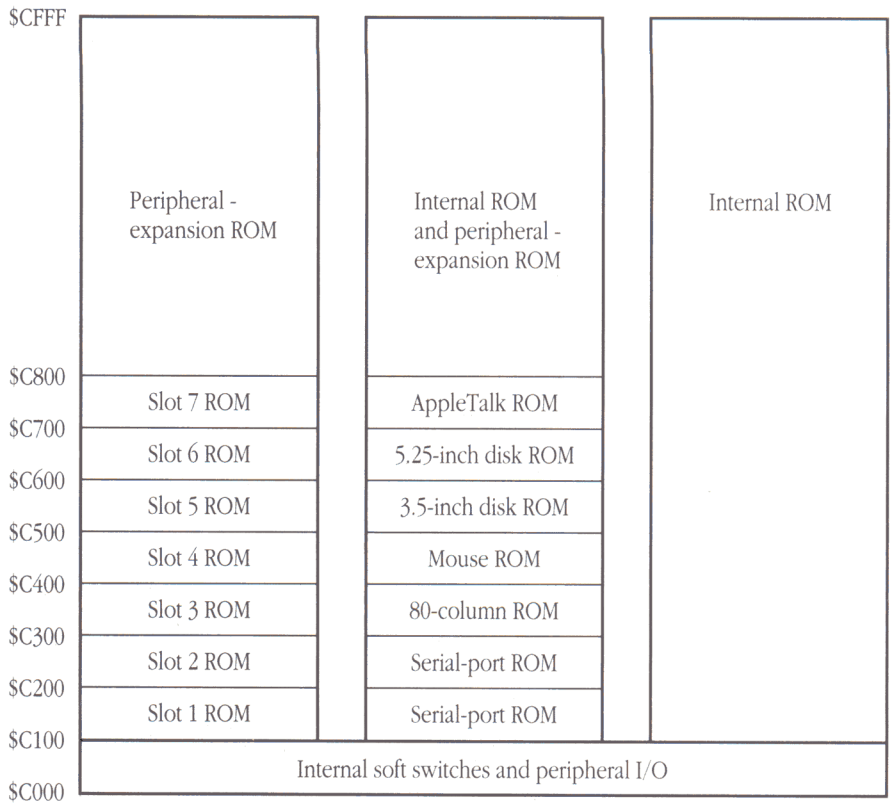
```
TAY
LDA   $0478,Y
STA   $04F8,Y
LDA   $0578,Y
STA   $05F8,Y
LDA   $0678,Y
STA   $06F8,Y
LDA   $0778,Y
STA   $07F8,Y
```

▲ Warning You must be very careful when you have your peripheral-card program store data at the base-address locations themselves because they are temporary storage locations; the RAM at those locations is used by the disk operating system. Always store the first byte of the ROM location of the expansion slot that is currently active ($Cn) in location $07F8 (MSLOT), and the first byte of the ROM location of the slot holding the controller card for the startup disk drive in location $05F8. ▲

Other uses of I/O memory space

The portion of memory space from location $C000 through $CFFF is normally allocated to I/O and program memory on the peripheral cards, but this computer has built-in functions that also use this memory space. Figure 8-7 shows the division of the address spaces assigned to the built-in and peripheral devices. The soft switches that control the allocation of this memory space are described in the next section.

■ **Figure 8-7** I/O memory map



Switching I/O memory

The built-in firmware uses two sets of soft switches to control the allocation of the I/O memory space from $C000 to $CFFF. The locations of these soft switches are given in Table 8-7.

◆ *Note:* Like the display switches described earlier in this chapter, these soft switches share their locations with the keyboard data and strobe functions. The switches are activated only by writing, and the states can be determined only by reading, as indicated in Table 8-7.

■ **Table 8-7** I/O memory switches

Name	Function	Location		Notes
		Hex	Dec	
SETSLOTC3ROM	Enable slot ROM at $C300	$C00B	49163	Write
SETINTC3ROM	Enable internal ROM at $C300	$C00A	49162	Write
RDC3ROM	Read SLOTC3ROM switch	$C017	49175	Read (1 = slot 3 ROM enabled, 0 = internal ROM enabled)
SETSLOTCXROM	Enable slot ROM at $Cx00	$C006	49159	Write
SETINTCXROM	Enable internal ROM at $Cx00	$C007	49158	Write
RDCXROM	Read SLOTCXROM switch	$C015	49173	Read (1 = slot ROM enabled, 0 = internal ROM enabled)

When SETSLOTC3ROM is on, the 256-byte ROM area at $C300 is available to a peripheral card in slot 3, which is the slot normally used for a terminal interface. Turning SETINTC3ROM on disables peripheral-card ROM in slot 3 and enables the built-in 80-column firmware. The 80-column firmware is assigned to the slot 3 address space because slot 3 is normally used with a terminal interface, so the firmware built into the Apple IIGS will work with programs that use slot 3 this way.

The bus and I/O signals are always available to a peripheral card in slot 3, even when the 80-column hardware and firmware are operating. Thus it is always possible to use this slot for any I/O peripheral that does not have built-in firmware.

When SETSLOTCXROM is on, the I/O memory space from $C100 to $C7FF is allocated to the expansion slots, as described earlier in this chapter, in the section "Peripheral-Card ROM Space." Setting SETINTCXROM disables the peripheral-card ROM and selects built-in ROM in all of the I/O memory space except the space from $C000 to $C0FF (used for soft switches and data I/O).

◆ *Note:* Setting SETINTCXROM enables built-in ROM in all of the I/O memory space (except the soft-switch area), including the $C300 space, which contains the 80-column firmware.

Developing cards for slot 3

In the original Apple IIe firmware, the internal slot 3 firmware was always switched on if there was an **80-column text card** (either 1K or 64K) in the auxiliary slot. This means that peripheral cards with their own ROM were effectively switched out of slot 3 when the system was turned on.

In the Apple IIGS, only the Control Panel may determine whether or not the peripheral card in slot 3 is selected.

When programming for cards in slot 3:

- You must support the AUXMOVE and XFER firmware routines. See the *Apple IIGS Firmware Reference* for information on these routines.
- Don't use unpublished entry points into the internal $Cn00 firmware, because they may change in future Apple II firmware versions.
- If your peripheral card is a **character I/O device,** you must follow the Pascal 1.1 firmware protocol. See the *Apple IIGS Firmware Reference* for more information.

Interrupts

The original Apple IIe offered little firmware support for interrupts. The Apple IIGS firmware provides improved interrupt support. Interrupts are easiest to use with ProDOS and Pascal 1.2 because they have interrupt support built in. DOS 3.3 has no built-in interrupt support.

The main purpose of the **interrupt handler** is to support interrupts in any memory configuration. You can best handle interrupts by saving the machine's state at the time of the interrupt, placing the Apple IIGS in a standard memory configuration before calling your program's interrupt handler, then restoring the original state when your program's interrupt handler is finished.

What is an interrupt?

An interrupt is a hardware signal that tells the computer to stop what it is currently doing and devote its attention to a more important task. Print spooling and mouse handling are examples of interrupt use: things that don't take up all the time available to the system, but that should be taken care of promptly to be most useful.

For example, the Apple IIGS mouse can send an interrupt to the computer every time it moves. If you handle that interrupt promptly, the mouse pointer's movement on the screen will be smooth instead of jerky and uneven.

Interrupt priority is handled by a daisy-chain arrangement using two pins, INT IN and INT OUT, on each peripheral-card slot. Each peripheral card breaks the chain when it makes an interrupt request. On peripheral cards that don't use interrupts, these pins should be connected together.

The daisy chain gives priority to the peripheral card in slot 7: If this card opens the connection between INT IN and INT OUT, or if there is no card in this slot, interrupt requests from cards in slots 1 through 6 can't get through. Similarly, slot 6 controls interrupt requests (IRQ) from slots 1 through 5, and so on down the line.

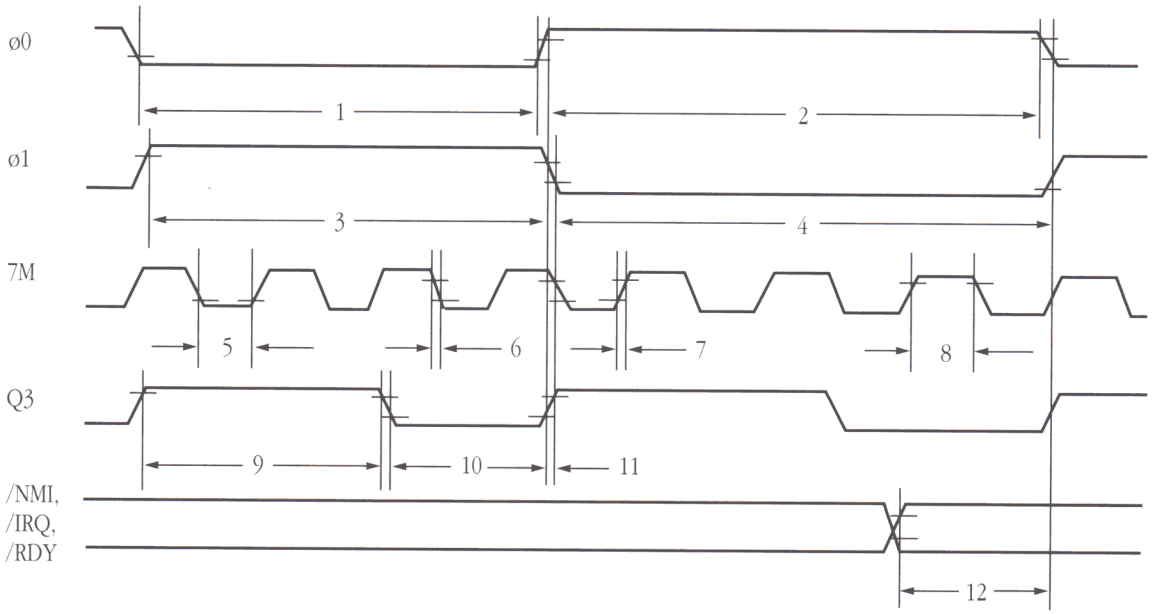
When the /IRQ line on the Apple IIGS microprocessor is activated (pulled low), the microprocessor transfers control through the vector in locations $FFFE to $FFFF. This vector is the address of the Monitor firmware's interrupt handler, which determines whether the request is due to an external IRQ or a BRK instruction and transfers control to the appropriate routine via the vectors stored in memory Page 3. (For further details on handling interrupts in the Apple IIGS, see the *Apple IIGS Firmware Reference.*)

The interrupt ROM code is available when shadowing is enabled *and* the inhibit I/O and language-card operation (IOLC) bit in the Shadow register is set. The SETINTCXROM and SETSLOTCXROM soft switches do not affect interrupt ROM accesses.

Timing diagrams

The following pages contain timing diagrams for the slot signals required to handle DMA and general slot I/O.

■ **Figure 8-8** I/O clock and control timing



■ **Table 8-8** I/O clock and control timing parameters

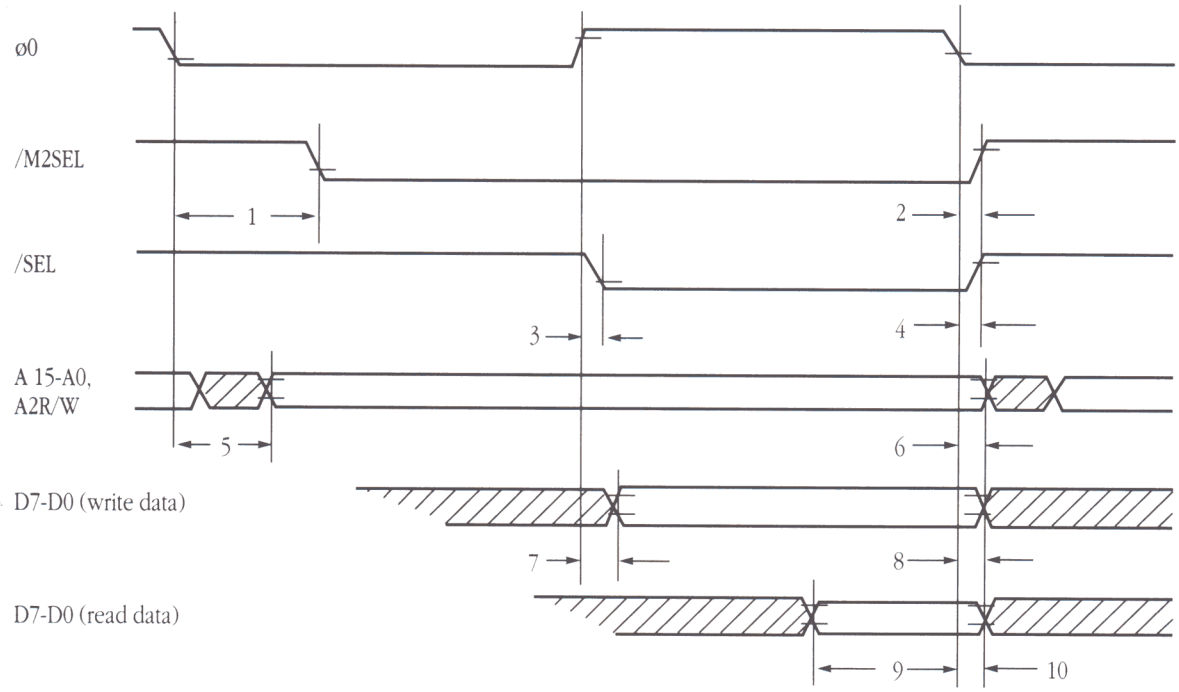
Number	Description	Minimum*	Maximum*
1	ø0 low time	480	
2	ø0 high time	480	
3	ø1 high time	480	
4	ø1 low time	480	
5	7M low time	60	
6	Fall time, all clocks	0	10
7	Rise time, all clocks	0	10
8	7M high time	60	
9	Q3 high time	270	
10	Q3 low time	200	
11	Skew, ø0 to other clock signals	–10	10
12	Control signal setup time	140	

* Time in nanoseconds.

◆ *Note:* All clock signals present on the I/O slots are buffered by the Slotmaker custom IC. These clock signals are delayed somewhat from the corresponding signals on the main logic board because of this buffering. All timing parameters in the timing diagrams in this chapter have been adjusted to account for this delay.

The standard Apple IIGS slot I/O timing is shown in Figure 8-9. The timing parameters are given in Table 8-9. When the computer is running in high-speed mode (2.8 MHz), the address bus to the I/O slots is not valid during the entire ø0 cycle, and therefore cannot be used to perform unqualified address decoding. The /M2SEL signal (which replaces the μSYNC signal found at pin 39 in previous Apple II models), indicates when a 1.024-MHz, synchronized memory cycle is taking place and, therefore, when the value on the address bus will remain valid during the current ø0 cycle. This means that cards that use the Apple II technique of "phantom slotting" to put multiple I/O devices on one card must use /M2SEL to qualify their address decoding.

■ **Figure 8-9** I/O read and write timing



■ **Table 8-9** I/O read and write timing parameters

Number	Description	Minimum*	Maximum*
1	/M2SEL low from ø0 low	0	160
2	/M2SEL hold time	-10	
3	I/O enable low from ø0 high (DEVn, /IOSELn, /IOSTRB)	0	15
4	I/O enable high from ø0 low (DEVn, /IOSELn, /IOSTRB)	10	
5	Address and A2R/W valid from ø0 low	0	100
6	Address and A2R/W hold time	15	
7	Write data valid delay	0	30
8	Write data hold time	30	
9	Read data setup time to ø0	140	
10	Read data hold time	10	

* Time in nanoseconds.

Read and write cycles that are directed to the I/O slots by /INH have the same timing parameters as normal I/O read and write cycles, as shown in Figure 8-10 and Table 8-10. When /INH is asserted, the computer responds as if a Mega II memory cycle were being performed.

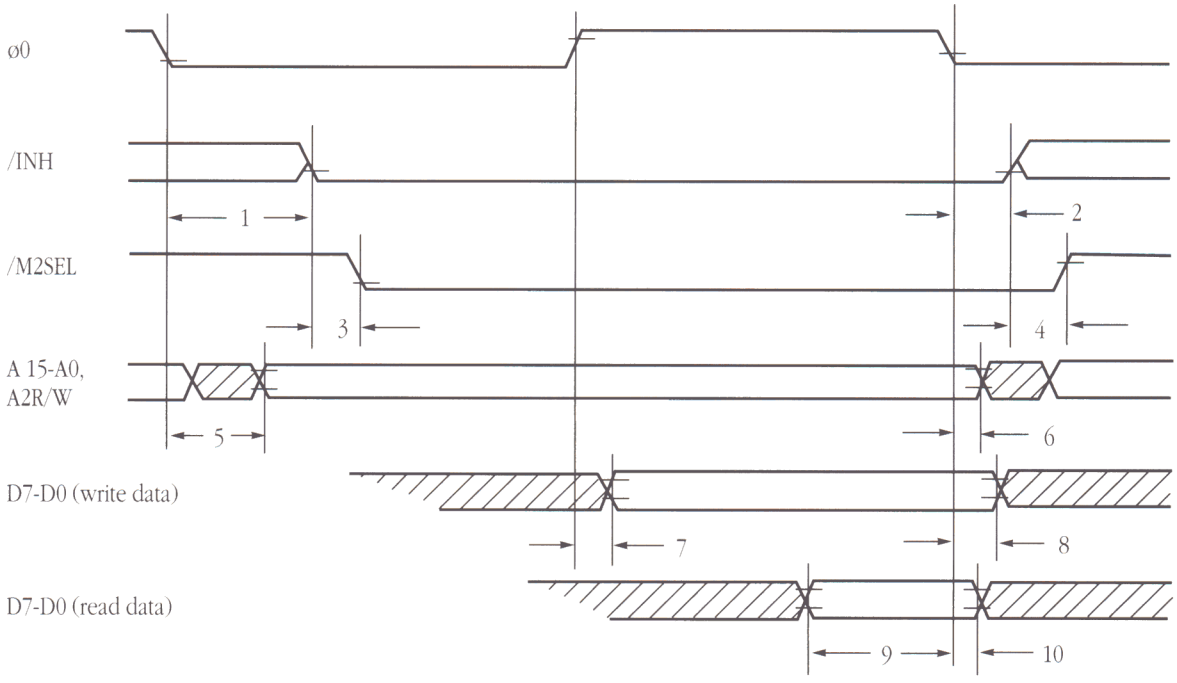
■ **Table 8-10** I/O read and write timing parameters with /INH active

Number	Description	Minimum*	Maximum*
1	/INH valid after ø0 low	0	175
2	/INH hold time	15	
3	/INH low to /M2SEL low delay	0	30
4	/INH high to /M2SEL high delay	0	30
5	Address and A2R/W valid from ø0 low	0	100
6	Address and A2R/W hold time	15	
7	Write data valid delay	30	
8	Write data hold time	30	
9	Read data setup time to ø0	140	
10	Read data hold time	10	

* Time in nanoseconds.

Cards that use the /INH signal will function properly only if the computer is running at 1.024 MHz. If the computer is running at high speed, the addresses that are seen by cards in the I/O slots are not guaranteed to be valid during an entire ø0 cycle. Also, since the upper 8 bits of the memory address are not available to cards (only 16 address lines are available at the slots), the potential of /INH is greatly reduced in this machine.

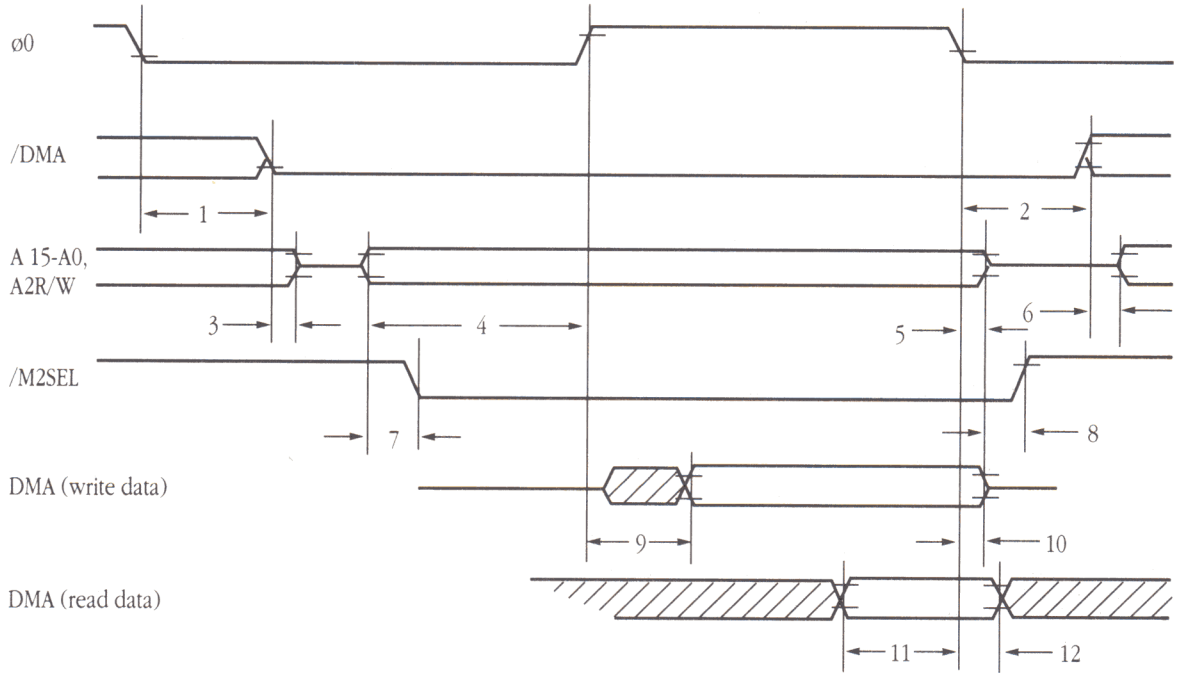
■ **Figure 8-10** I/O read and write timing with /INH active



DMA devices will work in the Apple IIGS computer only in 1.024-MHz mode. If the computer is running at high speed (2.8 MHz), only DMA accesses to the high-speed memory banks 0 through 127 will work. Accesses to all I/O and video memory must be done at 1.024 MHz. To slow the system, set the processor speed bit in the Speed register at location $C036 before requesting DMA.

DMA can be performed to or from any part of the Apple IIGS memory map, provided that the DMA bank register is first set to the appropriate bank. DMA read and write timing is shown in Figure 8-11 and Table 8-11.

■ **Figure 8-11** /DMA read and write timing



■ **Table 8-11** /DMA read and write timing parameters

Number	Description	Minimum*	Maximum*
1	/DMA low from ø0 low		120
2	/DMA high from ø0 low		120
3	A15–A0 and R/W float from /DMA		30
4	DMA address and A2R/W valid before ø0 goes high	300	
5	DMA address and A2R/W hold time	10	
6	/DMA high to A15–A0 and A2R/W active		30
7	DMA address valid to /M2SEL low		30
8	DMA address float to /M2SEL high		30
9	ø0 high to write data valid	100	
10	DMA write data hold time	10	
11	DMA read data setup time	125	
12	DMA read data hold time	30	

* Time in nanoseconds.

Chapter 9 **Power Supply**

The Apple IIGS power supply has the same four-supply, switching, load-sensing design as the Apple II, II Plus, and IIe models. The following sections describe the design of this unit.

Description

The power supply changes high-voltage alternating current (AC) into low-voltage direct current (DC). The Apple IIGS does this by using a switching-type power supply that allows simple, maintenance-free operation.

▲ **Warning** The power supply contains dangerously high voltages, and should be opened by an authorized Apple service technician only. ▲

The power supply also contains special load-sensing circuitry; whenever this circuitry detects a short or a no-load condition, the power supply will no longer provide voltages to the computer. This condition is easily recognized: The supply will emit two audible chirps per second. This condition will persist until you correct the situation or turn the power supply off.

Specifications

The Apple IIGS power supply operates on regular household 120-volt alternating current. The power supply provides +12 volts, –12 volts, +5 volts, –5 volts, and two ground-return lines.

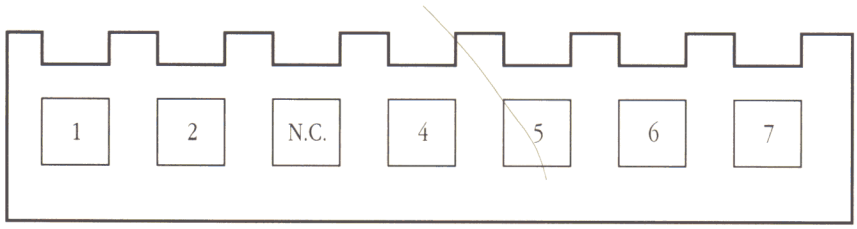
The power input requirements are 107- to 132-volt alternating current. The power output specifications are as follows:

- +12 volts at 1 A
- –12 volts at 0.25 A
- +5 volts at 4 A
- –5 volts at 0.25 A

Power connector

The power connector is a 6-pin, molex-type, keyed in-line socket. Figure 9-1 shows the pin configuration of the power connector. Table 9-1 gives a description of each pin.

■ **Figure 9-1** Power-supply connector



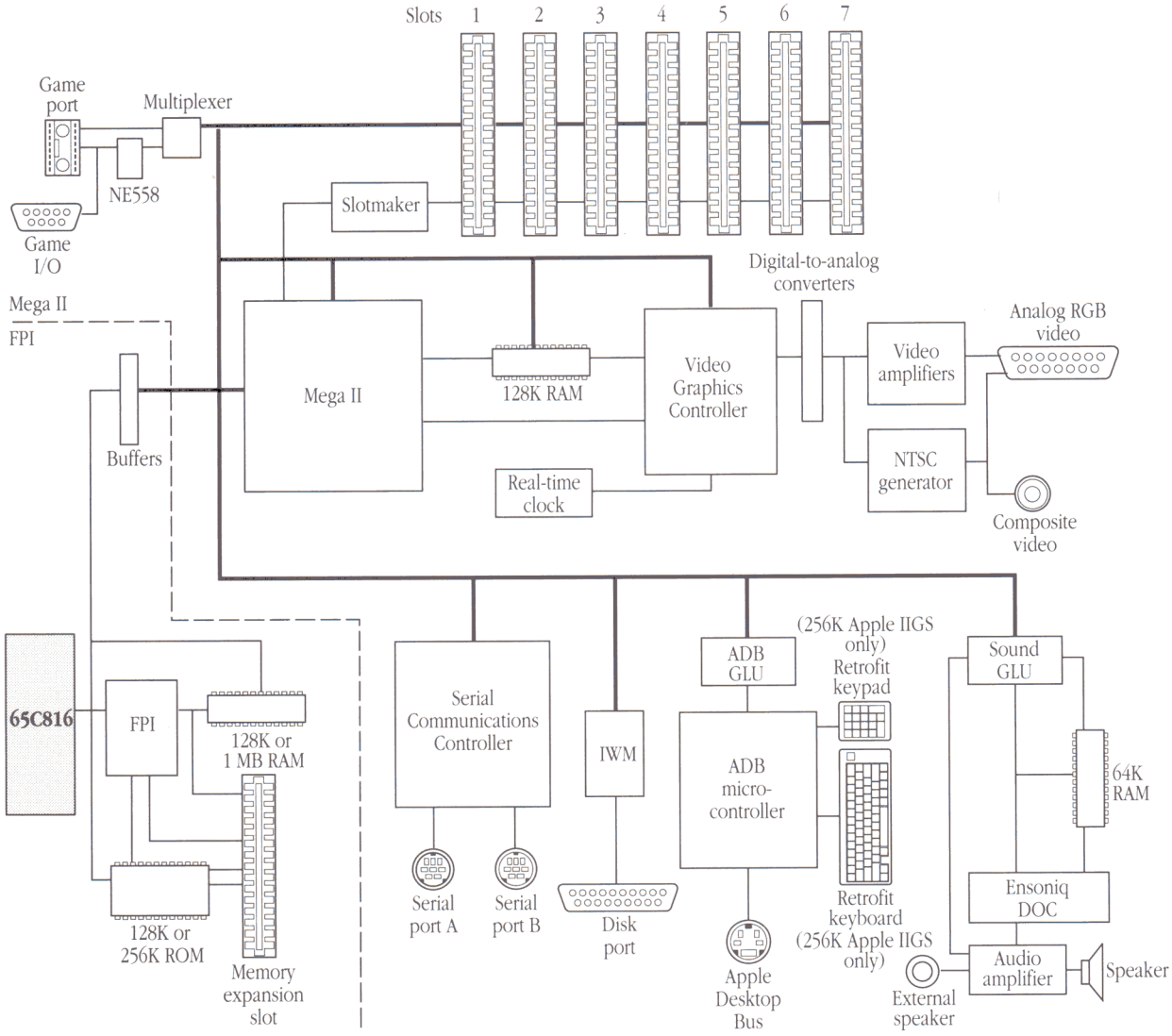
■ **Table 9-1** Pins on the power-supply connector

Pin	Signal	Description
1	GND	Ground
2	GND	Ground
3	N.C.	No connection
4	+5V	+5-volt supply
5	+12V	+12-volt supply
6	–12V	–12-volt supply
7	–5V	–5-volt supply

Chapter 10 The 65C816 Microprocessor

The microprocessor is the intelligence of the computer system. It is this device that recognizes the instructions encoded by the programmer and manipulates the other devices in the system (VGC, the Mega II, the DOC) that result in output such as video and sound. Figure 10-1 shows the Apple IIGS block diagram and the relationship of the microprocessor to the rest of the computer.

■ **Figure 10-1** 65C816 in the Apple IIGS system



The Apple IIGS uses the 16-bit 65C816 microprocessor, a CMOS design based on the 6502 chip. The microprocessor provides this computer with greater computing power in these ways:

- It can operate as a 16-bit 65C816 or an 8-bit 6502.
- The 4 MB address range increases the potential program and data size.
- The 16-bit internal data registers increase the computer's data-handling capability.
- The 2.8-MHz processor speeds computations.

This chapter describes the new features of this microprocessor and its capability to emulate the 6502. Also, each of the 65C816 internal registers is described briefly.

The features of the 65C816

The 65C816 microprocessor shares many characteristics with the 6502 and 65C02 used in other Apple II–family computers. It also introduces new features not found in other Apple II computers. These are

- the 16-bit accumulator
- the 16-bit X and Y Index registers
- the relocatable direct page
- the relocatable stack
- the 24-bit internal address bus
- the 8-bit data address bank register
- the 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
- the ability to emulate the 6502 8-bit microprocessor

For detailed descriptions of these features, refer to the manufacturers' data sheets at the end of this chapter. To learn how to implement these features, refer to the *Apple IIGS Workshop Assembler Reference.*

The 65C816 microprocessor shares some features with the 6502 and 65C02 microprocessors used in previous Apple II models. Table 10-1 lists some of these features.

The 65C816 is software compatible with the 6502 family of microprocessors. Actually, the 65C816 has an emulation mode, in which it becomes an 8-bit 6502. By emulating the 6502, the 65C816 can execute most programs written for Apple II computers.

■ **Table 10-1** Some 6500 family ties

Characteristic	6502 1975*	65C02 1983*	65C816 1985*
Construction	**NMOS**	CMOS	CMOS
ALU bits	8	8	16
Address bus bits	16	16	24†
Data bus bits	8	8	8
Maximum memory	64K	64K	16M
Largest stack	256	256	64K
Defined **opcodes**	151	178	256
Addressing modes	13	15	24
Relocatable direct (zero) page?	No	No	Yes
6502 software compatible?	Yes	Yes	Yes
Fast block-move instructions?	No	No	Yes

* Year available.

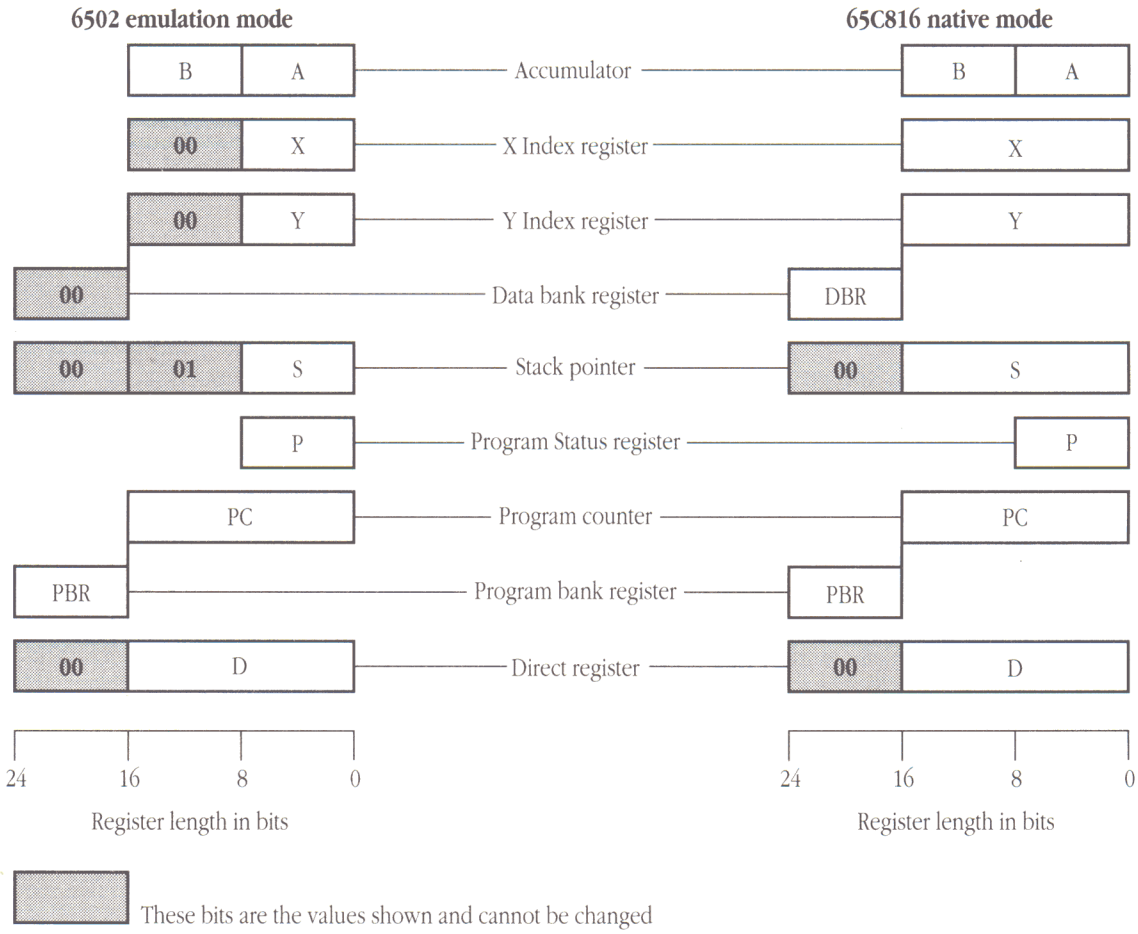
† High 8 bits multiplexed onto data bus.

The 16-bit 65C816

In the Apple IIGS, the 65C816 normally operates in either of two modes: 6502 emulation mode or 65C816 native mode. Figure 10-2 shows the sizes of the 65C816's internal 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 16 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 the new addressing modes 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.

■ **Figure 10-2** 65C816 registers



Microprocessor differences

The 65C816 microprocessor differs from the 6502 in several ways. This section describes some of those differences and their impact on program execution.

The registers

The 65C816 contains all the registers found in the 6502. In addition, the new microprocessor has three additional registers that make it a more powerful chip. These new registers provide additional addressing capability and greater data-handling capability. The nine registers within the 65C816 are described below. (To learn how to use the registers in the 65C816, see the *Apple IIGS Programmer's Workshop Assembler Reference.*)

The accumulator

The accumulator (also known as the Arithmetic Logic Unit—ALU) is a 16-bit register that holds all values while arithmetic and logical calculations are performed. The entire 16 bits of the accumulator in the 65C816 is available in both emulation mode as well as native mode. The low half of the accumulator is called the A register, while the upper half is referred to as the B register. Some documentation on the 65C816 refers to the entire 16-bit register as the C register, but within this manual, we will refer to the register as the accumulator (all 16 bits), or the A or B register (8-bit values within the accumulator).

The results of calculations within this register affect the status bits in the Program Status register.

X Index register

The X Index register is a 16-bit register that is used as an address offset value when calculating an effective address. When the Program Status register x bit is set, the upper 8 bits of the X Index register are filled with 0's that cannot be altered.

Y Index register

The Y Index register is a 16-bit register that is used as an address offset value when calculating an effective address. When the Program Status register x bit is set, the upper 8 bits are filled with 0's that cannot be altered.

Data bank register

In native mode, the data bank register is an 8-bit register that contains the most significant byte of the effective 24-bit address specified in the X or Y Index registers when only 16 bits of the address are provided. In emulation mode, it contains 0's that cannot be altered.

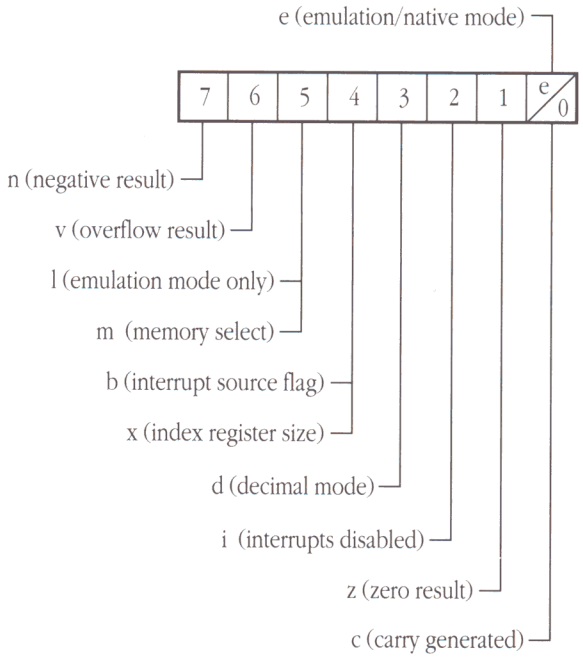
Stack pointer

In the earlier Apple II computers, the stack was located at $100 through $1FF in memory. In the 65C816, the stack can be located anywhere in bank $00, but may not exceed 64K. The stack pointer contains the address of the next available stack location. The stack "grows" in a downward direction (toward lower addresses just as with a 6502 stack); **PUSH** and **PULL** instructions place and remove bytes from the "top" of the stack (actually the lowest address).

Program Status register

The Program Status register is an 8-bit register that contains status bits that are set or cleared as a result of the condition of the accumulator after each operation within the accumulator. Also, this register contains the e, m, and x bits that control the emulation mode and register size. This register remains 8 bits in size in both native and emulation modes. Figure 10-3 shows the format of the Program Status register. Table 10-2 describes these bits.

■ **Figure 10-3** Program Status register



■ **Table 10-2** Bits in the Program Status register

Bit	Value	Description
7	0	n (negative result): This bit reflects the high bit of the accumulator (bit 7 or bit 15, depending upon the state of the m bit). When it is 0, the value in the register is a signed positive value.
	1	This bit will be set to 1 if the last operation in the accumulator resulted in a negative value, which will set the accumulator high bit.
6	0	v (accumulator overflow): This bit indicates whether or not an overflow condition has occurred. An overflow is when an arithmetic result is greater than can be represented by the register (either 8 or 16 bits, depending upon the state of the m bit). The overflow bit is set when the carry out of the most significant bit is different from the carry out of the next most significant bit. The value of the overflow bit may also be described as the exclusive-OR of the carry into and out of the most significant bit. When this bit is 0, no overflow has occurred.
	1	This bit will be set to 1 if the last operation in the accumulator resulted in an overflow.
5	0	m (memory and accumulator size): Setting this bit to 0 will set the accumulator size to 16 bits.
	1	Setting this bit to 1 will set the accumulator size to 8 bits. In emulation mode, this bit is permanently set to 1.
4	0	x (Index register size): Setting this bit to 0 will set the X and Y Index register sizes to 16 bits. The Index register size cannot be determined by reading this bit. (See "break flag," below).
	1	Setting this bit to 1 will set the X and Y Index register sizes to 8 bits. In emulation mode this bit is permanently set to 1.
		b (break flag): Reading bit 4 will reflect the state of an interrupt. If a 0 is read in this bit position, the current interrupt was caused by software. If a 1 is read in this bit position, the current interrupt was caused by hardware.

■ **Table 10-2** Bits in the Program Status register (Continued)

Bit	Value	Description
3	0	d (decimal mode): Setting this bit to 0 instructs the microprocessor to carry out mathematical calculations in hexadecimal, representing values in hex values between $0 and $F.
	1	Setting this bit to 1 instructs the microprocessor to treat all mathematical computations as binary-coded decimal, limiting the represented values to numbers between 0 and 9.
2	0	i (interrupts disabled): Setting this bit to 0 allows interrupt requests to be serviced.
	1	Setting this bit to 1 disables all interrupt requests that are made.
1	0	z (zero result): This bit will be 0 if the last operation resulted in a nonzero value. This flag is not limited to use with arithmetic calculations.
	1	This bit will be 1 if the last operation resulted in a value of 0.
0	0	c (carry generated): This bit will be 0 if the last arithmetic calculation did not result in a carry.
	1	This bit will be 1 if the last arithmetic calculation generated a carry.
	0	e (emulation mode): This bit sits "behind" the carry bit and is exchanged with the carry bit. Setting this bit to 1 places the microprocessor in 6502 emulation mode. In this mode, the Index registers (X and Y) become 8 bits. Stack commands and arithmetic calculations are 8-bit operations. All others remain 16-bit operations.
	1	Setting this bit to 1 places the microprocessor in native, 16-bit mode. All operations in this mode are 16-bit operations.

Program counter

The program counter is a 16-bit register that is concatenated with the program bank register to obtain the resulting 24-bit address of the next instruction to be fetched for execution. Note that the upper 8 bits of the program counter will not increment across page boundaries. In emulation mode, this register retains its 24-bit size. (See the next section, "Program Bank Register.")

Program bank register

The program bank register is an 8-bit register that contains the most significant byte of the 24-bit program counter address. In emulation mode this register is available, although limited in its use.

Direct register

In the previous Apple computers, the zero page (called the direct page in the 65C816) was located in the low $100 bytes of memory, and could not be moved. In the 65C816, the direct page can be located anywhere in bank $00. The starting (low-byte) address of the direct page is determined by the Direct register. This address can be any value from $0000 through $FF00. Although the direct page can begin anywhere in bank $00, there is a one-cycle penalty when it does not begin on a page boundary (when the low byte of the Direct register is not $00). In emulation mode (e bit = 1), the high 8 bits of this register are set to $01, and all stack references are limited to page 1.

Emulating the 6502

As mentioned earlier, the 65C816 is capable of emulating a 6502 microprocessor. In emulation mode, the 65C816 will execute the complete 65C816 instruction set (which includes all 6502 instructions), but many of these instructions will be of limited use because of the reduced width of the registers. For instance, the X Index and Y Index registers are 16 bits wide in native mode and are reduced to 8 bits in emulation mode. Note in Figure 10-2 that certain bits in some of the registers are filled with specific values that cannot be altered when the m bit is set.

To emulate the 6502 microprocessor, set the e bit to 1. You may then run programs that were written for the 6502.

Operating speed

The Apple IIGS can run the 65C816 processor at one of two speeds: 1.024 MHz and 2.8 MHz. The FPI controls the clock input signal to the microprocessor and selects the appropriate speed as indicated by the clock speed bit in the Speed register.

Further reading

To learn more about programming the 65C816 microprocessor, read the following books:

Fischer, Michael. *65816/65802 Assembly Language Programming*. Berkeley, CA: Osborne/McGraw-Hill, 1986.

Eyes, David, and Ron Lichty. *Programming the 65816*. New York: Brady/Prentice-Hall, 1986.

65C816 data sheets

On the following pages are the data sheets from one of the two manufacturers of the 65C816 microprocessor.

W65C816

CMOS W65C816 and W65C802 16-Bit Microprocessor Family

Features

- Advanced CMOS design for low power consumption and increased noise immunity
- Single 3-6V power supply, 5V specified
- Emulation mode allows complete hardware and software compatibility with 6502 designs
- 24-bit address bus allows access to 16 MBytes of memory space
- Full 16-bit ALU, Accumulator, Stack Pointer, and Index Registers
- Valid Data Address (VDA) and Valid Program Address (VPA) output allows dual cache and cycle steal DMA implementation
- Vector Pull (VP) output indicates when interrupt vectors are being addressed. May be used to implement vectored interrupt design
- Abort (ABORT) input and associated vector supports virtual memory system design
- Separate program and data bank registers allow program segmentation or full 16-MByte linear addressing
- New Direct Register and stack relative addressing provides capability for re-entrant, re-cursive and re-locatable programming
- 24 addressing modes—13 original 6502 modes, plus 11 new addressing modes with 91 instructions using 255 opcodes
- New Wait for Interrupt (WAI) and Stop the Clock (STP) instructions further reduce power consumption, decrease interrupt latency and allows synchronization with external events
- New Co-Processor instruction (COP) with associated vector supports co-processor configurations, i.e., floating point processors
- New block move ability

General Description

WDC's W65C802 and W65C816 are CMOS 16-bit microprocessors featuring total software compatibility with their 8-bit NMOS and CMOS 6500-series predecessors. The W65C802 is pin-to-pin compatible with 8-bit devices currently available, while the W65C816 extends addressing to a full 16 megabytes. These devices offer the many advantages of CMOS technology, including increased noise immunity, higher reliability, and greatly reduced power requirements. A software switch determines whether the processor is in the 8-bit "emulation" mode, or in the native mode, thus allowing existing systems to use the expanded features.

As shown in the processor programming model, the Accumulator, ALU, X and Y Index registers, and Stack Pointer register have all been extended to 16 bits. A new 16-bit Direct Page register augments the Direct Page addressing mode (formerly Zero Page addressing). Separate Program Bank and Data Bank registers allow 24-bit memory addressing with segmented or linear addressing.

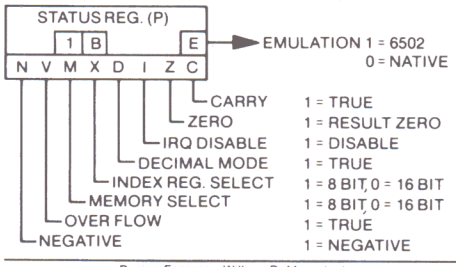
Four new signals provide the system designer with many options. The ABORT input can interrupt the currently executing instruction without modifying internal register, thus allowing virtual memory system design. Valid Data Address (VDA) and Valid Program Address (VPA) outputs facilitate dual cache memory by indicating whether a data segment or program segment is accessed. Modifying a vector is made easy by monitoring the Vector Pull (VP) output.

Note: To assist the design engineer, a Caveat and Application Information section has been included within this data sheet.

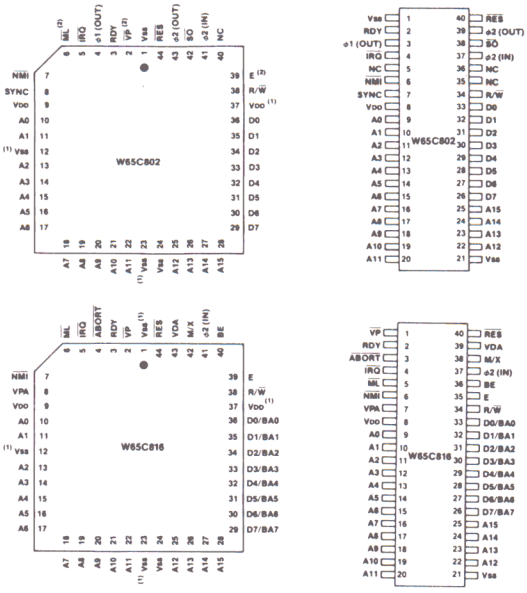
W65C816 Processor Programming Model

8 BITS	8 BITS		8 BITS
Data Bank Reg. (DBR)	X Register Hi (XH)	(X)	X Register Low (XL)
Data Bank Reg. (DBR)	Y Register Hi (YH)	(Y)	Y Register Low (YL)
00	Stack Register Hi (SH)	(S)	Stack Reg. Low (SL)
= 6502 Registers	Accumulator (B)	(C)	Accumulator (A)
Program Bank Reg. (PBR)	Program (PCH)	(PC)	Counter (PCL)
00	Direct Reg. Hi (DH)	(D)	Direct Reg. Low (DL)

Status Register Coding



Pin Configuration



For notes, refer to Packaging Information section.

Design Engineer: William D. Mensch, Jr.

THE WESTERN DESIGN CENTER, INC.
2166 East Brown Road • Mesa, Arizona 85203 • 602-962-4545

Advance Information Data Sheet:
This is advanced information and specifications are subject to change without notice.

Absolute Maximum Ratings: (Note 1)

Rating	Symbol	Value
Supply Voltage	V_{DD}	-0.3V to +7.0V
Input Voltage	V_{IN}	-0.3V to V_{DD} +0.3V
Operating Temperature	T_A	0°C to +70°C
Storage Temperature	T_S	-55°C to +150°C

This device contains input protection against damage due to high static voltages or electric fields; however, precautions should be taken to avoid application of voltages higher than the maximum rating.

Notes:

1. Exceeding these ratings may cause permanent damage. Functional operation under these conditions is not implied.

DC Characteristics (All Devices): V_{DD} = 5.0V ±5%, V_{SS} = 0V, T_A = 0°C to +70°C

Parameter	Symbol	Min	Max	Unit
Input High Voltage	V_{IH}			
RES, RDY, IRQ, Data, SO, BE,		2.0	V_{DD} + 0.3	V
φ2 (IN), NMI, ABORT		0.7 V_{DD}	V_{DD} + 0.3	V
Input Low Voltage	V_{IL}			
RES, RDY, IRQ, Data, SO, BE,		-0.3	0.8	V
φ2 (IN), NMI, ABORT		-0.3	0.2	V
Input Leakage Current (V_{IN} = 0 to V_{DD})	I_{IN}			
RES, NMI, IRQ, SO, BE, ABORT (Internal Pullup)		-100	1	μA
RDY (Internal Pullup, Open Drain)		-100	10	μA
φ2 (IN)		-1	1	μA
Address, Data, R/W (Off State, BE = 0)		-10	10	μA
Output High Voltage (I_{OH} = -100μA)	V_{OH}			
SYNC, Data, Address, R/W, ML, VP, M/X, E, VDA, VPA,				
φ1 (OUT), φ2 (OUT)		0.7 V_{DD}	—	V
Output Low Voltage (I_{OL} = 1.6mA)	V_{OL}			
SYNC, Data, Address, R/W, ML, VP, M/X, E, VDA, VPA,				
φ1 (OUT), φ2 (OUT)		—	0.4	V
Supply Current (No Load)	I_{DD}		4	mA/MHz
Standby Current (No Load, Data Bus = V_{SS} or V_{DD})	I_{SB}	—		
RES, NMI, IRQ, SO, BE, ABORT, φ2 = V_{DD})			10	μA
Capacitance (V_{IN} = 0V, T_A = 25°C, f = 2 MHz)				
Logic, φ2 (IN)	C_{IN}	—	10	pF
Address, Data, R/W (Off State)	C_{TS}	—	15	pF

Pin Function Table

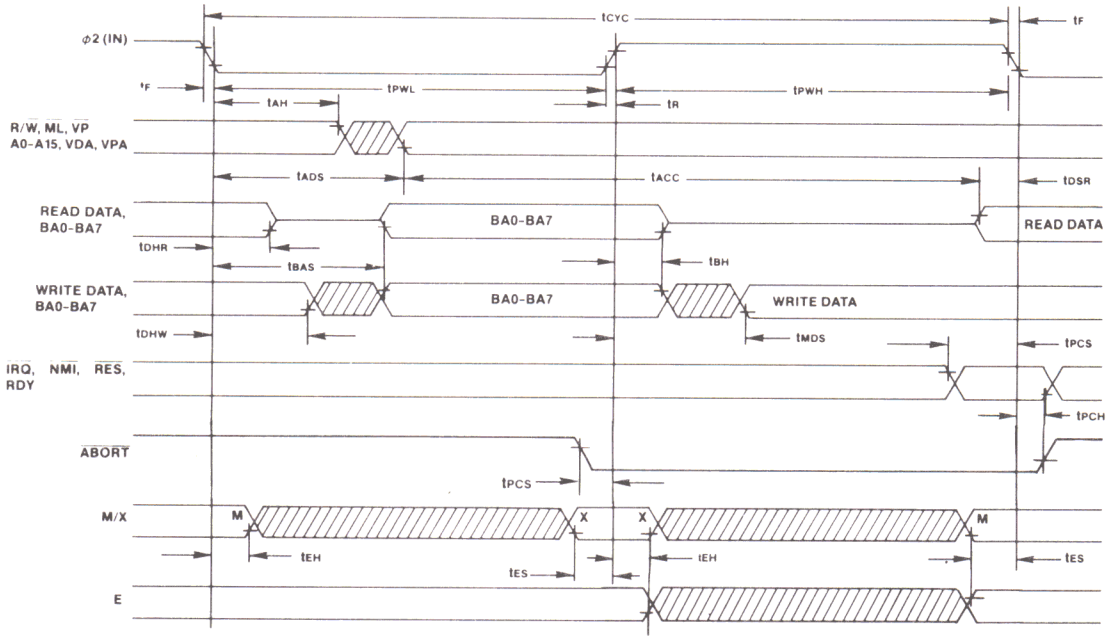
Pin	Description
A0-A15	Address Bus
ABORT	Abort Input
BE	Bus Enable
φ2 (IN)	Phase 2 In Clock
φ1 (OUT)	Phase 1 Out Clock
φ2 (OUT)	Phase 2 Out Clock
D0-D7	Data Bus (G65SC802)
D0/BA0-D7/BA7	Data Bus, Multiplexed (G65SC816)
E	Emulation Select
IRQ	Interrupt Request
ML	Memory Lock
M/X	Mode Select (P_M or Px)

Pin	Description
NC	No Connection
NMI	Non-Maskable Interrupt
RDY	Ready
RES	Reset
R/W	Read/Write
SO	Set Overflow
SYNC	Synchronize
VDA	Valid Data Address
VP	Vector Pull
VPA	Valid Program Address
V_{DD}	Positive Power Supply (+5 Volts)
V_{SS}	Internal Logic Ground

AC Characteristics (W65C816): V_{DD} = 5.0V ±5%, V_{SS} = 0V, T_A = 0°C to +70°C

Parameter	Symbol	2 MHz		4 MHz		6 MHz		8 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
Cycle Time	t_{CYC}	500	DC	250	DC	167	DC	125	DC	nS
Clock Pulse Width Low	t_{PWL}	0.240	10	0.120	10	0.080	10	0.060	10	µS
Clock Pulse Width High	t_{PWH}	240	∞	120	∞	80	∞	60	∞	nS
Fall Time, Rise Time	t_F, t_R	—	10	—	10	—	5	—	5	nS
A0-A15 Hold Time	t_{AH}	10	—	10	—	10	—	10	—	nS
A0-A15 Setup Time	t_{ADS}	—	100	—	75	—	60	—	40	nS
BA0-BA7 Hold Time	t_{BH}	10	—	10	—	10	—	10	—	nS
BA0-BA7 Setup Time	t_{BAS}	—	100	—	90	—	65	—	45	nS
Access Time	t_{ACC}	365	—	130	—	87	—	70	—	nS
Read Data Hold Time	t_{DHR}	10	—	10	—	10	—	10	—	nS
Read Data Setup Time	t_{DSR}	40	—	30	—	20	—	15	—	nS
Write Data Delay Time	t_{MDS}	—	100	—	70	—	60	—	40	nS
Write Data Hold Time	t_{DHW}	10	—	10	—	10	—	10	—	nS
Processor Control Setup Time	t_{PCS}	40	—	30	—	20	—	15	—	nS
Processor Control Hold Time	t_{PCH}	10	—	10	—	10	—	10	—	nS
E,MX Output Hold Time	t_{EH}	10	—	10	—	5	—	5	—	nS
E,MX Output Setup Time	t_{ES}	50	—	50	—	25	—	15	—	nS
Capacitive Load (Address, Data, and R/W)	C_{EXT}	—	100	—	100	—	35	—	35	pF
BE to High Impedance State	t_{BHZ}	—	30	—	30	—	30	—	30	nS
BE to Valid Data	t_{BVD}	—	30	—	30	—	30	—	30	nS

Timing Diagram (W65C816)



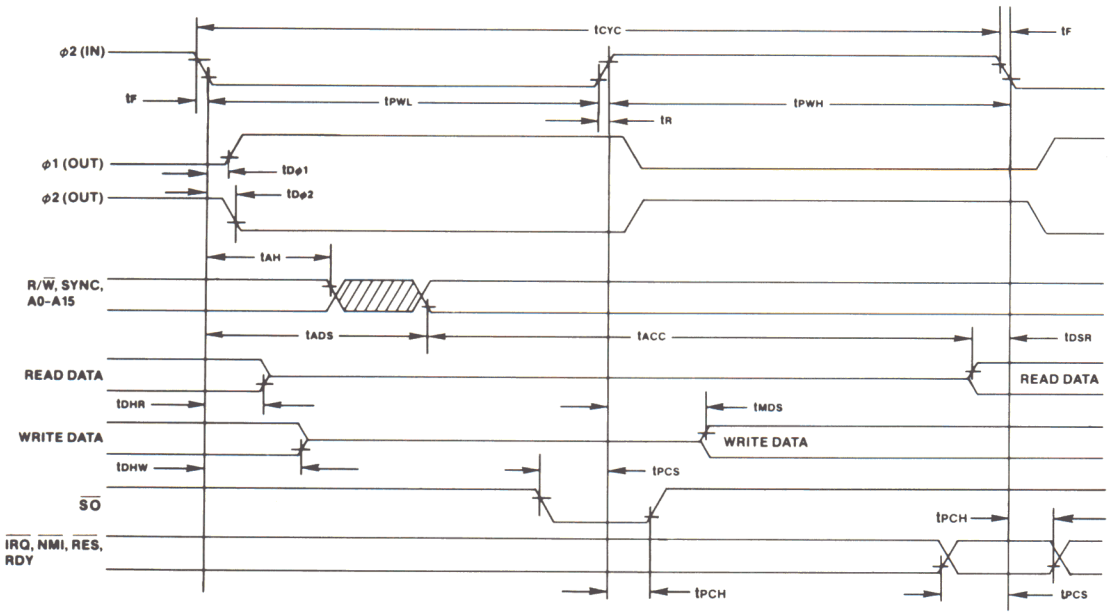
Timing Notes:

1. Voltage levels are V_L < 0.4V, V_H > 2.4V
2. Timing measurement points are 0.8V and 2.0V

AC Characteristics (W65C802): V_{DD} = 5.0V ±5%, V_{SS} = 0V, T_A = 0° C to +70° C

		2 MHz		4 MHz		6 MHz		8 MHz		
Parameter	**Symbol**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Unit**
Cycle Time	t_{CYC}	500	DC	250	DC	167	DC	125	DC	nS
Clock Pulse Width Low	t_{PWL}	0.240	10	0.120	10	0.080	10	0.060	10	µS
Clock Pulse Width High	t_{PWH}	240	∞	120	∞	80	∞	60	∞	nS
Fall Time, Rise Time	t_F, t_R	—	10	—	10	—	5	—	5	nS
Delay Time, ϕ2 (IN) to ϕ1 (OUT)	$t_{D\phi1}$	—	20	—	20	—	20	—	20	nS
Delay Time, ϕ2 (IN) to ϕ2 (OUT)	$t_{D\phi2}$	—	40	—	40	—	40	—	40	nS
Address Hold Time	t_{AH}	10	—	10	—	10	—	10	—	nS
Address Setup Time	t_{ADS}	—	100	—	75	—	60	—	40	nS
Access Time	t_{ACC}	365	—	130	—	87	—	70	—	nS
Read Data Hold Time	t_{DHR}	10	—	10	—	10	—	10	—	nS
Read Data Setup Time	t_{DSR}	40	—	30	—	20	—	15	—	nS
Write Data Delay Time	t_{MDS}	—	100	—	70	—	60	—	40	nS
Write Data Hold Time	t_{DHW}	10	—	10	—	10	—	10	—	nS
Processor Control Setup Time	t_{PCS}	40	—	30	—	20	—	15	—	nS
Processor Control Hold Time	t_{PCH}	10	—	10	—	10	—	10	—	nS
Capacitive Load (Address, Data, and R/W)	C_{EXT}	—	100	—	100	—	35	—	35	pF

Timing Diagram (W65C802)



Timing Notes:

1. Voltage levels are $V_L < 0.4V$, $V_H > 2.4V$
2. Timing measurement points are 0.8V and 2.0V

Functional Description

The W65C802 offers the design engineer the opportunity to utilize both existing software programs and hardware configurations, while also achieving the added advantages of increased register lengths and faster execution times. The W65C802's "ease of use" design and implementation features provide the designer with increased flexibility and reduced implementation costs. In the Emulation mode, the W65C802 not only offers software compatibility, but is also hardware (pin-to-pin) compatible with 6502 designs . . . plus it provides the advantages of 16-bit internal operation in 6502-compatible applications. The W65C802 is an excellent direct replacement microprocessor for 6502 designs.

The W65C816 provides the design engineer with upward mobility and software compatibility in applications where a 16-bit system configuration is desired. The W65C816's 16-bit hardware configuration, coupled with current software allows a wide selection of system applications. In the Emulation mode, the W65C816 offers many advantages, including full software compatibility with 6502 coding. In addition, the W65C816's powerful instruction set and addressing modes make it an excellent choice for new 16-bit designs.

Internal organization of the W65C802 and W65C816 can be divided into two parts: 1) The Register Section, and 2) The Control Section. Instructions (or opcodes) obtained from program memory are executed by implementing a series of data transfers within the Register Section. Signals that cause data transfers to be executed are generated within the Control Section. Both the W65C802 and the W65C816 have a 16-bit internal architecture with an 8-bit external data bus.

Instruction Register and Decode

An opcode enters the processor on the Data Bus, and is latched into the Instruction Register during the instruction fetch cycle. This instruction is then decoded, along with timing and interrupt signals, to generate the various Instruction Register control signals.

Timing Control Unit (TCU)

The Timing Control Unit keeps track of each instruction cycle as it is executed. The TCU is set to zero each time an instruction fetch is executed, and is advanced at the beginning of each cycle for as many cycles as is required to complete the instruction. Each data transfer between registers depends upon decoding the contents of both the Instruction Register and the Timing Control Unit.

Arithmetic and Logic Unit (ALU)

All arithmetic and logic operations take place within the 16-bit ALU. In addition to data operations, the ALU also calculates the effective address for relative and indexed addressing modes. The result of a data operation is stored in either memory or an internal register. Carry, Negative, Overflow and Zero flags may be updated following the ALU data operation.

Internal Registers (Refer to Programming Model)

Accumulators (A, B, C)

The Accumulator is a general purpose register which stores one of the operands, or the result of most arithmetic and logical operations. In the Native mode (E=0), when the Accumulator Select Bit (M) equals zero, the Accumulator is established as 16 bits wide (A + B = C). When the Accumulator Select Bit (M) equals one, the Accumulator is 8 bits wide (A). In this case, the upper 8 bits (B) may be used for temporary storage in conjunction with the Exchange Accumulator (XBA) instruction.

Data Bank Register (DBR)

During modes of operation, the 8-bit Data Bank Register holds the default bank address for memory transfers. The 24-bit address is composed of the 16-bit instruction effective address and the 8-bit Data Bank address. The register value is multiplexed with the data value and is present on the Data/Address lines during the first half of a data transfer memory cycle for the W65C816. The Data Bank Register is initialized to zero during Reset.

Direct (D)

The 16-bit Direct Register provides an address offset for all instructions using direct addressing. The effective bank zero address is formed by adding the 8-bit instruction operand address to the Direct Register. The Direct Register is initialized to zero during Reset.

Index (X and Y)

There are two Index Registers (X and Y) which may be used as general purpose registers or to provide an index value for calculation of the effective address. When executing an instruction with indexed addressing, the microprocessor fetches the opcode and the base address, and then modifies the address by adding the Index Register contents to the address prior to performing the desired operation. Pre-indexing or post-indexing of indirect addresses may be selected. In the Native mode (E=0), both Index Registers are 16 bits wide (providing the Index Select Bit (X) equals zero). If the Index Select Bit (X) equals one, both registers will be 8 bits wide, and the high byte is forced to zero.

Processor Status (P)

The 8-bit Processor Status Register contains status flags and mode select bits. The Carry (C), Negative (N), Overflow (V), and Zero (Z) status flags serve to report the status of most ALU operations. These status flags are tested by use of Conditional Branch instructions. The Decimal (D), IRQ Disable (I), Memory/Accumulator (M), and Index (X) bits are used as mode select flags. These flags are set by the program to change microprocessor operations.

The Emulation (E) select and the Break (B) flags are accessible only through the Processor Status Register. The Emulation mode select flag is selected by the Exchange Carry and Emulation Bits (XCE) instruction. Table 1, W65C802 and W65C816 Mode Comparison, illustrates the features of the Native (E=0) and Emulation (E=1) modes. The M and X flags are always equal to one in the Emulation mode. When an interrupt occurs during the Emulation mode, the Break flag is written to stack memory as bit 4 of the Processor Status Register.

Program Bank Register (PBR)

The 8-bit Program Bank Register holds the bank address for all instruction fetches. The 24-bit address consists of the 16-bit instruction effective address and the 8-bit Program Bank address. The register value is multiplexed with the data value and presented on the Data/Address lines during the first half of a program memory read cycle. The Program Bank Register is initialized to zero during Reset. The PHK instruction pushes the PBR register onto the Stack.

Program Counter (PC)

The 16-bit Program Counter Register provides the addresses which are used to step the microprocessor through sequential program instructions. The register is incremented each time an instruction or operand is fetched from program memory.

Stack Pointer (S)

The Stack Pointer is a 16-bit register which is used to indicate the next available location in the stack memory area. It serves as the effective address in stack addressing modes as well as subroutine and interrupt processing. The Stack Pointer allows simple implementation of nested subroutines and multiple-level interrupts. During the Emulation mode, the Stack Pointer high-order byte (SH) is always equal to one. The bank address for all stack operations is Bank zero.

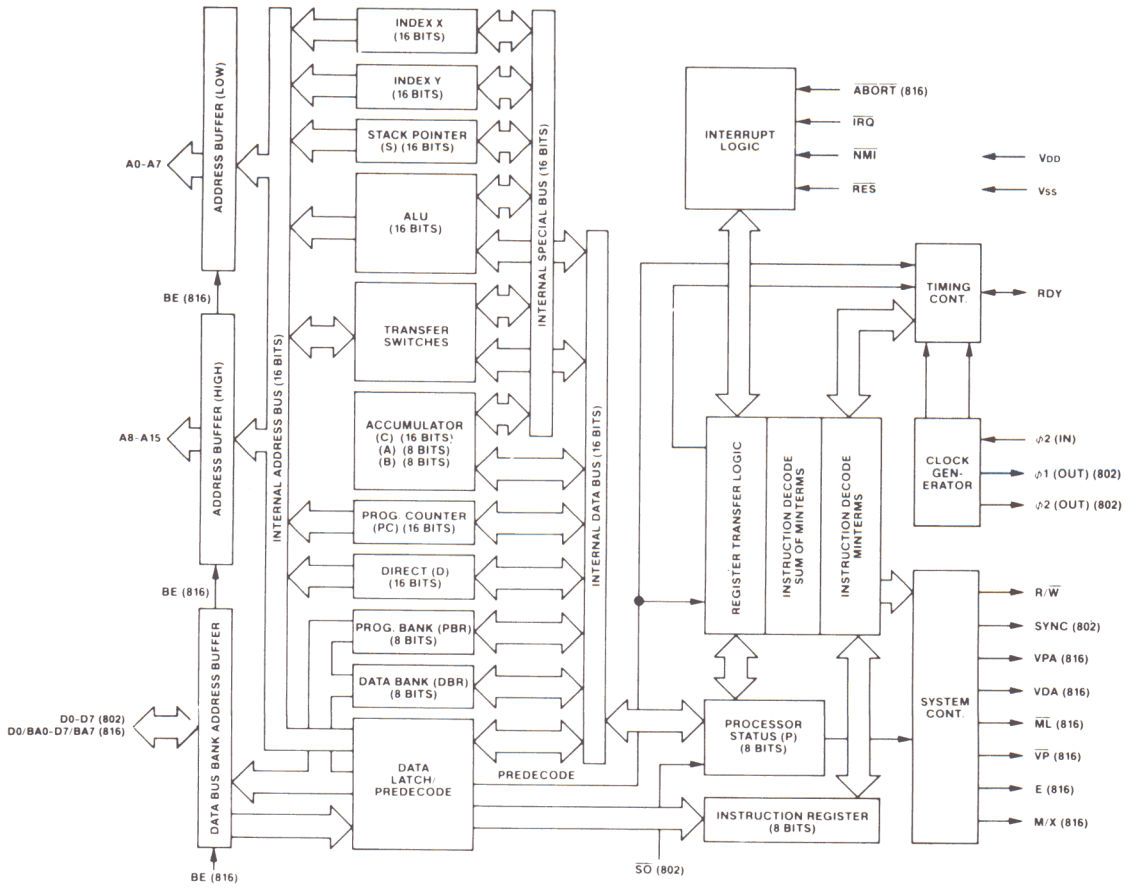


Figure 1. Block Diagram — Internal Architecture

Signal Description

The following Signal Description applies to both the W65C802 and the W65C816 except as otherwise noted.

Abort (ABORT)—W65C816

The Abort input is used to abort instructions (usually due to an Address Bus condition). A negative transition will inhibit modification of any internal register during the current instruction. Upon completion of this instruction, an interrupt sequence is initiated. The location of the aborted opcode is stored as the return address in stack memory. The Abort vector address is 00FFF8,9 (Emulation mode) or 00FFE8,9 (Native mode). Note that ABORT is a pulse-sensitive signal; i.e., an abort will occur whenever there is a negative pulse (or level) on the ABORT pin during a φ2 clock.

Address Bus (A0-A15)

These sixteen output lines form the Address Bus for memory and I/O exchange on the Data Bus. When using the W65C816, the address lines may be set to the high impedance state by the Bus Enable (BE) signal.

Bus Enable (BE)—W65C816

The Bus Enable input signal allows external control of the Address and Data Buffers, as well as the R/W signal. With Bus Enable high, the R/W and Address Buffers are active. The Data/Address Buffers are active during the first half of every cycle and the second half of a write cycle. When BE is low, these buffers are disabled. Bus Enable is an asynchronous signal.

Data Bus (D0-D7)—W65C802

The eight Data Bus lines provide an 8-bit bidirectional Data Bus for use during data exchanges between the microprocessor and external memory or peripherals. Two memory cycles are required for the transfer of 16-bit values.

Data/Address Bus (D0/BA0-D7/BA7)—W65C816

These eight lines multiplex address bits BA0-BA7 with the data value. The

address is present during the first half of a memory cycle, and the data value is read or written during the second half of the memory cycle. Two memory cycles are required to transfer 16-bit values. These lines may be set to the high impedance state by the Bus Enable (BE) signal.

Emulation Status (E)—W65C816

The Emulation Status output reflects the state of the Emulation (E) mode flag in the Processor Status (P) Register. This signal may be thought of as an opcode extension and used for memory and system management.

Interrupt Request ($\overline{IRQ}$)

The Interrupt Request input signal is used to request that an interrupt sequence be initiated. When the IRQ Disable (I) flag is cleared, a low input logic level initiates an interrupt sequence after the current instruction is completed. The Wait for Interrupt (WAI) instruction may be executed to ensure the interrupt will be recognized immediately. The Interrupt Request vector address is 00FFFE,F (Emulation mode) or 00FFEE,F (Native mode). Since $\overline{IRQ}$ is a level-sensitive input, an interrupt will occur if the interrupt source was not cleared since the last interrupt. Also, no interrupt will occur if the interrupt source is cleared prior to interrupt recognition.

Memory Lock ($\overline{ML}$)—W65C816

The Memory Lock output may be used to ensure the integrity of Read-Modify-Write instructions in a multiprocessor system. Memory Lock indicates the need to defer arbitration of the next bus cycle. Memory Lock is low during the last three or five cycles of ASL, DEC, INC, LSR, ROL, ROR, TRB, and TSB memory referencing instructions, depending on the state of the M flag.

Memory/Index Select Status (M/X)—W65C816

This multiplexed output reflects the state of the Accumulator (M) and Index (X) select flags (bits 5 and 4 of the Processor Status (P) Register. Flag M is valid during the Phase 2 clock negative transition and Flag X is valid during the Phase 2 clock positive transition. These bits may be thought of as opcode extensions and may be used for memory and system management.

Non-Maskable Interrupt ($\overline{NMI}$)

A negative transition on the NMI input initiates an interrupt sequence. A high-to-low transition initiates an interrupt sequence after the current instruction is completed. The Wait for Interrupt (WAI) instruction may be executed to ensure that the interrupt will be recognized immediately. The Non-Maskable Interrupt vector address is 00FFFA,B (Emulation mode) or 00FFEA,B (Native mode). Since NMI is an edge-sensitive input, an interrupt will occur if there is a negative transition while servicing a previous interrupt. Also, no interrupt will occur if NMI remains low.

Phase 1 Out (ϕ1 (OUT))—W65C802

This inverted clock output signal provides timing for external read and write operations. Executing the Stop (STP) instruction holds this clock in the low state.

Phase 2 In (ϕ2 (IN))

This is the system clock input to the microprocessor internal clock generator (equivalent to ϕ0 (IN) on the 6502). During the low power Standby Mode, ϕ2 (IN) should be held in the high state to preserve the contents of internal registers.

Phase 2 Out (ϕ2 (OUT))—W65C802

This clock output signal provides timing for external read and write operations. Addresses are valid (after the Address Setup Time (T_{ADS})) following the negative transition of Phase 2 Out. Executing the Stop (STP) instruction holds Phase 2 Out in the High state.

Read/Write (R/$\overline{W}$)

When the R/$\overline{W}$ output signal is in the high state, the microprocessor is reading data from memory or I/O. When in the low state, the Data Bus contains valid data from the microprocessor which is to be stored at the addressed memory location. When using the W65C816, the R/$\overline{W}$ signal may be set to the high impedance state by Bus Enable (BE).

Ready (RDY)

This bidirectional signal indicates that a Wait for Interrupt (WAI) instruction has been executed allowing the user to halt operation of the microprocessor. A low input logic level will halt the microprocessor in its current state (note that when in the Emulation mode, the W65C802 stops only during a read cycle). Returning RDY to the active high state allows the microprocessor to continue following the next Phase 2 In Clock negative transition. The RDY signal is internally pulled low following the execution of a Wait for Interrupt (WAI) instruction, and then returned to the high state when a $\overline{RES}$, $\overline{ABORT}$, $\overline{NMI}$, or $\overline{IRQ}$ external interrupt is provided. This feature may be used to eliminate interrupt latency by placing the WAI instruction at the beginning of the $\overline{IRQ}$ servicing routine. If the $\overline{IRQ}$ Disable flag has been set, the next instruction will be executed when the $\overline{IRQ}$ occurs. The processor will not stop after a WAI instruction if RDY has been forced to a high state. The Stop (STP) instruction has no effect on RDY.

Reset ($\overline{RES}$)

The Reset input is used to initialize the microprocessor and start program execution. The Reset input buffer has hysteresis such that a simple R-C timing circuit may be used with the internal pullup device. The RES signal must be held low for at least two clock cycles after V_{DD} reaches operating voltage. Ready (RDY) has no effect while RES is being held low. During this Reset conditioning period, the following processor initialization takes place:

Registers

D	=	0000								SH	= 01
DBR	=	00								XH	= 00
PBR	=	00								YH	= 00
		N	V	M	X	D	I	Z	C/E		
P	=	*	*	1	1	0	1	*	*/1	* = Not Initialized	

STP and WAI instructions are cleared.

Signals

E	= 1		VDA	= 0
M/X	= 1		$\overline{VP}$	= 1
R/$\overline{W}$	= 1		VPA	= 0
SYNC	= 0			

When Reset is brought high, an interrupt sequence is initiated:

- R/$\overline{W}$ remains in the high state during the stack address cycles.
- The Reset vector address is 00FFFC,D.

Set Overflow ($\overline{SO}$)—W65C802

A negative transition on this input sets the Overflow (V) flag, bit 6 of the Processor Status (P) Register.

Synchronize (SYNC)—W65C802

The SYNC output is provided to identify those cycles during which the microprocessor is fetching an opcode. The SYNC signal is high during an opcode fetch cycle, and when combined with Ready (RDY), can be used for single instruction execution.

Valid Data Address (VDA) and Valid Program Address (VPA)—W65C816

These two output signals indicate valid memory addresses when high (logic 1), and must be used for memory or I/O address qualification.

VDA	VPA	
0	0	Internal Operation—Address and Data Bus available. The Address Bus may be invalid.
0	1	Valid program address—may be used for program cache control.
1	0	Valid data address—may be used for data cache control.
1	1	Opcode fetch—may be used for program cache control and single step control.

V_{DD} and V_{SS}

V_{DD} is the positive supply voltage and V_{SS} is system logic ground. Pin 21 of the two V_{SS} pins on the W65C802 should be used for system ground.

Vector Pull ($\overline{VP}$)—W65C816

The Vector Pull output indicates that a vector location is being addressed during an interrupt sequence. $\overline{VP}$ is low during the last two interrupt sequence cycles, during which time the processor reads the interrupt vector. The $\overline{VP}$ signal may be used to select and prioritize interrupts from several sources by modifying the vector addresses.

Table 1. W65C816 Compatibility Issues

	W65C816/802	W65C02	NMOS 6502
1. S (Stack)	Always page 1 (E = 1), 8 bits 16 bits when (E = 0).	Always page 1, 8 bits	Always page 1, 8 bits
2. X (X Index Register)	Indexed page zero always in page 0 (E = 1), Cross page (E = 0).	Always page 0	Always page 0
3. Y (Y Index Register)	Indexed page zero always in page 0 (E = 1), Cross page (E = 0).	Always page 0	Always page 0
4. A (Accumulator)	8 bits (M = 1), 16 bits (M = 0)	8 bits	8 bits
5. P (Flag Registor)	N, V, and Z flags valid in decimal mode. D = 0 after reset or interrupt.	N, V, and Z flags valid in decimal mode. D = 0 after reset and interrupt.	N, V, and Z flags invalid in decimal mode. D = unknown after reset. D not modified after interrupt.
6. Timing A. ABS, X ASL, LSR, ROL, ROR With No Page Crossing	7 cycles	6 cycles	7 cycles
B. Jump Indirect Operand = XXFF	5 cycles	6 cycles	5 cycles and invalid page crossing
C. Branch Across Page	4 cycles (E = 1) 3 cycles (E = 0)	4 cycles	4 cycles
D. Decimal Mode	No additional cycle	Add 1 cycle	No additional cycle
7. BRK Vector	00FFFE,F (E = 1) BRK bit = 0 on stack if $\overline{IRQ}$, $\overline{NMI}$, $\overline{ABORT}$. 00FFE6, 7 (E = 0) X = X on Stack always.	FFFE,F BRK bit = 0 on stack if $\overline{IRQ}$, $\overline{NMI}$.	FFFE,F BRK bit = 0 on stack if $\overline{IRQ}$, $\overline{NMI}$.
8. Interrupt or Break Bank Address	PBR not pushed (E = 1) RTI PBR not pulled (E = 1) PBR pushed (E = 0) RTI PBR pulled (E = 0)	Not available	Not available
9. Memory Lock ($\overline{ML}$)	$\overline{ML}$ = 0 during Read, Modify and Write cycles.	$\overline{ML}$ = 0 during Modify and Write.	Not available
10. Indexed Across Page Boundary (d),y; a,x; a,y	Extra read of invalid address. (Note 1)	Extra read of last instruction fetch.	Extra read of invalid address.
11. RDY Pulled During Write Cycle.	Ignored (E = 1) for W65C802 only. Processor stops (E = 0).	Processor stops	Ignored
12. WAI and STP Instructions.	Available	Available	Not available
13. Unused OP Codes	One reserved OP Code specified as WDM will be used in future systems. The W65C816 performs a no-operation.	No operation	Unknown and some "hang up" processor.
14. Bank Address Handling	PBR = 00 after reset or interrupts.	Not available	Not available
15. R/$\overline{W}$ During Read-Modify-Write Instructions	E = 1, R/$\overline{W}$ = 0 during Modify and Write cycles. E = 0, R/$\overline{W}$ = 0 only during Write cycle.	R/$\overline{W}$ = 0 only during Write cycle	R/$\overline{W}$ = 0 during Modify and Write cycles.
16. Pin 7	W65C802 = SYNC. W65C816 = VPA	SYNC	SYNC
17. COP Instruction Signatures 00-7F user defined Signatures 80-FF reserved	Available	Not available	Not available

Note 1. See Caveat section for additional information.

W65C802 and W65C816 Microprocessor Addressing Modes

The W65C816 is capable of directly addressing 16 MBytes of memory. This address space has special significance within certain addressing modes, as follows:

Reset and Interrupt Vectors

The Reset and Interrupt vectors use the majority of the fixed addresses between 00FFE0 and 00FFFF.

Stack

The Stack may use memory from 000000 to 00FFFF. The effective address of Stack and Stack Relative addressing modes will always be within this range.

Direct

The Direct addressing modes are usually used to store memory registers and pointers. The effective address generated by Direct, Direct,X and Direct,Y addressing modes is always in Bank 0 (000000-00FFFF).

Program Address Space

The Program Bank register is not affected by the Relative, Relative Long, Absolute, Absolute Indirect, and Absolute Indexed Indirect addressing modes or by incrementing the Program Counter from FFFF. The only instructions that affect the Program Bank register are: RTI, RTL, JML, JSL, and JMP Absolute Long. Program code may exceed 64K bytes although code segments may not span bank boundaries.

Data Address Space

The data address space is contiguous throughout the 16 MByte address space. Words, arrays, records, or any data structures may span 64 KByte bank boundaries with no compromise in code efficiency. The following addressing modes generate 24-bit effective addresses:

- Direct Indexed Indirect (d,x)
- Direct Indirect Indexed (d),y
- Direct Indirect (d)
- Direct Indirect Long [d]
- Direct Indirect Long Indexed [d],y
- Absolute a
- Absolute a,x
- Absolute a,y
- Absolute Long al
- Absolute Long Indexed al,x
- Stack Relative Indirect Indexed (d,s),y

The following addressing mode descriptions provide additional detail as to how effective addresses are calculated.

Twenty-four addressing modes are available for use with the W65C802 and W65C816 microprocessors. The "long" addressing modes may be used with the W65C802; however, the high byte of the address is not available to the hardware. Detailed descriptions of the 24 addressing modes are as follows:

1. Immediate Addressing—#

The operand is the second byte (second and third bytes when in the 16-bit mode) of the instruction.

2. Absolute—a

With Absolute addressing the second and third bytes of the instruction form the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the operand address.

Instruction:	opcode	addrl	addrh
Operand Address:	DBR	addrh	addrl

3. Absolute Long—al

The second, third, and fourth byte of the instruction form the 24-bit effective address.

Instruction:	opcode	addrl	addrh	baddr
Operand Address:	baddr	addrh	addrl	

4. Direct—d

The second byte of the instruction is added to the Direct Register (D) to form the effective address. An additional cycle is required when the Direct Register is not page aligned (DL not equal 0). The Bank register is always 0.

Instruction:	opcode	offset
		Direct Register
	+	offset
Operand Address:	00	effective address

5. Accumulator—A

This form of addressing always uses a single byte instruction. The operand is the Accumulator.

6. Implied—i

Implied addressing uses a single byte instruction. The operand is implicitly defined by the instruction.

7. Direct Indirect Indexed—(d),y

This address mode is often referred to as Indirect,Y. The second byte of the instruction is added to the Direct Register (D). The 16-bit contents of this memory location is then combined with the Data Bank register to form a 24-bit base address. The Y Index Register is added to the base address to form the effective address.

Instruction:	opcode	offset
		Direct Register
	+	offset
	00	direct address
then:		
	00	(direct address)
	+ DBR	
		base address
	+	Y Reg
Operand Address:		effective address

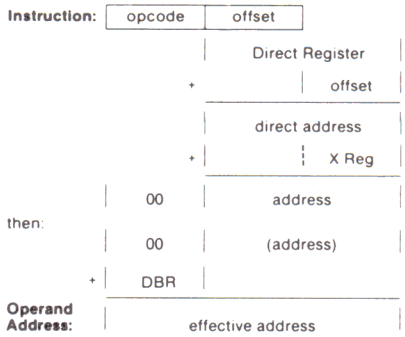
8. Direct Indirect Long Indexed—[d],y

With this addressing mode, the 24-bit base address is pointed to by the sum of the second byte of the instruction and the Direct Register. The effective address is this 24-bit base address plus the Y Index Register.

Instruction:	opcode	offset
		Direct Register
	+	offset
	00	direct address
then:		
		(direct address)
	+	Y Reg
Operand Address:		effective address

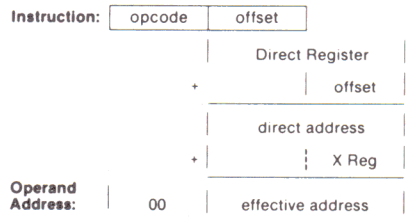
9. Direct Indexed Indirect—(d,x)

This address mode is often referred to as Indirect,X. The second byte of the instruction is added to the sum of the Direct Register and the X Index Register. The result points to the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



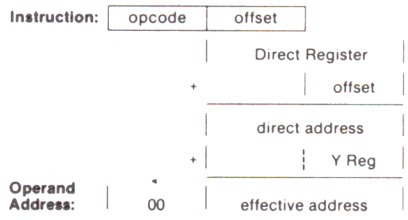
10. Direct Indexed With X—d,x

The second byte of the instruction is added to the sum of the Direct Register and the X Index Register to form the 16-bit effective address. The operand is always in Bank 0.



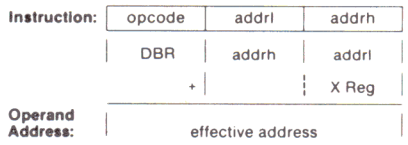
11. Direct Indexed With Y—d,y

The second byte of the instruction is added to the sum of the Direct Register and the Y Index Register to form the 16-bit effective address. The operand is always in Bank 0.



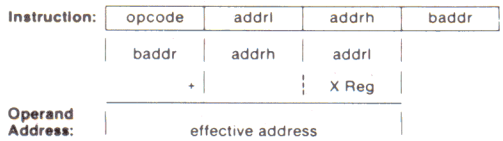
12. Absolute Indexed With X—a,x

The second and third bytes of the instruction are added to the X Index Register to form the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



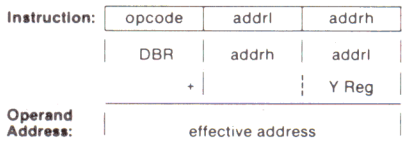
13. Absolute Long Indexed With X—al,x

The second, third and fourth bytes of the instruction form a 24-bit base address. The effective address is the sum of this 24-bit address and the X Index Register.



14. Absolute Indexed With Y—a,y

The second and third bytes of the instruction are added to the Y Index Register to form the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



15. Program Counter Relative—r

This address mode, referred to as Relative Addressing, is used only with the Branch instructions. If the condition being tested is met, the second byte of the instruction is added to the Program Counter, which has been updated to point to the opcode of the next instruction. The offset is a signed 8-bit quantity in the range from -128 to 127. The Program Bank Register is not affected.

16. Program Counter Relative Long—rl

This address mode, referred to as Relative Long Addressing, is used only with the Unconditional Branch Long instruction (BRL) and the Push Effective Relative instruction (PER). The second and third bytes of the instruction are added to the Program Counter, which has been updated to point to the opcode of the next instruction. With the branch instruction, the Program Counter is loaded with the result. With the Push Effective Relative instruction, the result is stored on the stack. The offset is a signed 16-bit quantity in the range from -32768 to 32767. The Program Bank Register is not affected.

17. Absolute Indirect—(a)

The second and third bytes of the instruction form an address to a pointer in Bank 0. The Program Counter is loaded with the first and second bytes at this pointer. With the Jump Long (JML) instruction, the Program Bank Register is loaded with the third byte of the pointer.

New PC = (indirect address)

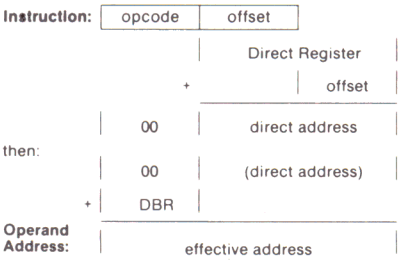
with JML:

New PC = (indirect address)

New PBR = (indirect address +2)

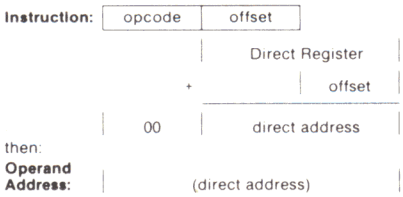
18. Direct Indirect—(d)

The second byte of the instruction is added to the Direct Register to form a pointer to the low-order 16 bits of the effective address. The Data Bank Register contains the high-order 8 bits of the effective address.



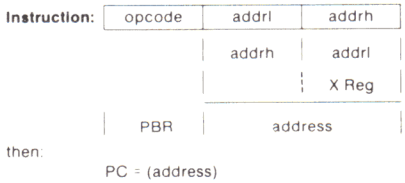
19. Direct Indirect Long—[d]

The second byte of the instruction is added to the Direct Register to form a pointer to the 24-bit effective address.



20. Absolute Indexed Indirect—(a,x)

The second and third bytes of the instruction are added to the X Index Register to form a 16-bit pointer in Bank 0. The contents of this pointer are loaded in the Program Counter. The Program Bank Register is not changed.

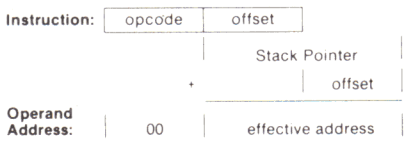


21. Stack—s

Stack addressing refers to all instructions that push or pull data from the stack, such as Push, Pull, Jump to Subroutine, Return from Subroutine, Interrupts, and Return from Interrupt. The bank address is always 0. Interrupt Vectors are always fetched from Bank 0.

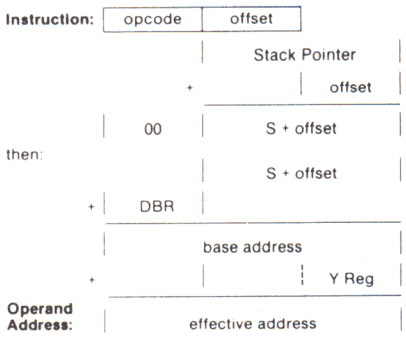
22. Stack Relative—d,s

The low-order 16 bits of the effective address is formed from the sum of the second byte of the instruction and the Stack Pointer. The high-order 8 bits of the effective address is always zero. The relative offset is an unsigned 8-bit quantity in the range of 0 to 255.



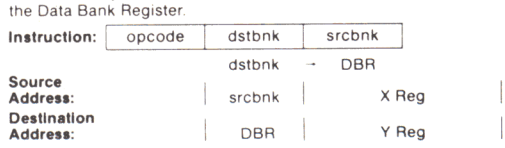
23. Stack Relative Indirect Indexed—(d,s),y

The second byte of the instruction is added to the Stack Pointer to form a pointer to the low-order 16-bit base address in Bank 0. The Data Bank Register contains the high-order 8 bits of the base address. The effective address is the sum of the 24-bit base address and the Y Index Register.



24. Block Source Bank, Destination Bank—xyc

This addressing mode is used by the Block Move instructions. The second byte of the instruction contains the high-order 8 bits of the destination address. The Y index Register contains the low-order 16 bits of the destination address. The third byte of the instruction contains the high-order 8 bits of the source address. The X Index Register contains the low-order 16 bits of the source address. The C Accumulator contains one less than the number of bytes to move. The second byte of the block move instructions is also loaded into the Data Bank Register.



Increment (MVN) or decrement (MVP) X and Y.
Decrement C (if greater than zero), then PC+3 → PC.

Table 2. W65C802 and W65C816 Instruction Set—Alphabetical Sequence

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift One Bit Left, Memory or Accumulator
BCC	Branch on Carry Clear ($P_C = 0$)
BCS	Branch on Carry Set ($P_C = 1$)
BEQ	Branch if Equal ($P_Z = 1$)
BIT	Bit Test
BMI	Branch if Result Minus ($P_N = 1$)
BNE	Branch if Not Equal ($P_Z = 0$)
BPL	Branch if Result Plus ($P_N = 0$)
BRA	Branch Always
BRK	Force Break
BRL	Branch Always Long
BVC	Branch on Overflow Clear ($P_V = 0$)
BVS	Branch on Overflow Set ($P_V = 1$)
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
COP	Coprocessor
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory or Accumulator by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive OR" Memory with Accumulator
INC	Increment Memory or Accumulator by One
INX	Increment Index X by One
INY	Increment Index Y by One
JML	Jump Long
JMP	Jump to New Location
JSL	Jump Subroutine Long
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift One Bit Right (Memory or Accumulator)
MVN	Block Move Negative
MVP	Block Move Positive
NOP	No Operation
ORA	"OR" Memory with Accumulator
PEA	Push Effective Absolute Address on Stack (or Push Immediate Data on Stack)
PEI	Push Effective Indirect Address on Stack (or Push Direct Data on Stack)
PER	Push Effective Program Counter Relative Address on Stack
PHA	Push Accumulator on Stack
PHB	Push Data Bank Register on Stack
PHD	Push Direct Register on Stack
PHK	Push Program Bank Register on Stack
PHP	Push Processor Status on Stack
PHX	Push Index X on Stack
PHY	Push Index Y on Stack
PLA	Pull Accumulator from Stack
PLB	Pull Data Bank Register from Stack
PLD	Pull Direct Register from Stack
PLP	Pull Processor Status from Stack
PLX	Pull Index X from Stack
PLY	Pull Index Y form Stack
REP	Reset Status Bits
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTL	Return from Subroutine Long
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
SEP	Set Processor Status Bite
STA	Store Accumulator in Memory
STP	Stop the Clock
STX	Store Index X in Memory
STY	Store Index Y in Memory
STZ	Store Zero in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TCD	Transfer C Accumulator to Direct Register
TCS	Transfer C Accumulator to Stack Pointer Register
TDC	Transfer Direct Register to C Accumulator
TRB	Test and Reset Bit
TSB	Test and Set Bit
TSC	Transfer Stack Pointer Register to C Accumulator
TSX	Transfer Stack Pointer Register to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer Register
TXY	Transfer Index X to Index Y
TYA	Transfer Index Y to Accumulator
TYX	Transfer Index Y to Index X
WAI	Wait for Interrupt
WDM	Reserved for Future Use
XBA	Exchange B and A Accumulator
XCE	Exchange Carry and Emulation Bits

For alternate mnemonics, see Table 7.

Table 3. Vector Locations

E 1			E 0		
OOFFFE,F	$\overline{IRQ}$/BRK	Hardware/Software	OOFFEE,F	$\overline{IRQ}$	Hardware
OOFFFC,D	RESET	Hardware	OOFFEC,D	(Reserved)	
OOFFFA,B	$\overline{NMI}$	Hardware	OOFFEA,B	$\overline{NMI}$	Hardware
OOFFF8,9	$\overline{ABORT}$	Hardware	OOFFE8,9	$\overline{ABORT}$	Hardware
OOFFF6,7	(Reserved)		OOFFE6,7	BRK	Software
OOFFF4,5	COP	Software	OOFFE4,5	COP	Software

The VP output is low during the two cycles used for vector location access.
When an interrupt is executed, D = 0 and I = 1 in Status Register P

Table 4. Opcode Matrix

MSD	LSD 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	MSD
0	BRK s 2 8	ORA (d,x) 2 6	COP s 2 ★ 8	ORA d,s 2 ★ 4	TSB d 2 ● 5	ORA d 2 3	ASL d 2 5	ORA [d] 2 ★ 6	PHP s 1 3	ORA # 2 2	ASL A 1 2	PHD s 1 ★ 4	TSB a 3 ● 6	ORA a 3 4	ASL a 3 6	ORA al 4 ★ 5	0
1	BPL r 2 2	ORA (d),y 2 5	ORA (d) 2 ● 5	ORA (d,s),y 2 ★ 7	TRB d 2 ● 5	ORA d,x 2 4	ASL d,x 2 6	ORA [d],y 2 ★ 6	CLC i 1 2	ORA a,y 3 4	INC A 1 ● 2	TCS i 1 ★ 2	TRB a 3 ● 6	ORA a,x 3 4	ASL a,x 3 7	ORA al,x 4 ★ 5	1
2	JSR a 3 6	AND (d,x) 2 6	JSL al 4 ★ 8	AND d,s 2 ★ 4	BIT d 2 3	AND d 2 3	ROL d 2 5	AND [d] 2 ★ 6	PLP s 1 4	AND # 2 2	ROL A 1 2	PLD s 1 ★ 5	BIT a 3 4	AND a 3 4	ROL a 3 6	AND al 4 ★ 5	2
3	BMI r 2 2	AND (d),y 2 5	AND (d) 2 ● 5	AND (d,s),y 2 ★ 7	BIT d,x 2 ● 4	AND d,x 2 4	ROL d,x 2 6	AND [d],y 2 ★ 6	SEC i 1 2	AND a,y 3 4	DEC A 1 ● 2	TSC i 1 ★ 2	BIT a,x 3 ● 4	AND a,x 3 4	ROL a,x 3 7	AND al,x 4 ★ 5	3
4	RTI s 1 7	EOR (d,x) 2 6	WDM 2 ★ 2	EOR d,s 2 ★ 4	MVP xyc 3 ★ 7	EOR d 2 3	LSR d 2 5	EOR [d] 2 ★ 6	PHA s 1 3	EOR # 2 2	LSR A 1 2	PHK s 1 ★ 3	JMP a 3 3	EOR a 3 4	LSR a 3 6	EOR al 4 ★ 5	4
5	BVC r 2 2	EOR (d),y 2 5	EOR (d) 2 ● 5	EOR (d,s),y 2 ★ 7	MVN xyc 3 ★ 7	EOR d,x 2 4	LSR d,x 2 6	EOR [d],y 2 ★ 6	CLI i 1 2	EOR a,y 3 4	PHY s 1 ● 3	TCD i 1 ★ 2	JMP al 4 ★ 4	EOR a,x 3 4	LSR a,x 3 7	EOR al,x 4 ★ 5	5
6	RTS s 1 6	ADC (d,x) 2 6	PER s 3 ★ 6	ADC d,s 2 ★ 4	STZ d 2 ● 3	ADC d 2 3	ROR d 2 5	ADC [d] 2 ★ 6	PLA s 1 4	ADC # 2 2	ROR A 1 2	RTL s 1 ★ 6	JMP (a) 3 5	ADC a 3 4	ROR a 3 6	ADC al 4 ★ 5	6
7	BVS r 2 2	ADC (d),y 2 5	ADC (d) 2 ● 5	ADC (d,s),y 2 ★ 7	STZ d,x 2 ● 4	ADC d,x 2 4	ROR d,x 2 6	ADC [d],y 2 ★ 6	SEI i 1 2	ADC a,y 3 4	PLY s 1 ● 4	TDC i 1 ★ 2	JMP (a,x) 3 ● 6	ADC a,x 3 4	ROR a,x 3 7	ADC al,x 4 ★ 5	7
8	BRA r 2 ● 2	STA (d,x) 2 6	BRL rl 3 ★ 3	STA d,s 2 ★ 4	STY d 2 3	STA d 2 3	STX d 2 3	STA [d] 2 ★ 6	DEY i 1 2	BIT # 2 ● 2	TXA i 1 2	PHB s 1 ★ 3	STY a 3 4	STA a 3 4	STX a 3 4	STA al 4 ★ 5	8
9	BCC r 2 2	STA (d),y 2 6	STA (d) 2 ● 5	STA (d,s),y 2 ★ 7	STY d,x 2 4	STA d,x 2 4	STX d,y 2 4	STA [d],y 2 ★ 6	TYA i 1 2	STA a,y 3 5	TXS i 1 2	TXY i 1 ★ 2	STZ a 3 ● 4	STA a,x 3 5	STZ a,x 3 ● 5	STA al,x 4 ★ 5	9
A	LDY # 2 2	LDA (d,x) 2 6	LDX # 2 2	LDA d,s 2 ★ 4	LDY d 2 3	LDA d 2 3	LDX d 2 3	LDA [d] 2 ★ 6	TAY i 1 2	LDA # 2 2	TAX i 1 2	PLB s 1 ★ 4	LDY a 3 4	LDA a 3 4	LDX a 3 4	LDA al 4 ★ 5	A
B	BCS r 2 2	LDA (d),y 2 5	LDA (d) 2 ● 5	LDA (d,s),y 2 ★ 7	LDY d,x 2 4	LDA d,x 2 4	LDX d,y 2 4	LDA [d],y 2 ★ 6	CLV i 1 2	LDA a,y 3 4	TSX i 1 2	TYX i 1 ★ 2	LDY a,x 3 4	LDA a,x 3 4	LDX a,y 3 4	LDA al,x 4 ★ 5	B
C	CPY # 2 2	CMP (d,x) 2 6	REP # 2 ★ 3	CMP d,s 2 ★ 4	CPY d 2 3	CMP d 2 3	DEC d 2 5	CMP [d] 2 ★ 6	INY i 1 2	CMP # 2 2	DEX i 1 2	WAI i 1 ● 3	CPY a 3 4	CMP a 3 4	DEC a 3 6	CMP al 4 ★ 5	C
D	BNE r 2 2	CMP (d),y 2 5	CMP (d) 2 ● 5	CMP (d,s),y 2 ★ 7	PEI s 2 ★ 6	CMP d,x 2 4	DEC d,x 2 6	CMP [d],y 2 ★ 6	CLD i 1 2	CMP a,y 3 4	PHX s 1 ● 3	STP i 1 ● 3	JML (a) 3 ★ 6	CMP a,x 3 4	DEC a,x 3 7	CMP al,x 4 ★ 5	D
E	CPX # 2 2	SBC (d,x) 2 6	SEP # 2 ★ 3	SBC d,s 2 ★ 4	CPX d 2 3	SBC d 2 3	INC d 2 5	SBC [d] 2 ★ 6	INX i 1 2	SBC # 2 2	NOP i 1 2	XBA i 1 ★ 3	CPX a 3 4	SBC a 3 4	INC a 3 6	SBC al 4 ★ 5	E
F	BEQ r 2 2	SBC (d),y 2 5	SBC (d) 2 ● 5	SBC (d,s),y 2 ★ 7	PEA s 3 ★ 5	SBC d,x 2 4	INC d,x 2 6	SBC [d],y 2 ★ 6	SED i 1 2	SBC a,y 3 4	PLX s 1 ● 4	XCE i 1 ★ 2	JSR (a,x) 3 ★ 6	SBC a,x 3 4	INC a,x 3 7	SBC al,x 4 ★ 5	F
MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	MSD

symbol	addressing mode	symbol	addressing mode
#	immediate	[d]	direct indirect long
A	accumulator	[d],y	direct indirect long indexed
r	program counter relative	a	absolute
rl	program counter relative long	a,x	absolute indexed (with x)
i	implied	a,y	absolute indexed (with y)
s	stack	al	absolute long
d	direct	al,x	absolute long indexed
d,x	direct indexed (with x)	d,s	stack relative
d,y	direct indexed (with y)	(d,s),y	stack relative indirect indexed
(d)	direct indirect	(a)	absolute indirect
(d,x)	direct indexed indirect	(a,x)	absolute indexed indirect
(d),y	direct indirect indexed	xyc	block move

Op Code Matrix Legend

INSTRUCTION MNEMONIC

BASE NO. BYTES

★ - New W65C816/802 Opcodes
● - New W65C02 Opcodes
Blank - NMOS 6502 Opcodes

ADDRESSING MODE

BASE NO. CYCLES

Table 5. Operation, Operation Codes, and Status Register

MNE-MONIC	OPERATION	#	a	al	d	A	i	(d),y	[d],y	(d,x)	d,x	d,y	a,x	al,x	a,y	r	rl	(a)	(d)	[d]	(a,x)	s	d,s	(d,s),y	xyc	PROCESSOR STATUS CODE 7 6 5 4 3 2 1 0	MNE-MONIC
																										N V M X D I Z C	E = 0
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	N V 1 B D I Z C	E = 1
ADC	A + M + C → A	69	6D	6F	65			71	77	61	75		7D	7F	79				72	67			63	73		N V . . . . Z C	ADC
AND	A∧M → A	29	2D	2F	25			31	37	21	35		3D	3F	39				32	27			23	33		N . . . . . Z .	AND
ASL	C - [15/7 0] ← 0		0E		06	0A					16		1E													N . . . . . Z C	ASL
BCC	BRANCH IF C = 0															90										. . . .	BCC
BCS	BRANCH IF C = 1															B0										. . . .	BCS
BEQ	BRANCH IF Z = 1															F0										. . . .	BEQ
BIT	AAM (NOTE 1)	89	2C		24						34		3C													M7 M6 . . . . Z .	BIT
BMI	BRANCH IF N = 1															30										. . . .	BMI
BNE	BRANCH IF Z = 0															D0										. . . .	BNE
BPL	BRANCH IF N = 0															10										. . . .	BPL
BRA	BRANCH ALWAYS															80										. . . .	● BRA
BRK	BREAK (NOTE 2)																					00				. . . . 0 1 . .	BRK
BRL	BRANCH LONG ALWAYS																82									. . . .	★ BRL
BVC	BRANCH IF V = 0															50										. . . .	BVC
BVS	BRANCH IF V = 1															70										. . . .	BVS
CLC	0 → C						18																			 0	CLC
CLD	0 → D						D8																			. . . . 0 . . .	CLD
CLI	0 → I						58																			. . . . . 0 . .	CLI
CLV	0 → V						B8																			. . . .	CLV
CMP	A-M	C9	CD	CF	C5			D1	D7	C1	D5		DD	DF	D9				D2	C7			C3	D3		N . . . . . Z C	CMP
COP	CO-PROCESSOR																					02				. . . . 0 1 . .	★ COP
CPX	X-M	E0	EC		E4																					N . . . . . Z C	CPX
CPY	Y-M	C0	CC		C4																					N . . . . . Z C	CPY
DEC	DECREMENT		CE		C6	3A					D6		DE													N . . . . . Z .	DEC
DEX	X - 1 → X						CA																			N . . . . . Z .	DEX
DEY	Y - 1 → Y						88																			N . . . . . Z .	DEY
EOR	A∨M → A	49	4D	4F	45	1A		51	57	41	55		5D	5F	59				52	47			43	53		N . . . . . Z .	EOR
INC	INCREMENTS		EE		E6						F6		FE													N . . . . . Z .	INC
INX	X + 1 → X						E8																			N . . . . . Z .	INX
INY	Y + 1 → Y						C8																			N . . . . . Z .	INY
JML	JUMP LONG TO NEW LOC.																	DC									★ JML
JMP	JUMP TO NEW LOC.		4C	5C														6C			7C						JMP
JSL	JUMP LONG TO SUB.			22																							JSL
JSR	JUMP TO SUB.		20																		FC						JSR
LDA	M → A	A9	AD	AF	A5			B1	B7	A1	B5		BD	BF	B9				B2	A7			A3	B3		N . . . . . Z .	LDA
LDX	M → X	A2	AE		A6							B6			BE											N . . . . . Z .	LDX
LDY	M → Y	A0	AC		A4						B4		BC													N . . . . . Z .	LDY
LSR	0 → [15/7 0] → C		4E		46	4A					56		5E													0 . . . . . Z C	LSR
MVN	M → M NEGATIVE																								54		★ MVN
MVP	M → M POSITIVE																								44		★ MVP
NOP	NO OPERATION						EA																				NOP
ORA	AVM - A	09	0D	0F	05			11	17	01	15		1D	1F	19				12	07			03	13		N . . . . . Z .	ORA
PEA	Mpc + 1, Mpc + 2 - Ms - 1, Ms S - 2 - S																					F4					★ PEA
PEI	M(d), M(d + 1) - Ms - 1, Ms S - 2 - S																					D4					★ PEI
PER	Mpc + rl, Mpc + rl + 1 - Ms - 1, Ms S - 2 - S																					62					★ PER
PHA	A - Ms, S - 1 - S																					48					PHA
PHB	DBR → Ms, S - 1 - S																					8B					PHB
PHD	D → Ms, Ms - 1, S - 2 - S																					0B					★ PHD
PHK	PBR → Ms, S - 1 - S																					4B					★ PHK
PHP	P → Ms, S - 1 - S																					08					PHP
PHX	X → Ms, S - 1 - S																					DA					● PHX
PHY	Y → Ms, S - 1 - S																					5A					● PHY
PLA	S + 1 → S, Ms → A																					68				N . . . . . Z .	PLA
PLB	S + 1 → S, Ms → DBR																					AB				N . . . . . Z .	★ PLB
PLD	S + 2 → S, Ms - 1, Ms → D																					2B				N . . . . . Z .	★ PLD
PLP	S + 1 → S, Ms → P																					28				N V M X D I Z C	PLP
PLX	S + 1 → S, Ms → X																					FA				N . . . . . Z .	PLX
PLY	S + 1 → S, Ms → Y																					7A				N . . . . . Z .	PLY
REP	MAP → P	C2																								N V M X D I Z C	★ REP
ROL	[15/7 0] ← C		2E		26	2A					36		3E													N . . . . . Z C	ROL
ROR	C → [15/7 0]		6E		66	6A					76		7E													N . . . . . Z C	ROR
RTI	RTRN FROM INT.																					40				N V M X D I Z C	RTI
RTL	RTRN FROM SUB. LONG																					6B					★ RTL
RTS	RTRN SUBROUTINE																					60					RTS
SBC	A - M - C̄ → A	E9	ED	EF	E5			F1	F7	E1	F5		FD	FF	F9				F2	E7			E3	F3		N V . . . . Z C	SBC
SEC	1 → C						38																			 1	SEC
SED	1 → D						F8																			. . . . 1 . . .	SED
SEI	1 → I						78																			. . . . . 1 . .	SEI
SEP	MVP → P	E2																								N V M X D I Z C	★ SEP
STA	A → M		8D	8F	85			91	97	81	95		9D	9F	99				92	87			83	93			STA
STP	STOP (1 → φ2)						DB																				● STP
STX	X → M		8E		86							96															STX
STY	Y → M		8C		84						94																STY
STZ	00 → M		9C		64						74		9E														● STZ
TAX	A → X						AA																			N . . . . . Z .	TAX
TAY	A → Y						A8																			N . . . . . Z .	TAY
TCD	C → D						5B																			N . . . . . Z .	★ TCD
TCS	C → S						1B																				★ TCS
TDC	D → C						7B																			N . . . . . Z .	★ TDC
TRB			1C		14																					. . . . . Z .	● TRB
TSB	AVM → M		0C		04																					. . . . . Z .	● TSB
TSC	S → C						3B																			N . . . . . Z .	★ TSC
TSX	S → X						BA																			N . . . . . Z .	TSX
TXA	X → A						8A																			N . . . . . Z .	TXA
TXS	X → S						9A																				TXS
TXY	X → Y						9B																			N . . . . . Z .	★ TXY
TYA	Y → A						98																			N . . . . . Z .	TYA
TYX	Y → X						BB																			N . . . . . Z .	★ TYX
WAI	0 → RDY						CB																				● WAI
WDM	NO OPERATION (RESERVED)						42																				★ WDM
XBA	B ⟷ A						EB																			N . . . . . Z .	★ XBA
XCE	C ⟷ E						FB																			. . . . E	★ XCE

Notes:
1. Bit immediate N and V flags not affected. When M = 0, M_{15} - N and M_{14} - V.
2. Break Bit (B) in Status register indicates hardware or software break.

3. ★ = New W65C816/802 Instructions
● = New W65C02 Instructions
Blank = NMOS 6502

- Add
- Subtract
∧ AND
V OR
⊻ Exclusive OR

Table 6. Detailed Instruction Operation

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
1	Immediate **#**		1	1	1	1	1	PBR,PC	Op Code	1
	(LDY,CPY,CPX,LDX,ORA,		2	1	1	0	1	PBR,PC+1	IDL	1
	AND,EOR,ADC,BIT,LDA,	(1)(8)	2a	1	1	0	1	PBR,PC+2	IDH	1
	CMP,SBC,REP,SEP)									
	(14 Op Codes)									
	(2 and 3 bytes)									
	(2 and 3 cycles)									
2a	Absolute **a**		1	1	1	1	1	PBR,PC	Op Code	1
	(BIT,STY,STZ,LDY,		2	1	1	0	1	PBR,PC+1	AAL	1
	CPY,CPX,STX,LDX,		3	1	1	0	1	PBR,PC+2	AAH	1
	ORA,AND,EOR,ADC,		4	1	1	1	0	DBR,AA	Data Low	1 0
	STA,LDA,CMP,SBC)	(1)	4a	1	1	1	0	DBR,AA+1	Data High	1 0
	(18 Op Codes)									
	(3 bytes)									
	(4 and 5 cycles)									
2b	Absolute (R-M-W) **a**		1	1	1	1	1	PBR,PC	Op Code	1
			2	1	1	0	1	PBR,PC+1	AAL	1
	(ASL,ROL,LSR,ROR		3	1	1	0	1	PBR,PC+2	AAH	1
	DEC,INC,TSB,TRB)		4	1	0	1	0	DBR,AA	Data Low	1
	(6 Op Codes)	(1)	4a	1	0	1	0	DBR,AA+1	Data High	1
	(3 bytes)	(3)	5	1	0	0	0	DBR,AA+1	IO	1
	(6 and 8 cycles)	(1)	6a	1	0	1	0	DBR,AA+1	Data High	0
			6	1	0	1	0	DBR,AA	Data Low	0
2c	Absolute (JUMP) **a**		1	1	1	1	1	PBR,PC	Op Code	1
	(JMP)(4C)		2	1	1	0	1	PBR,PC+1	NEW PCL	1
	(1 Op Code)		3	1	1	0	1	PBR,PC+2	NEW PCH	1
	(3 bytes)		1	1	1	1	1	PBR,NEW PC	Op Code	1
	(3 cycles)									
2d	Absolute (Jump to		1	1	1	1	1	PBR,PC	Op Code	1
	subroutine) **a**		2	1	1	0	1	PBR,PC+1	NEW PCL	1
	(JSR)		3	1	1	0	1	PBR,PC+2	NEW PCH	1
	(1 Op Code)		4	1	1	0	0	PBR,PC+2	IO	1
	(3 bytes)		5	1	1	1	0	0,S	PCH	0
	(6 cycles)		6	1	1	1	0	0,S-1	PCL	0
	(different order from N6502)		1	1	1	1	1	PBR,NEW PC	Next Op Code	1
★3a	Absolute Long **al**		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC,		2	1	1	0	1	PBR,PC+1	AAL	1
	STA,LDA,CMP,SBC)		3	1	1	0	1	PBR,PC+2	AAH	1
	(8 Op Codes)		4	1	1	0	1	PBR,PC+3	AAB	1
	(4 bytes)		5	1	1	1	0	AAB,AA	Data Low	1 0
	(5 and 6 cycles)	(1)	5a	1	1	1	0	AAB,AA+1	Data High	1 0
★3b	Absolute Long (JUMP) **al**		1	1	1	1	1	PBR,PC	Op Code	1
	(JMP)		2	1	1	0	1	PBR,PC+1	NEW PCL	1
	(1 Op Code)		3	1	1	0	1	PBR,PC+2	NEW PCH	1
	(4 bytes)		4	1	1	0	1	PBR,PC+3	NEW BR	1
	(4 cycles)		1	1	1	1	1	NEW PBR,PC	Op Code	1
★3c	Absolute Long (Jump to		1	1	1	1	1	PBR,PC	Op Code	1
	Subroutine Long) **al**		2	1	1	0	1	PBR,PC+1	NEW PCL	1
	(JSL)		3	1	1	0	1	PBR,PC+2	NEW PCH	1
	(1 Op Code)		4	1	1	1	0	0,S	PBR	0
	(4 bytes)		5	1	1	0	0	0,S	IO	1
	(7 cycles)		6	1	1	0	1	PBR,PC+3	NEW PBR	1
			7	1	1	1	0	0,S-1	PCH	0
			8	1	1	1	0	0,S-2	PCL	0
			1	1	1	1	1	NEW PBR,PC	Next Op Code	1
4a	Direct **d**		1	1	1	1	1	PBR,PC	Op Code	1
	(BIT,STZ,STY,LDY,		2	1	1	0	1	PBR,PC+1	DO	1
	CPY,CPX,STX,LDX,	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	ORA,AND,EOR,ADC,		3	1	1	1	0	0,D+DO	Data Low	1/0
	STA,LDA,CMP,SBC)	(1)	3a	1	1	1	0	0,D+DO+1	Data High	1/0
	(18 Op Codes)									
	(2 bytes)									
	(3,4 and 5 cycles)									
4b	Direct (R-M-W) **d**		1	1	1	1	1	PBR,PC	Op Code	1
	(ASL,ROL,LSR,ROR		2	1	1	0	1	PBR,PC+1	DO	1
	DEC,INC,TSB,TRB)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(6 Op Codes)		3	1	0	1	0	0,D+DO	Data Low	1
	(2 bytes)	(1)	3a	1	0	1	0	0,D+DO+1	Data High	1
	(5,6,7 and 8 cycles)	(3)	4	1	0	0	0	0,D+DO+1	IO	1
		(1)	5a	1	0	1	0	0,D+DO+1	Data High	0
			5	1	0	1	0	0,D+DO	Data Low	0
5	Accumulator **A**		1	1	1	1	1	PBR,PC	Op Code	1
	(ASL,INC,ROL,DEC,LSR,ROR)		2	1	1	0	0	PBR,PC+1	IO	1
	(6 Op Codes)									
	(1 byte)									
	(2 cycles)									
6a	Implied **i**		1	1	1	1	1	PBR,PC	Op Code	1
	(DEY,INY,INX,DEX,NOP,		2	1	1	0	0	PBR,PC+1	IO	1
	XCE,TYA,TAY,TXA,TXS,									
	TAX,TSX,TCS,TSC,TCD,									
	TDC,TXY,TYX,CLC,SEC,									
	CLI,SEI,CLV,CLD,SED)									
	(25 Op Codes)									
	(1 byte)									
	(2 cycles)									
★6b	Implied **i**		1	1	1	1	1	PBR,PC	Op Code	1
	(XBA)		2	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		3	1	1	0	0	PBR,PC+1	IO	1
	(1 byte)									
	(3 cycles)									
								RDY		
● 6c	Wait For Interrupt									
	(WAI)		1	1	1	1	1	1 PBR,PC	Op Code	1
	(1 Op Code)	(9)	2	1	1	0	0	1 PBR,PC+1	IO	1
	(1 byte)		3	1	1	0	0	0 PBR,PC+1	IO	1
	(3 cycles)	IRQ,NMI	1	1	1	1	1	1 PBR,PC+1	IRQ(BRK)	1
● 6d	Stop-The-Clock									
	(STP)		1	1	1	1	1	1 PBR,PC	Op Code	1
	(1 Op Code)		2	1	1	0	0	1 PBR,PC+1	IO	1
	(1 byte)	RES: 1	3	1	1	0	0	1 PBR,PC+1	IO	1
	(3 cycles)	RES: 0	1c	1	1	0	0	1 PBR,PC+1	RES(BRK)	1
		RES: 0	1b	1	1	0	0	1 PBR,PC+1	RES(BRK)	1
		RES: 1	1a	1	1	0	0	1 PBR,PC+1	RES(BRK)	1
	See 21a Stack (Hardware interrupt)		1	1	1	1	1	1 PBR,PC+1	BEGIN	1

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
7	Direct Indirect Indexed **(d),y**		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC,		2	1	1	0	1	PBR,PC+1	DO	1
	STA,LDA,CMP,SBC)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		3	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(5,6,7 and 8 cycles)	(4)	4a	1	1	0	0	DBR,AAH,AAL+Y	YL IO	1
			5	1	1	1	0	DBR,AA+Y	Data Low	1/0
		(1)	5a	1	1	1	0	DBR,AA+Y+1	Data High	1/0
8	Direct Indirect		1	1	1	1	1	PBR,PC	Op Code	1
	Indexed Long **[d],y**		2	1	1	0	1	PBR,PC+1	DO	1
	(ORA,AND,EOR,ADC,	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	STA,LDA,CMP,SBC)		3	1	1	1	0	0,D+DO	AAL	1
	(8 Op Codes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(2 bytes)		5	1	1	1	0	0,D+DO+2	AAB	1
	(6,7 and 8 cycles)		6	1	1	1	0	AAB,AA+Y	Data Low	1/0
		(1)	6a	1	1	1	0	AAB,AA+Y+1	Data High	1/0
9	Direct Indexed Indirect **(d,x)**		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC,		2	1	1	0	1	PBR,PC+1	DO	1
	STA,LDA,CMP,SBC)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		3	1	1	0	0	PBR,PC+1	IO	1
	(2 bytes)		4	1	1	1	0	0,D+DO+X	AAL	1
	(6,7 and 8 cycles)		5	1	1	1	0	0,D+DO+X+1	AAH	1
			6	1	1	1	0	DBR,AA	Data Low	1/0
		(1)	6a	1	1	1	0	DBR,AA+1	Data High	1/0
10a	Direct **X d,x**		1	1	1	1	1	PBR,PC	Op Code	1
	(BIT,STZ,STY,LDY,		2	1	1	0	1	PBR,PC+1	DO	1
	ORA,AND,EOR,ADC,	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	STA,LDA,CMP,SBC)		3	1	1	0	0	PBR,PC+1	IO	1
	(11 Op Codes)		4	1	1	1	0	0,D+DO+X	Data Low	1/0
	(2 bytes)	(1)	4a	1	1	1	0	0,D+DO+X+1	Data High	1/0
	(4,5 and 6 cycles)									
10b	Direct X(R-M-W) **d,x**		1	1	1	1	1	PBR,PC	Op Code	1
	(ASL,ROL,LSR,ROR,		2	1	1	0	1	PBR,PC+1	DO	1
	DEC,INC)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(6 Op Codes)		3	1	1	0	0	PBR,PC+1	IO	1
	(2 bytes)		4	1	0	1	0	0,D+DO+X	Data Low	1
	(6,7,8 and 9 cycles)	(1)	4a	1	0	1	0	0,D+DO+X+1	Data High	1
		(3)	5	1	0	0	0	0,D+DO+X+1	IO	1
		(1)	6a	1	0	1	0	0,D+DO+X+1	Data High	0
			6	1	0	1	0	0,D+DO+X	Data Low	0
11	Direct Y **d,y**		1	1	1	1	1	PBR,PC	Op Code	1
	(STX,LDX)		2	1	1	0	1	PBR,PC+1	DO	1
	(2 Op Codes)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(2 bytes)		3	1	1	0	0	PBR,PC+1	IO	1
	(4,5 and 6 cycles)		4	1	1	1	0	0,D+DO+Y	Data Low	1/0
		(1)	4a	1	1	1	0	0,D+DO+Y+1	Data High	1/0
12a	Absolute,X **a,x**		1	1	1	1	1	PBR,PC	Op Code	1
	(BIT,LDY,STZ,		2	1	1	0	1	PBR,PC+1	AAL	1
	ORA,AND,EOR,ADC,		3	1	1	0	1	PBR,PC+2	AAH	1
	STA,LDA,CMP,SBC)	(4)	3a	1	1	0	0	DBR,AAH,AAL+XL	IO	1
	(11 Op Codes)		4	1	1	1	0	DBR,AA+X	Data Low	1/0
	(3 bytes)	(1)	4a	1	1	1	0	DBR,AA+X+1	Data High	1/0
	(4,5 and 6 cycles)									
12b	Absolute,X(R-M-W) **a,x**		1	1	1	1	1	PBR,PC	Op Code	1
	(ASL,ROL,LSR,ROR,		2	1	1	0	1	PBR,PC+1	AAL	1
	DEC,INC)		3	1	1	0	1	PBR,PC+2	AAH	1
	(6 Op Codes)		4	1	1	0	0	DBR,AAH,AAL+XL	IO	1
	(3 bytes)		5	1	0	1	0	DBR,AA+X	Data Low	1
	(7 and 9 cycles)	(1)	5a	1	0	1	0	DBR,AA+X+1	Data High	1
		(3)	6	1	0	0	0	DBR,AA+X+1	IO	1
		(1)	7a	1	0	1	0	DBR,AA+X+1	Data High	0
			7	1	0	1	0	DBR,AA+X	Data Low	0
★13	Absolute Long,X **al,x**		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC,		2	1	1	0	1	PBR,PC+1	AAL	1
	STA,LDA,CMP,SBC)		3	1	1	0	1	PBR,PC+2	AAH	1
	(8 Op Codes)		4	1	1	0	1	PBR,PC+3	AAB	1
	(4 bytes)		5	1	1	1	0	AAB,AA+X	Data Low	1/0
	(5 and 6 cycles)	(1)	5a	1	1	1	0	AAB,AA+X+1	Data High	1/0
14	Absolute,Y **a,y**		1	1	1	1	1	PBR,PC	Op Code	1
	(LDX,ORA,AND,EOR,ADC,		2	1	1	0	1	PBR,PC+1	AAL	1
	STA,LDA,CMP,SBC)		3	1	1	0	1	PBR,PC+2	AAH	1
	(9 Op Codes)	(4)	3a	1	1	0	0	DBR,AAH,AAL+YL	IO	1
	(3 bytes)		4	1	1	1	0	DBR,AA+Y	Data Low	1/0
	(4,5 and 6 cycles)	(1)	4a	1	1	1	0	DBR,AA+Y+1	Data High	1/0
15	Relative **r**		1	1	1	1	1	PBR,PC	Op Code	1
	(BPL,BMI,BVC,BVS,BCC,		2	1	1	0	1	PBR,PC+1	Offset	1
	BCS,BNE,BEQ,BRA)	(5)	2a	1	1	0	0	PBR,PC+1	IO	1
	(9 Op Codes)	(6)	2b	1	1	0	0	PBR,PC+1	IO	1
	(2 bytes)		1	1	1	1	1	PBR,PC+Offset	Op Code	1
	(2,3 and 4 cycles)									
★16	Relative Long **rl**		1	1	1	1	1	PBR,PC	Op Code	1
	(BRL)		2	1	1	0	1	PBR,PC+1	Offset Low	1
	(1 Op Code)		3	1	1	0	1	PBR,PC+2	Offset High	1
	(3 bytes)		4	1	1	0	0	PBR,PC+2	IO	1
	(4 cycles)		1	1	1	1	1	PBR,PC+Offset	Op Code	1
17a	Absolute Indirect **(a)**		1	1	1	1	1	PBR,PC	Op Code	1
	(JMP)		2	1	1	0	1	PBR,PC+1	AAL	1
	(1 Op Code)		3	1	1	0	1	PBR,PC+2	AAH	1
	(3 bytes)		4	1	1	1	0	0,AA	NEW PCL	1
	(5 cycles)		5	1	1	1	0	0,AA+1	NEW PCH	1
			1	1	1	1	1	PBR,NEW PC	Op Code	1
★17b	Absolute Indirect **(a)**		1	1	1	1	1	PBR,PC	Op Code	1
			2	1	1	0	1	PBR,PC+1	AAL	1
	(JML)		3	1	1	0	1	PBR,PC+2	AAH	1
	(1 Op Code)		4	1	1	1	0	0,AA	NEW PCL	1
	(3 bytes)		5	1	1	1	0	0,AA+1	NEW PCH	1
	(6 cycles)		6	1	1	1	0	0,AA+2	NEW PBR	1
			1	1	1	1	1	NEW PBR,PC	Op Code	1
● 18	Direct Indirect **(d)**		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC,		2	1	1	0	1	PBR,PC+1	DO	1
	STA,LDA,CMP,SBC)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		3	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(5,6 and 7 cycles)		5	1	1	1	0	DBR,AA	Data Low	1/0
		(1)	5a	1	1	1	0	DBR,AA+1	Data Low	1/0

Table 6. Detailed Instruction Operation (continued)

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
★19	Direct Indirect Long [d]		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC		2	1	1	0	1	PBR,PC+1	DO	1
	STA,LDA,CMP,SBC)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		3	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(6,7 and 8 cycles)		5	1	1	1	0	0,D+DO+2	AAB	1
			6	1	1	1	0	AAB,AA	Data Low	1/0
		(1)	6a	1	1	1	0	AAB,AA+1	Data High	1/0
20a	Absolute Indexed Indirect (a,x)		1	1	1	1	1	PBR,PC	Op Code	1
	(JMP)		2	1	1	0	1	PBR,PC+1	AAL	1
	(1 Op Code)		3	1	1	0	1	PBR,PC+2	AAH	1
	(3 bytes)		4	1	1	0	0	PBR,PC+2	IO	1
	(6 cycles)		5	1	1	0	1	PBR,AA+X	NEW PCL	1
			6	1	1	0	1	PBR,AA+X+1	NEW PCH	1
			1	1	1	1	1	PBR,NEW PC	Op Code	1
★20b	Absolute Indexed Indirect		1	1	1	1	1	PBR,PC	Op Code	1
	(Jump to Subroutine Indexed		2	1	1	0	1	PBR,PC+1	AAL	1
	Indirect) (a,x)		3	1	1	1	0	0,S	PCH	0
	(JSR)		4	1	1	1	0	0,S-1	PCL	0
	(1 Op Code)		5	1	1	0	1	PBR,PC+2	AAH	1
	(3 bytes)		6	1	1	0	0	PBR,PC+2	IO	1
	(8 cycles)		7	1	1	0	1	PBR,AA+X	NEW PCL	1
			8	1	1	0	1	PBR,AA+X+1	NEW PCH	1
			1	1	1	1	1	PBR,NEW PC	Next Op Code	1
21a	Stack (Hardware		1	1	1	1	1	PBR,PC	IO	1
	Interrupts) s	(3)	2	1	1	0	0	PBR,PC	IO	1
	(IRQ,NMI,ABORT,RES)	(7)	3	1	1	1	0	0,S	PBR	0
	(4 hardware interrupts)		4	1	1	1	0	0,S-1	PCH	0
	(0 bytes)		5	1	1	1	0	0,S-2	PCL	0
	(7 and 8 cycles)		6	1	1	1	0	0,S-3	P	0
			7	0	1	1	0	0,VA	AAVL	1
			8	0	1	1	0	0,VA+1	AAVH	1
			1	1	1	1	1	0,AAV	Next Op Code	1
21b	Stack (Software		1	1	1	1	1	PBR,PC	Op Code	1
	Interrupts) s	(3)	2	1	1	0	1	PBR,PC+1	Signature	1
	(BRK,COP)	(7)	3	1	1	1	0	0,S	PBR	0
	(2 Op Cooes)	(10)	4	1	1	1	0	0,S-1	PCH	0
	(2 bytes)	(10)	5	1	1	1	0	0,S-2	PCL	0
	(7 and 8 cycles)	(10)	6	1	1	1	0	0,S-3 (COP Latches)	P	0
			7	0	1	1	0	0,VA	AAVL	1
			8	0	1	1	0	0,VA+1	AAVH	1
			1	1	1	1	1	0,AAV	Next Op Code	1
21c	Stack (Return from		1	1	1	1	1	PBR,PC	Op Code	1
	Interrupt) s		2	1	1	0	0	PBR,PC+1	IO	1
	(RTI)	(3)	3	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		4	1	1	1	0	0,S+1	P	1
	(1 byte)		5	1	1	1	0	0,S+2	PCL	1
	(6 and 7 cycles)		6	1	1	1	0	0,S+3	PCH	1
	(different order from N6502)	(7)	7	1	1	1	0	0,S+4	PBR	1
			1	1	1	1	1	PBR,PC	New Op Code	1
21d	Stack (Return from		1	1	1	1	1	PBR,PC	Op Code	1
	Subroutine) s		2	1	1	0	0	PBR,PC+1	IO	1
	(RTS)		3	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		4	1	1	1	0	0,S+1	PCL	1
	(1 byte)		5	1	1	1	0	0,S+2	PCH	1
	(6 cycles)		6	1	1	0	0	0,S+2	IO	1
			1	1	1	1	1	PBR,PC	Op Code	1
★21e	Stack (Return from		1	1	1	1	1	PBR,PC	Op Code	1
	Subroutine Long) s		2	1	1	0	0	PBR,PC+1	IO	1
	(RTL)		3	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		4	1	1	1	0	0,S+1	NEW PCL	1
	(1 byte)		5	1	1	1	0	0,S+2	NEW PCH	1
	(6 cycles)		6	1	1	1	0	0,S+3	NEW PBR	1
			1	1	1	1	1	NEW PBR,PC	Next Op Code	1
21f	Stack (Push) s		1	1	1	1	1	PBR,PC	Op Code	1
	(PHP,PHA,PHY,PHX,		2	1	1	0	0	PBR,PC+1	IO	1
	PHD,PHK,PHB)	(1)	3a	1	1	1	0	0,S	Register High	1
	(7 Op Codes)		3	1	1	1	0	0,S-1	Register Low	1
	(1 byte)									
	(3 and 4 cycles)									
21g	Stack (Pull) s		1	1	1	1	1	PBR,PC	Op Code	1
	(PLP,PLA,PLY,PLX,PLD,PLB)		2	1	1	0	0	PBR,PC+1	IO	1
	(Different than N6502)		3	1	1	0	0	PBR,PC+1	IO	1
	(6 Op Codes)		4	1	1	1	0	0,S+1	Register Low	1
	(1 byte)	(1)	4a	1	1	1	0	0,S+2	Register High	1
	(4 and 5 cycles)									
★21h	Stack (Push Effective		1	1	1	1	1	PBR,PC	Op Code	1
	Indirect Address) s		2	1	1	0	1	PBR,PC+1	DO	1
	(PEI)	(2)	2a	1	1	0	0	PBR,PC+1	IO	1
	(1 Op Code)		3	1	1	1	0	0,D+DO	AAL	1
	(2 bytes)		4	1	1	1	0	0,D+DO+1	AAH	1
	(6 and 7 cycles)		5	1	1	1	0	0,S	AAH	0
			6	1	1	1	0	0,S-1	AAL	0
★21i	Stack (Push Effective		1	1	1	1	1	PBR,PC	Op Code	1
	Absolute Address) s		2	1	1	0	1	PBR,PC+1	AAL	1
	(PEA)		3	1	1	0	1	PBR,PC+2	AAH	1
	(1 Op Code)		4	1	1	1	0	0,S	AAH	0
	(3 bytes)		5	1	1	1	0	0,S-1	AAL	0
	(5 cycles)									
★21j	Stack (Push Effective		1	1	1	1	1	PBR,PC	Op Code	1
	Program Counter Relative		2	1	1	0	1	PBR,PC+1	Offset Low	1
	Address) s		3	1	1	0	1	PBR,PC+2	Offset High	1
	(PER)		4	1	1	0	0	PBR,PC+2	IO	1
	(1 Op Code)		5	1	1	1	0	0,S	PCH+OFF+ CARRY	0
	(3 bytes)									
	(6 cycles)		6	1	1	1	0	0,S-1	PCL+OFFSET	0
★22	Stack Relative d,s		1	1	1	1	1	PBR,PC	Op Code	1
	(ORA,AND,EOR,ADC		2	1	1	0	1	PBR,PC+1	SO	1
	STA,LDA,CMP,SBC)		3	1	1	0	0	PBR,PC+1	IO	1
	(8 Op Codes)		4	1	1	1	0	0,S+SO	Data Low	1/0
	(2 bytes)	(1)	4a	1	1	1	0	0,S+SO+1	Data High	1/0
	(4 and 5 cycles)									

	ADDRESS MODE		CYCLE	VP	ML	VDA	VPA	ADDRESS BUS	DATA BUS	R/W
★23	Stack Relative Indirect		1	1	1	1	1	PBR,PC	Op Code	1
	Indexed (d,s),y		2	1	1	0	1	PBR,PC+1	SO	1
	(ORA,AND,EOR,ADC		3	1	1	0	0	PBR+PC+1	IO	1
	STA,LDA,CMP,SDC)		4	1	1	1	0	0,S+SO	AAL	1
	(8 Op Codes)		5	1	1	1	0	0,S+SO+1	AAH	1
	(2 bytes)		6	1	1	0	0	0,S+SO+1	IO	1
	(7 and 8 Cycles)		7	1	1	1	0	DBR,AA+Y	Data Low	1/0
		(1)	7a	1	1	1	0	DBR,AA+Y+1	Data High	1/0
★24a	Block Move Positive		1	1	1	1	1	PBR,PC	Op Code	1
	(forward) xyc		2	1	1	0	1	PBR,PC+1	DBA	1
	(MVP)		3	1	1	0	1	PBR,PC+2	SBA	1
	(1 Op Code)	N-2	4	1	1	1	0	SBA,X	Source Data	1
	(3 bytes)	Byte	5	1	1	1	0	DBA,Y	Dest Data	0
	(7 cycles)	C-2	6	1	1	0	0	DBA,Y	IO	1
	x Source Address		7	1	1	0	0	DBA,Y	IO	1
	y Destination		1	1	1	1	1	PBR,PC	Op Code	1
	c Number of Bytes to Move -1		2	1	1	0	1	PBR,PC+1	DBA	1
	x,y Decrement		3	1	1	0	1	PBR,PC+2	SBA	1
	MVP is used when the	N-1	4	1	1	1	0	SBA,X-1	Source Data	1
	destination start address	Byte	5	1	1	1	0	DBA,Y-1	Dest Data	0
	is higher (more positive)	C-1	6	1	1	0	0	DBA,Y-1	IO	1
	than the source start address		7	1	1	0	0	DBA,Y-1	IO	1
	FFFFFF		1	1	1	1	1	PBR,PC	Op Code	1
	Dest Start	N Byte	2	1	1	0	1	PBR,PC+1	DBA	1
	Source Start	Last	3	1	1	0	1	PBR,PC+2	SBA	1
	Dest End	C-0	4	1	1	1	0	SBA,X-2	Source Data	1
	Source End		5	1	1	1	0	DBA,Y-2	Dest Data	0
	000000		6	1	1	0	0	DBA,Y-2	IO	1
			7	1	1	0	0	DBA,Y-2	IO	1
			1	1	1	1	1	PBR,PC+3	Next Op Code	1
★24b	Block Move Negative		1	1	1	1	1	PBR,PC	Op Code	1
	(backward) xyc		2	1	1	0	1	PBR,PC+1	DBA	1
	(MVN)	N-2	3	1	1	0	1	PBR,PC+2	SBA	1
	(1 Op Code)	Byte	4	1	1	1	0	SBA,X	Source Data	1
	(3 bytes)	C-2	5	1	1	1	0	DBA,Y	Dest Data	0
	(7 cycles)		6	1	1	0	0	DBA,Y	IO	1
	x Source Address		7	1	1	0	0	DBA,Y	IO	1
	y Destination									
	c Number of Bytes to Move -1		1	1	1	1	1	PBR,PC	Op Code	1
	x,y Increment		2	1	1	0	1	PBR,PC+1	DBA	1
	FFFFFF	N-1	3	1	1	0	1	PBR,PC+2	SBA	1
	Source End	Byte	4	1	1	1	0	SBA,X+1	Source Data	1
	Dest End	C-1	5	1	1	1	0	DBA,Y+1	Dest Data	0
	Source Start		6	1	1	0	0	DBA,Y+1	IO	1
	Dest Start		7	1	1	0	0	DBA,Y+1	IO	1
			1	1	1	1	1	PBR,PC	Op Code	1
	000000	N Byte	2	1	1	0	1	PBR,PC+1	DBA	1
		C-0	3	1	1	0	1	PBR,PC+2	SBA	1
	MVN is used when the		4	1	1	1	0	SBA,X+2	Source Data	1
	destination start address		5	1	1	1	0	DBA,Y+2	Dest Data	0
	is lower (more negative)		6	1	1	0	0	DBA,Y+2	IO	1
	than the source start		7	1	1	0	0	DBA,Y+2	IO	1
	address		1	1	1	1	1	PBR,PC+3	Next Op Code	1

Notes

(1) Add 1 byte (for immediate only) for M=0 or X=0 (i.e. 16 bit data); add 1 cycle for M=0 or X=0

(2) Add 1 cycle for direct register low (DL) not equal 0

(3) Special case for aborting instruction. This is the last cycle which may be aborted or the Status, PBR or DBR registers will be updated.

(4) Add 1 cycle for indexing across page boundaries, or write, or X=0. When X=1 or in the emulation mode, this cycle contains invalid addresses.

(5) Add 1 cycle if branch is taken

(6) Add 1 cycle if branch is taken across page boundaries in 6502 emulation mode (E=1)

(7) Subtract 1 cycle for 6502 emulation mode (E=1)

(8) Add 1 cycle for REP,SEP

(9) Wait at cycle 2 for 2 cycles after $\overline{NMI}$ or $\overline{IRQ}$ active input

(10) R/$\overline{W}$ remains high during Reset

Abbreviations

AAB Absolute Address Bank
AAH Absolute Address High
AAL Absolute Address Low
AAVH Absolute Address Vector High
AAVL Absolute Address Vector Low
C Accumulator
D Direct Register
DBA Destination Bank Address
DBR Data Bank Register
DO Direct Offset
IDH Immediate Data High
IDL Immediate Data Low
IO Internal Operation
P Status Register
PBR Program Bank Register
PC Program Counter
R-M-W Read-Modify-Write
S Stack Address
SBA Source Bank Address
SO Stack Offset
VA Vector Address
x,y Index Registers
★ New W65C816/802 Addressing Modes
● New W65C02 Addressing Modes
Blank NMOS 6502 Addressing Modes

Recommended W65C816 and W65C802 Assembler Syntax Standards

Directives

Assembler directives are those parts of the assembly language source program which give directions to the assembler; this includes the definition of data area and constants within a program. This standard excludes any definitions of assembler directives.

Comments

An assembler should provide a way to use any line of the source program as a comment. The recommended way of doing this is to treat any blank line, or any line that starts with a semi-colon or an asterisk as a comment. Other special characters may be used as well.

The Source Line

Any line which causes the generation of a single W65C816 or W65C802 machine language instruction should be divided into four fields: a label field, the operation code, the operand, and the comment field.

The Label Field—The label field begins in column one of the line. A label must start with an alphabetic character, and may be followed by zero or more alphanumeric characters. An assembler may define an upper limit on the number of characters that can be in a label, so long as that upper limit is greater than or equal to six characters. An assembler may limit the alphabetic characters to upper-case characters if desired. If lower-case characters are allowed, they should be treated as identical to their upper-case equivalents. Other characters may be allowed in the label, so long as their use does not conflict with the coding of operand fields.

The Operation Code Field—The operation code shall consist of a three character sequence (mnemonic) from Table 3. It shall start no sooner than column 2 of the line, or one space after the label if a label is coded.

Many of the operation codes in Table 3 have duplicate mnemonics; when two or more machine language instructions have the same mnemonic, the assembler resolves the difference based on the operand.

If an assembler allows lower-case letters in labels, it must also allow lower-case letters in the mnemonic. When lower-case letters are used in the mnemonic, they shall be treated as equivalent to the upper-case counterpart. Thus, the mnemonics LDA, lda, and LdA must all be recognized, and are equivalent.

In addition to the mnemonics shown in Table 3, an assembler may provide the alternate mnemonics shown in Table 6.

Table 7. Alternate Mnemonics

Standard	Alias
BCC	BLT
BCS	BGE
CMP A	CMA
DEC A	DEA
INC A	INA
JSL	JSR
JML	JMP
TCD	TAD
TCS	TAS
TDC	TDA
TSC	TSA
XBA	SWA

JSL should be recognized as equivalent to JSR when it is specified with a long absolute address. JML is equivalent to JMP with long addressing forced.

The Operand Field—The operand field may start no sooner than one space after the operation code field. The assembler must be capable of at least twenty-four bit address calculations. The assembler should be capable of specifying addresses as labels, integer constants, and hexadecimal constants. The assembler must allow addition and subtraction in the operand field. Labels shall be recognized by the fact that they start alphabetic characters. Decimal numbers shall be recognized as containing only the decimal digits 0...9. Hexadecimal constants shall be recognized by prefixing the constant with a "$" character, followed by zero or more of either the decimal digits or the hexadecimal digits "A"..."F". If lower-case letters are allowed in the label field, then they shall also be allowed as hexadecimal digits.

All constants, no matter what their format, shall provide at least enough precision to specify all values that can be represented by a twenty-four bit signed or unsigned integer represented in two's complement notation.

Table 8 shows the operand formats which shall be recognized by the assembler. The symbol **d** is a label or value which the assembler can recognize as being less than $100. The symbol **a** is a label or value which the assembler can recognize as greater the $FF but less than $10000; the symbol **al** is a label or value that the assembler can recognize as being greater than $FFFF. The symbol EXT is a label which cannot be located by the assembler at the time the instruction is assembled. Unless instructed otherwise, an assembler shall assume that EXT labels are two bytes long. The symbols **r** and **rl** are 8 and 16 bit signed displacements calculated by the assembler.

Note that the operand does not determine whether or not immediate addressing loads one or two bytes; this is determined by the setting of the status register. This forces the requirement for a directive or directives that tell the assembler to generate one or two bytes of space for immediate loads. The directives provided shall allow separate settings for the accumulator and index registers.

The assembler shall use the <, >, and ^ characters after the # character in immediate address to specify which byte or bytes will be selected from the value of the operand. Any calculations in the operand must be performed before the byte selection takes place. Table 7 defines the action taken by each operand by showing the effect of the operator on an address. The column that shows a two byte immediate value show the bytes in the order in which they appear in memory. The coding of the operand is for an assembler which uses 32 bit address calculations, showing the way that the address should be reduced to a 24 bit value.

Table 8. Byte Selection Operator

Operand	One Byte Result	Two Byte Result
#$01020304	04	04 03
#<$01020304	04	04 03
#>$01020304	03	03 02
#^$01020304	02	02 01

In any location in an operand where an address, or expression resulting in an address, can be coded, the assembler shall recognize the prefix characters <, |, and >, which force one byte (direct page), two byte (absolute) or three byte (long absolute) addressing. In cases where the addressing mode is not forced, the assembler shall assume that the address is two bytes unless the assembler is able to determine the type of addressing required by context, in which case that addressing mode will be used. Addresses shall be truncated without error if an addressing mode is forced which does not require the entire value of the address. For example,

```
LDA   $0203        LDA   |$010203
```

are completely equivalent. If the addressing mode is not forced, and the type of addressing cannot be determined from context, the assembler shall assume that a two byte address is to be used. If an instruction does not have a short addressing mode (as in LDA, which has no direct page indexed by Y) and a short address is used in the operand, the assembler shall automatically extend the address by padding the most significant bytes with zeroes in order to extend the address to the length needed. As with immediate addressing, any expression evaluation shall take place before the address is selected; thus, the address selection character is only used once, before the address of expression.

The ! (exclamation point) character should be supported as an alternative to the | (vertical bar).

A long indirect address is indicated in the operand field of an instruction by surrounding the direct page address where the indirect address is found by square brackets; direct page addresses which contain sixteen-bit addresses are indicated by being surrounded by parentheses.

The operands of a block move instruction are specified as source bank, destination bank—the opposite order of the object bytes generated.

Comment Field—The comment field may start no sooner than one space after the operation code field or operand field depending on instruction type.

Table 9. Address Mode Formats

Addressing Mode	Format
Immediate	#d
	#a
	#al
	#EXT
	#·d
	#·a
	#·al
	#·EXT
	# ·d
	# ·a
	# ·al
	# ·EXT
	#^d
	#^a
	#^al
	#^EXT
Absolute	!d
	!a
	a
	!al
	!EXT
	EXT
Absolute Long	·d
	·a
	·al
	al
	·EXT
Direct Page	d
	·d
	·a
	·al
	·EXT
Accumulator	A
Implied Addressing	(no operand)
Direct Indirect Indexed	(d),y
	(·d),y
	(·a),y
	(·al),y
	(·EXT),y
Direct Indirect Indexed Long	[d],y
	[·d],y
	[·a],y
	[·al],y
	[·EXT],y
Direct Indexed Indirect	(d,x)
	(·d,x)
	(·a,x)
	(·al,x)
	(·EXT,x)
Direct Indexed by X	d,x
	·d,x
	·a,x
	·al,x
	·EXT,x
Direct Indexed by Y	d,y
	·d,y
	·a,y
	·al,y
	·EXT,y
Absolute Indexed by X	d,x
	!d,x
	a,x
	!a,x
	!al,x
	!EXT,x
	EXT,x

Addressing Mode	Format	
Absolute Indexed by Y	!d,y	
	d,y	
	a,y	
	!a,y	
	!al,y	
	!EXT,y	
	EXT,y	
Absolute Long Indexed by X	·d,x	
	·a,x	
	·al,x	
	al,x	
	·EXT,x	
Program Counter Relative and Program Counter Relative Long	d	(the assembler calculates **r** and **rl**)
	a	
	al	
	EXT	
Absolute Indirect	(d)	
	(!d)	
	(a)	
	(!a)	
	(!al)	
	(EXT)	
Direct Indirect	(d)	
	(·a)	
	(·al)	
	(·EXT)	
Direct Indirect Long	[d]	
	[·a]	
	[·al]	
	[·EXT]	
Absolute Indexed	(d,x)	
	(!d,x)	
	(a,x)	
	(!a,x)	
	(!al,x)	
	(EXT,x)	
	(!EXT,x)	
Stack Addressing	(no operand)	
Stack Relative Indirect Indexed	(d,s),y	
	(·d,s),y	
	(·a,s),y	
	(·al,s),y	
	(·EXT,s),y	
Block Move	d,d	
	d,a	
	d,al	
	d,EXT	
	a,d	
	a,a	
	a,al	
	a,EXT	
	al,d	
	al,a	
	al,al	
	al,EXT	
	EXT,d	
	EXT,a	
	EXT,al	
	EXT,EXT	

Note: The alternate ! (exclamation point) is used in place of the | (vertical bar).

Table 10. Addressing Mode Summary

Address Mode	Instruction Times In Memory Cycles		Memory Utilization In Number of Program Sequence Bytes	
	Original 8 Bit NMOS 6502	New W65C816	Original 8 Bit NMOS 6502	New W65C816
1. Immediate	2	2(3)	2	2(3)
2. Absolute	4(5)	4(3,5)	3	3
3. Absolute Long	—	5(3)	—	4
4. Direct	3(5)	3(3,4,5)	2	2
5. Accumulator	2	2	1	1
6. Implied	2	2	1	1
7. Direct Indirect Indexed (d),y	5(1)	5(1,3,4)	2	2
8. Direct Indirect Indexed Long [d], y	—	6(3,4)	—	2
9. Direct Indexed Indirect (d,x)	6	6(3,4)	2	2
10. Direct, X	4(5)	4(3,4,5)	2	2
11. Direct, Y	4	4(3,4)	2	2
12. Absolute, X	4(1,5)	4(1,3,5)	3	3
13. Absolute Long, X	—	5(3)	—	4
14. Absolute, Y	4(1)	4(1,3)	3	3
15. Relative	2(1,2)	2(2)	2	2
16. Relative Long	—	3(2)	—	3
17. Absolute Indirect (Jump)	5	5	3	3
18. Direct Indirect	—	5(3,4)	—	2
19. Direct Indirect Long	—	6(3,4)	—	2
20. Absolute Indexed Indirect (Jump)	—	6	—	3
21. Stack	3-7	3-8	1-3	1-4
22. Stack Relative	—	4(3)	—	2
23. Stack Relative Indirect Indexed	—	7(3)	—	2
24. Block Move X, Y, C (Source, Destination, Block Length)	—	7	—	3

NOTES:
1. Page boundary, add 1 cycle if page boundary is crossed when forming address.
2. Branch taken, add 1 cycle if branch is taken.
3. M = 0 or X = 0, 16 bit operation, add 1 cycle, add 1 byte for immediate.
4. Direct register low (DL) not equal zero, add 1 cycle.
5. Read-Modify-Write, add 2 cycles for M = 1, add 3 cycles for M = 0.

Caveats and Application Information

Stack Addressing

When in the Native mode, the Stack may use memory locations 000000 to 00FFFFF. The effective address of Stack, Stack Relative, and Stack Relative Indirect Indexed addressing modes will always be within this range. In the Emulation mode, the Stack address range is 000100 to 0001FF. The following opcodes and addressing modes will increment or decrement beyond this range when accessing two or three bytes:

JSL; JSR(a,x); PEA; PEI; PER; PHD; PLD; RTL; d,s; (d,s),y

Direct Addressing

The Direct Addressing modes are often used to access memory registers and pointers. The effective address generated by Direct; Direct,X and Direct,Y addressing modes will always be in the Native mode range 000000 to 00FFFF. When in the Emulation mode, the direct addressing range is 000000 to 0000FF, except for [Direct] and [Direct],Y addressing modes and the PEI instruction which will increment from 0000FE or 0000FF into the Stack area.

When in the Emulation mode and DH is not equal to zero, the direct addressing range is 00DH00 to 00DHFF, except for [Direct] and [Direct],Y addressing modes and the PEI instruction which will increment from 00DHFE or 00DHFF into the next higher page.

When in the Emulation mode and DL in not equal to zero, the direct addressing range is 000000 to 00FFFF.

Absolute Indexed Addressing (W65C816 Only)

The Absolute Indexed addressing modes are used to address data outside the direct addressing range. The W65C02 and W65C802 addressing range is 0000 to FFFF. Indexing from page FFXX may result in a 00YY data fetch when using the W65C02 or W65C802. In contrast, indexing from page ZZFFXX may result in ZZ+1,00YY when using the W65C816.

Future Microprocessors (i.e., W65C832)

Future WDC microprocessors will support all current W65C816 operating modes for both index and offset address generation.

$\overline{\text{ABORT}}$ Input (W65C816 Only)

$\overline{\text{ABORT}}$ should be held low for a period not to exceed one cycle. Also, if $\overline{\text{ABORT}}$ is held low during the Abort Interrupt sequence, the Abort Interrupt will be aborted. It is not recommended to abort the Abort Interrupt. The $\overline{\text{ABORT}}$ internal latch is cleared during the second cycle of the Abort Interrupt. Asserting the $\overline{\text{ABORT}}$ input after the following instruction cycles will cause registers to be modified:

- **Read-Modify-Write:** Processor status modified if $\overline{\text{ABORT}}$ is asserted after a modify cycle.
- **RTI:** Processor status will be modified if $\overline{\text{ABORT}}$ is asserted after cycle 3.
- **$\overline{\text{IRQ}}$, NMI, $\overline{\text{ABORT}}$ BRK, COP:** When $\overline{\text{ABORT}}$ is asserted after cycle 2, PBR and DBR will become 00 (Emulation mode) or PBR will become 00 (Native mode).

The Abort Interrupt has been designed for virtual memory systems. For this reason, asynchronous $\overline{\text{ABORT}}$'s may cause undesirable results due to the above conditions.

VDA and VPA Valid Memory Address Output Signals (W65C816 Only)

When VDA or VPA are high and during all write cycles, the Address Bus is always valid. VDA and VPA should be used to qualify all memory cycles. Note that when VDA and VPA are both low, invalid addresses may be generated. The Page and Bank addresses could also be invalid. This will be due to low byte addition only. The cycle when only low byte addition occurs is an optional cycle for instructions which read memory when the Index Register consists of 8 bits. This optional cycle becomes a standard cycle for the Store instruction, all instructions using the 16-bit Index Register mode, and the Read-Modify-Write instruction when using 8- or 16-bit Index Register modes.

Apple II, IIe, IIc and II+ Disk Systems (W65C816 Only)

VDA and VPA should not be used to qualify addresses during disk operation on Apple systems. Consult your Apple representative for hardware/software configurations.

DB/BA Operation when RDY is Pulled Low (W65C816 Only)

When RDY is low, the Data Bus is held in the data transfer state (i.e., $\phi 2$ high). The Bank address external transparent latch should be latched when the $\phi 2$ clock or RDY is low.

M/X Output (W65C816 Only)

The M/X output reflects the value of the M and X bits of the processor Status Register. The REP, SEP and PLP instructions may change the state of the M and X bits. Note that the M/X output is invalid during the instruction cycle following REP, SEP and PLP instruction execution. This cycle is used as the opcode fetch cycle of the next instruction.

All Opcodes Function in All Modes of Operation

It should be noted that all opcodes function in all modes of operation. However, some instructions and addressing modes are intended for W65C816 24-bit addressing and are therefore less useful for the W65C802. The following is a list of instructions and addressing modes which are primarily intended for W65C816 use:

JSL; RTL; [d]; [d],y; JMP al; JML; al; al,x

The following instructions may be used with the W65C802 even though a Bank Address is not multiplexed on the Data Bus:

PHK; PHB; PLB

The following instructions have "limited" use in the Emulation mode:

- The REP and SEP instructions cannot modify the M and X bits when in the Emulation mode. In this mode the M and X bits will always be high (logic 1).
- When in the Emulation mode, the MVP and MVN instructions use the X and Y Index Registers for the memory address. Also, the MVP and MVN instructions can only move data within the memory range 0000 (Source Bank) to 00FF (Destination Bank) for the W65C816, and 0000 to 00FF for the W65C802.

Indirect Jumps

The JMP (a) and JML (a) instructions use the direct Bank for indirect addressing, while JMP (a,x) and JSR (a,x) use the Program Bank for indirect address tables.

Switching Modes

When switching from the Native mode to the Emulation mode, the X and M bits of the Status Register are set high (logic 1), the high byte of the Stack is set to 01, and the high bytes of the X and Y Index Registers are set to 00. To save previous values, these bytes must always be stored before changing modes. Note that the low byte of the S, X and Y Registers and the low and high byte of the Accumulator (A and B) are not affected by a mode change.

How Hardware Interrupts, BRK, and COP Instructions Affect the Program Bank and the Data Bank Registers

When in the Native mode, the Program Bank register (PBR) is cleared to 00 when a hardware interrupt, BRK or COP is executed. In the Native mode, previous PBR contents is automatically saved on Stack.

In the Emulation mode, the PBR and DBR registers are cleared to 00 when a hardware interrupt, BRK or COP is executed. In this case, previous contents of the PBR are not automatically saved.

Note that a Return from Interrupt (RTI) should always be executed from the same "mode" which originally generated the interrupt.

Binary Mode

The Binary mode is set whenever a hardware or software interrupt is executed. The D flag within the Status Register is cleared to zero.

WAI Instruction

The WAI instruction pulls RDY low and places the processor in the WAI "low power" mode. $\overline{\text{NMI}}$, $\overline{\text{IRQ}}$ or RESET will terminate the WAI condition and transfer control to the interrupt handler routine. Note that an $\overline{\text{ABORT}}$ input will abort the WAI instruction, but will not restart the processor. When the Status Register I flag is set ($\overline{\text{IRQ}}$ disabled), the $\overline{\text{IRQ}}$ interrupt will cause the next instruction (following the WAI instruction) to be executed without going to the $\overline{\text{IRQ}}$ interrupt handler. This method results in the highest speed response to an $\overline{\text{IRQ}}$ input. When an interrupt

is received after an $\overline{ABORT}$ which occurs during the WAI instruction, the processor will return to the WAI instruction. Other than $\overline{RES}$ (highest priority), $\overline{ABORT}$ is the next highest priority, followed by $\overline{NMI}$ or $\overline{IRQ}$ interrupts.

STP Instruction

The STP instruction disables the $\phi 2$ clock to all circuitry. When disabled, the $\phi 2$ clock is held in the high state. In this case, the Data Bus will remain in the data transfer state and the Bank address will not be multiplexed onto the Data Bus. Upon executing the STP instruction, the $\overline{RES}$ signal is the only input which can restart the processor. The processor is restarted by enabling the $\phi 2$ clock, which occurs on the falling edge of the $\overline{RES}$ input. Note that the external oscillator must be stable and operating properly before $\overline{RES}$ goes high.

COP Signatures

Signatures 00-7F may be user defined, while signatures 80-FF are reserved for instructions on future microprocessors (i.e., W65C832). Contact WDC for software emulation of future microprocessor hardware functions.

WDM Opcode Use

The WDM opcode will be used on future microprocessors. For example, the new W65C832 uses this opcode to provide 32-bit floating-point and other 32-bit math and data operations. Note that the W65C832 will be a plug-to-plug replacement for the W65C816, and can be used where high-speed, 32-bit math processing is required. The W65C832 will be available in the near future.

RDY Pulled During Write

The NMOS 6502 does not stop during a write operation. In contrast, both the W65C02 and the W65C816 do stop during write operations. The W65C802 stops during a write when in the Native mode, but does not stop when in the Emulation mode.

MVN and MVP Affects on the Data Bank Register

The MVN and MVP instructions change the Data Bank Register to the value of the second byte of the instruction (destination bank address).

Interrupt Priorities

The following interrupt priorities will be in effect should more than one interrupt occur at the same time:

$\overline{RES}$	Highest Priority
$\overline{ABORT}$	
$\overline{NMI}$	
$\overline{IRQ}$	Lowest Priority

Transfers from 8-Bit to 16-Bit, or 16-Bit to 8-Bit Registers

All transfers from one register to another will result in a full 16-bit output from the source register. The destination register size will determine the number of bits actually stored in the destination register and the values stored in the processor Status Register. The following are always 16-bit transfers, regardless of the accumulator size:

TCS; TSC; TCD; TDC

Stack Transfers

When in the Emulation mode, a 01 is forced into SH. In this case, the B Accumulator will not be loaded into SH during a TCS instruction. When in the Native mode, the B Accumulator is transferred to SH. Note that in both the Emulation and Native modes, the full 16 bits of the Stack Register are transferred to the A, B and C Accumulators, regardless of the state of the M bit in the Status Register.

WDC Toolbox System-Emulator

Features

- Real-Time emulation of the W65C802/816 and the W65C02
- Uses an inexpensive Apple IIe Computer as host (software provided)
- 18K bytes of Emulation RAM, mappable in 2K blocks
- Optional RAM expansion to 256K
- Optional hardware Real-Time Trace Board
- Optional 802/816 Emulation Pod Unit
- Single-Step
- 48 bit trace memory of up to 2048 machine cycles
- Three 40-bit breakpoint control registers providing:
  - —Break on Address
  - —Break on Data
  - —Break on Control
  - —Break on User Status
  - —Break on Nth Occurance
  - —Coast Mode
- Microsecond execution timer
- Also available in In-Circuit-Evaluation chip or system test configuration

Product Overview

The Toolbox System-Emulator consists of a Main Unit and Interface Card that plugs into one of the Apple Computer's expansion slots. The Main Unit provides all necessary logic for breakpointing, single-stepping and mapping. In this configuration the user may perform basic debug operations or use the Toolbox in the Evaluation Mode.

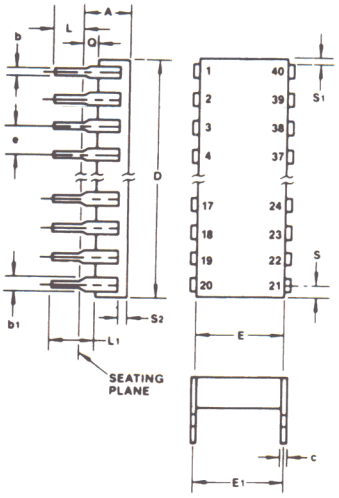
With the optional Real-Time Trace Board, the user now has 40 bits of trace memory within a window of 2048 machine cycles. A optional Emulation RAM Expansion Board is also available which increases the user's emulation RAM by 64K bytes or 256K bytes, with memory configuration under software control.

The Toolbox may be used with or without the optional Pod Unit. With the Pod Unit, the user can plug into the prototype microprocessor socket for hardware debug. Since the Main Unit remains the same regardless of the microprocessor used, the user does not have to learn a new set of Toolbox commands for each type of processor.

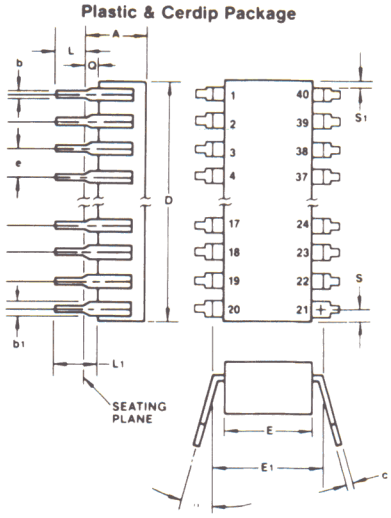
Apple IIe is a trademark of Apple Computer, Inc.

Packaging Information

Ceramic Package

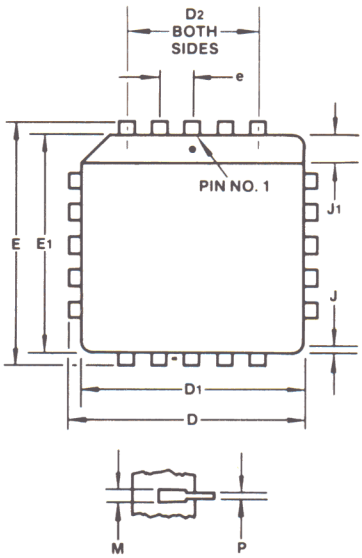


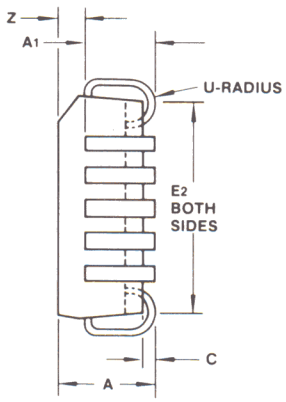
Plastic & Cerdip Package



40-PIN PACKAGE				
SYM-BOL	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	–	0.225	–	5.72
b	0.014	0.023	0.36	0.58
b1	0.030	0.070	0.76	1.78
c	0.008	0.015	0.20	0.38
D	–	2.096	–	53.24
E	0.510	0.620	12.95	15.75
E1	0.520	0.630	13.21	16.00
e	0.100 BSC		2.54 BSC	
L	0.125	0.200	3.18	5.08
L1	0.150	–	3.81	–
Q	0.020	0.060	0.51	1.52
S	–	0.098	–	2.49
S1	0.005	–	0.13	–
S2	0.005	–	0.13	–
α	0°	15°	0°	15°

Plastic Leaded Chip Carrier





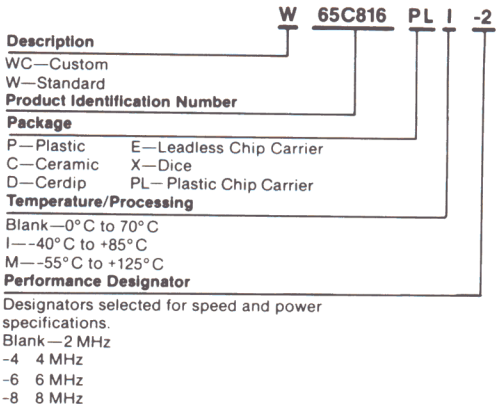
N = NO. LEADS

44-LEAD CARRIER				
SYM-BOL	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.165	0.180	4.20	4.57
A1	0.090	0.120	2.29	3.04
C	0.020	–	0.51	–
D	0.685	0.695	17.40	17.65
D1	0.650	0.656	16.510	16.662
D2	0.500 REF		12.70 BSC	
E	0.685	0.695	17.40	17.65
E1	0.650	0.656	16.510	16.662
E2	0.590	0.630	14.99	16.00
e	0.050 TYP		1.27 TYP	
J	–	0.020	–	0.51
J1	0.042	0.048	1.067	1.219
M	0.026	0.032	0.661	0.812
N	44		44	
P	0.013	0.021	0.331	0.533
Z	0.042	0.056	1.07	1.42

NOTES:

1. Power supply pins not available on the 40-pin version. These power supply pins have been added for improved high performance.
2. New pins, not available on 40-pin version.

Ordering Information



Sales Offices:
Technical or sales assistance may be requested from:
The Western Design Center, Inc.
2166 East Brown Road
Mesa, Arizona 85203
602/962-4545
TLX 6835057

WARNING:
MOS CIRCUITS ARE SUBJECT TO DAMAGE FROM STATIC DISCHARGE
Internal static discharge circuits are provided to minimize part damage due to environmental static electrical charge build-ups. Industry established recommendations for handling MOS circuits include

1 Ship and store product in conductive shipping tubes or in conductive foam plastic. Never ship or store product in non-conductive plastic containers or non-conductive plastic foam material.
2 Handle MOS parts only at conductive work stations.
3 Ground all assembly and repair tools.

Represented in your area by:

WDC reserves the right to make changes at any time and without notice.

The Western Design Center, Inc. 2166 E. Brown Rd./Mesa, AZ 85203 FAX6028356442
602/962-4545/TLX 6835057
Revised August 1986
Published in U.S.A. November 1986

Appendix A Roadmap to the Apple IIGS Technical Manuals

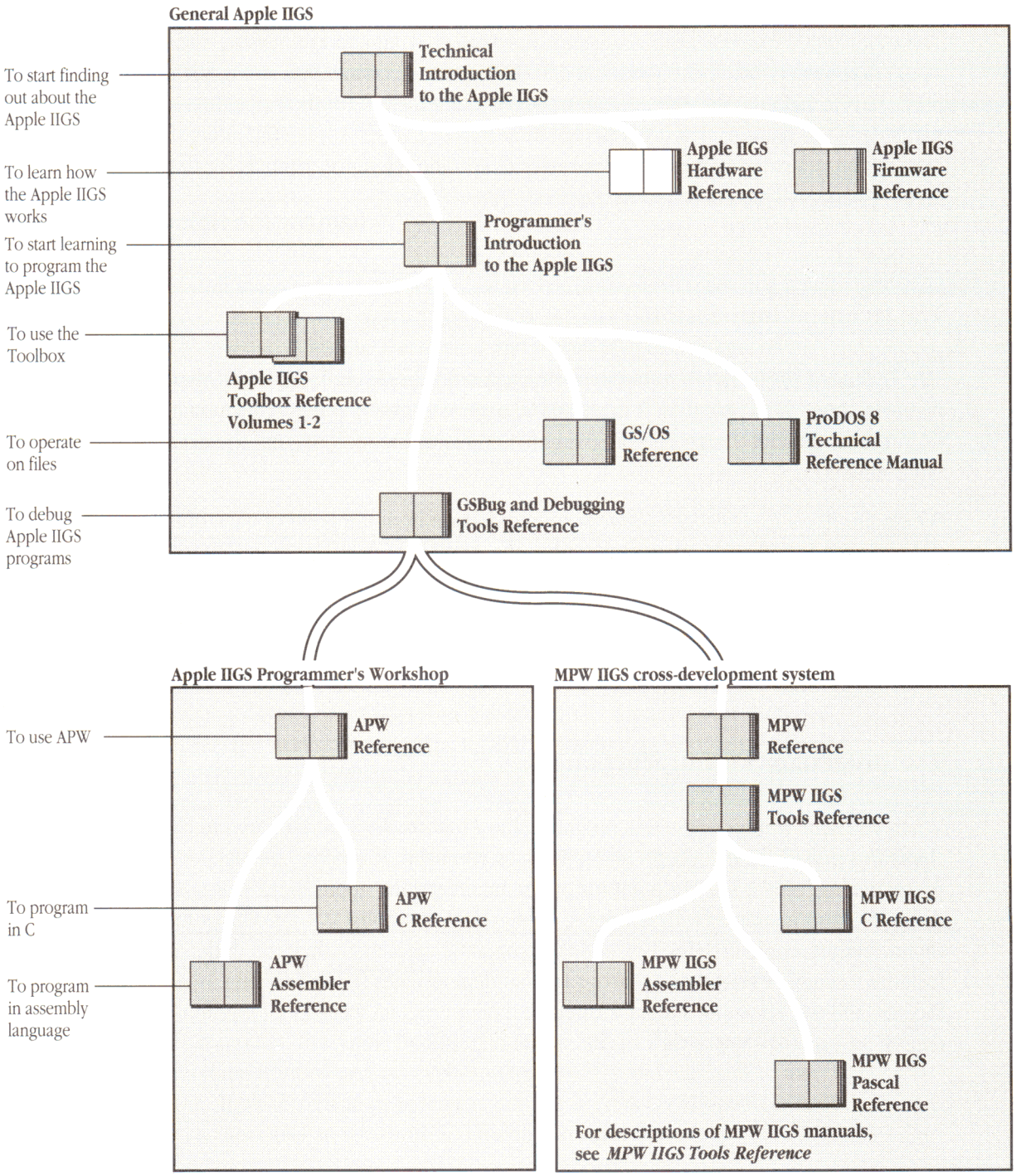
The Apple IIGS personal computer has many advanced features, making it more complex than earlier models of the Apple II. To describe it fully, Apple has produced a suite of technical manuals. Depending on the way you intend to use the Apple IIGS, you may need to refer to a select few of the manuals, 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** 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 specifications
Apple IIGS Toolbox Reference, Volume 2	More toolbox specifications
Apple IIGS Programmer's Workshop Reference	The development environment
Apple IIGS Programmer's Workshop Assembler Reference	Using the APW **Assembler**
Apple IIGS Programmer's Workshop C Reference	Using C on the Apple IIGS
ProDOS 8 Technical Reference Manual	Standard Apple II operating system
GS/OS Reference	Apple IIGS operating system and System Loader
Human Interface Guidelines: The Apple Desktop Interface	Guidelines for the desktop interface
Apple Numerics Manual	Numerics for all Apple computers

■ **Figure A-1** Roadmap to the technical manuals



The 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 those of other Apple computers. Having read the introductory manuals, you can 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 manual 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 Apple IIGS programs, 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 that use the Apple desktop interface (with windows, menus, and the mouse). 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.)

The machine reference manuals

There are two reference manuals for the machine itself: the *Apple IIGS Hardware Reference* (this book) 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 manual includes information about interrupt routines and low-level I/O subroutines for the serial ports, the disk port, and the Apple Desktop Bus interface, which controls the keyboard and the mouse. The manual also describes the Monitor, a low-level programming and debugging aid for assembly-language programs.

The toolbox reference manuals

Like the Macintosh® computer, 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. The *Apple IIGS Toolbox Reference,* Volume 2, contains information about the rest of the tools and also tells how to write and install your own tool set.

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 reference manual

The **Apple IIGS Programmer's Workshop (APW)** is the development environment for the Apple IIGS computer. APW is a set of programs that enables developers to create and debug application programs on the Apple IIGS. The *Apple IIGS Programmer's Workshop Reference* includes information about the APW Shell, Editor, Linker, Debugger, and utility programs; these are the parts of the workshop that all developers need, regardless of which programming language they use.

The APW reference manual describes the way you use the workshop to create an application, and includes examples and illustrations to show how this is done. In addition, this manual documents the APW Shell to provide the information necessary to write an APW utility or a language compiler for the workshop.

Included in the APW reference manual are complete descriptions of two standard Apple IIGS file formats: the text file format and the object module format. The text file format is used for all files written or read as "standard ASCII files" by Apple IIGS programs running under ProDOS 16. The object module format is used for the output of all APW compilers and for all files loadable by the Apple IIGS System Loader.

The programming-language reference manuals

Apple currently provides 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 use the assembler or compiler for that language. The manuals for the languages Apple provides are the *Apple IIGS Programmer's Workshop Assembler Reference* and the *Apple IIGS Programmer's Workshop C Reference.*

The *Apple IIGS Programmer's Workshop Reference* and the two programming-language manuals are available through the Apple Programmer's and Developer's Association (APDA™).

The operating-system reference manuals

There are three operating systems that run on the Apple IIGS: GS/OS™, ProDOS® 16, and ProDOS 8. Each operating system is described in its own manual: the *GS/OS Reference, Apple IIGS ProDOS 16 Reference,* or the *ProDOS 8 Technical Reference Manual.* GS/OS uses the full power of the Apple IIGS and is not compatible with earlier models of the Apple II. The *GS/OS Reference* describes the features of GS/OS and also includes information agbout the System Loader, which works closely with GS/OS to load programs into memory. If you are writing a program that does any file manipulation or that writes to or reads from a disk, you must have the *GS/OS Reference.* It is a rare applications programmer who will not need this book at some time; for system programmers, it is essential.

GS/OS maintains a complete set of ProDOS 16 calls and implements them just as ProDOS 16 does. Therefore, it is unlikely that you will need to refer to the *Apple IIGS ProDOS 16 Reference.*

ProDOS 8, previously called *ProDOS,* is the standard operating system for most Apple II computers with 8-bit CPUs. ProDOS also runs on the Apple IIGS, but it cannot access certain advanced Apple IIGS features. You need the *ProDOS 8 Technical Reference Manual* only if you are writing programs that can run on 8-bit Apple II computers.

APW and MPW manuals

Apple provides two development environments for writing Apple IIGS programs. See Figure A-1.

- The Apple IIGS Programmer's Workshop (APW): APW is a native development system—it runs on the Apple IIGS and produces Apple IIGS code. It is described in the *Apple IIGS Programmer's Workshop Reference* and related language books.
- The Apple IIGS Macintosh Programmer's Workshop (MPW): MPW is a cross-development system—it runs on the Macintosh, but produces Apple IIGS code. Much of MPW IIGS is described in separate MPW IIGS language reference manuals, but the parts needed for cross-development—the editor and the build tools—are described in the *Macintosh Programmer's Workshop Reference.* That book is the only Macintosh manual you need when using MPW IIGS.

The 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: The Apple Desktop Interface* and *Apple Numerics Manual.* If you develop programs for any Apple computer, you should know about those manuals.

The *Human Interface Guidelines* manual describes Apple's standards for the desktop interface of any program that runs on an Apple computer. If you are writing a commercial application for the Apple IIGS, you should be fully 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 for Binary Floating-Point Arithmetic* (IEEE Std 754-1985). 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 or robust arithmetic, you'll probably want to use the SANE routines in the Apple IIGS. The *Apple IIGS Toolbox Reference* tells how to use the SANE routines in your programs. The *Apple Numerics Manual* is the comprehensive reference for the SANE numerics routines.

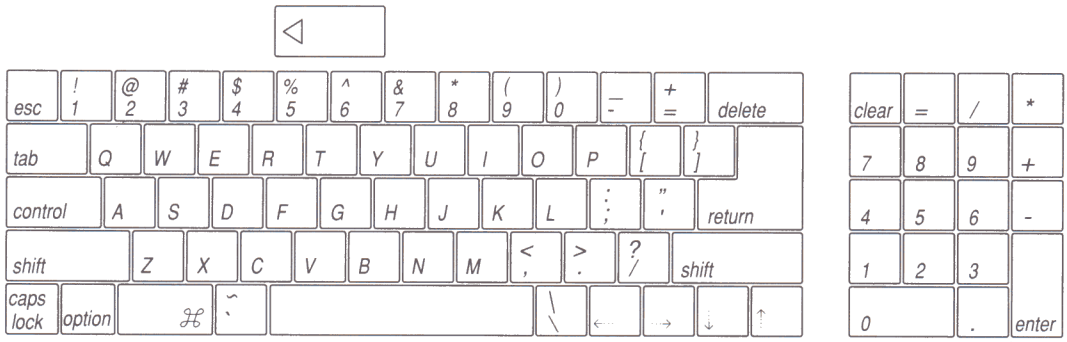
Appendix B International Keyboards

Apple makes different versions of the Apple IIGS for different countries. The different versions have different keyboards and display characters that reflect the different typing conventions of the different countries. The ADB keyboard on the Apple IIGS is available in the following versions:

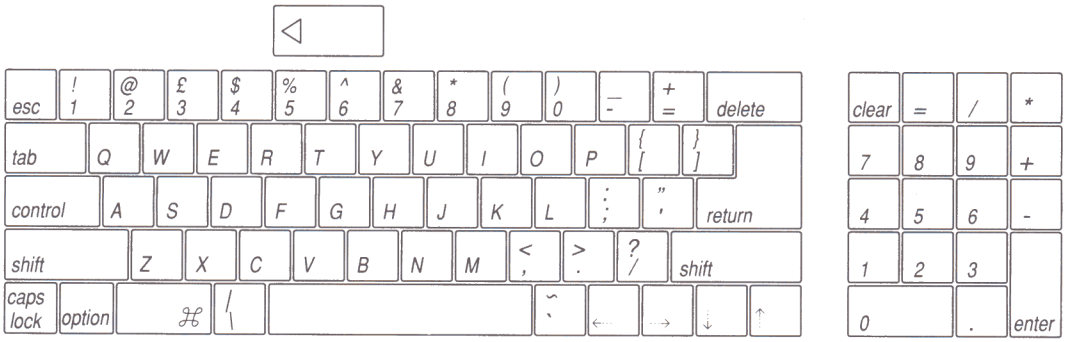
- U.S. English
- U.K. English
- Canadian
- French
- German
- Italian
- Spanish
- Swedish

The keyboards on the localized versions of the Apple IIGS are all mechanically the same; that is, the shapes and arrangement of the keys are the same—only the legends are different. The character decodings for the different versions are all stored in the keyboard decoder ROM. In addition to the international character sets listed above, the keyboard decoder ROM contains characters for Danish and the Dvorak keyboard layouts. Figures B-1 through B-8 show the legends on the different keyboards.

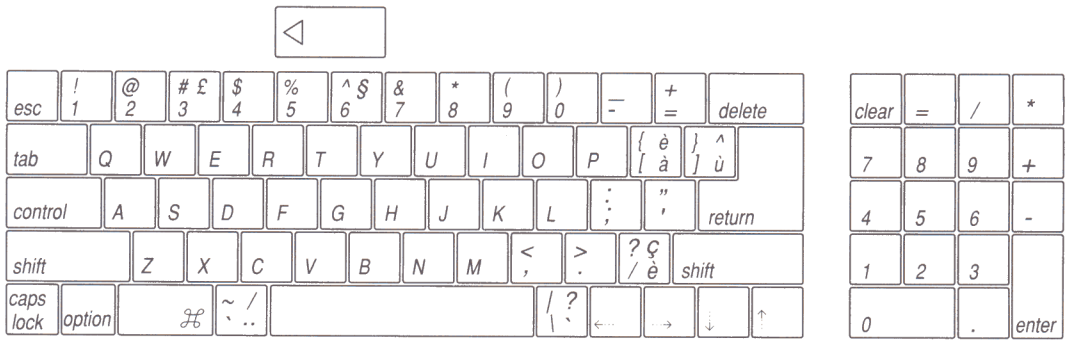
■ **Figure B-1** U.S. English keyboard



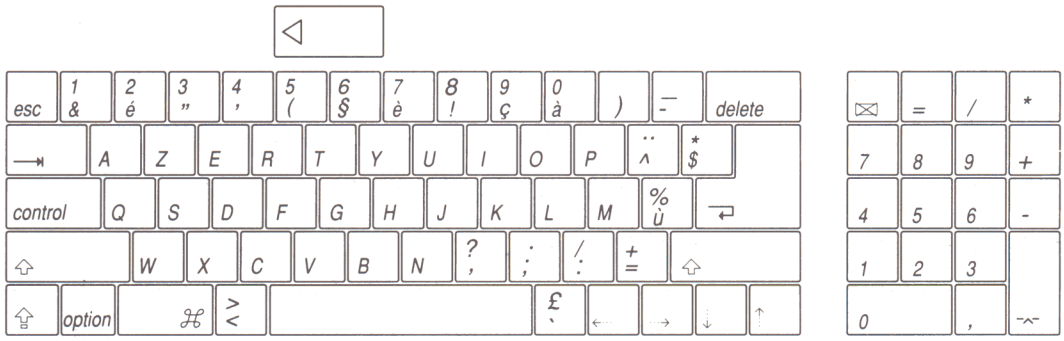
■ **Figure B-2** U.K. English keyboard



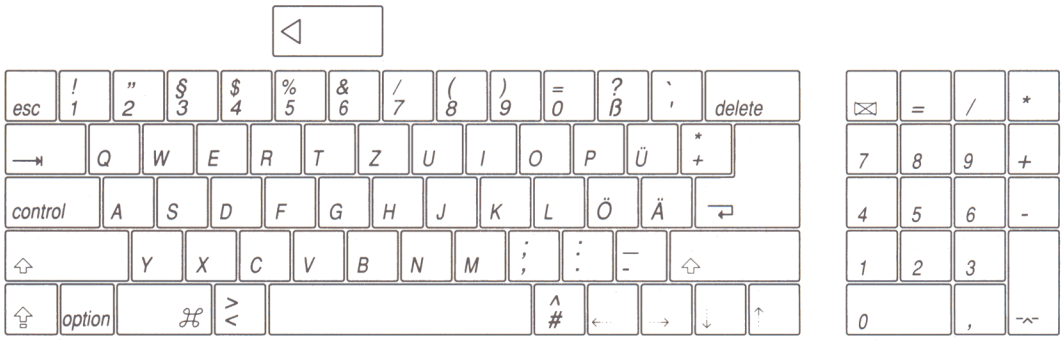
■ **Figure B-3** Canadian keyboard



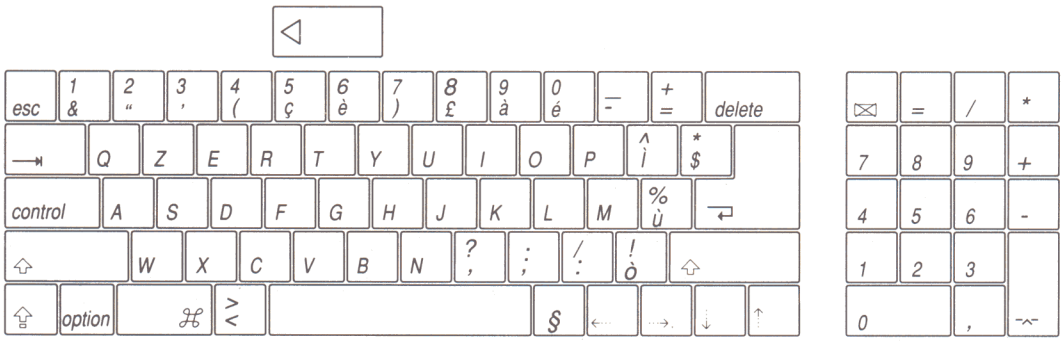
■ **Figure B-4** French keyboard



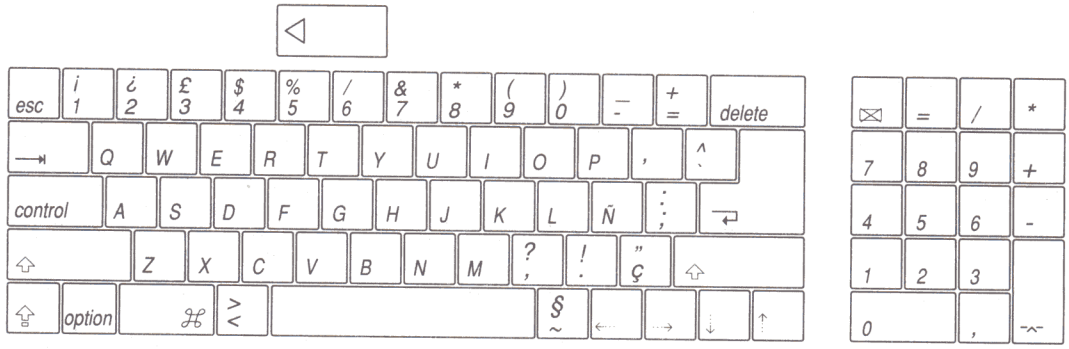
■ **Figure B-5** German keyboard



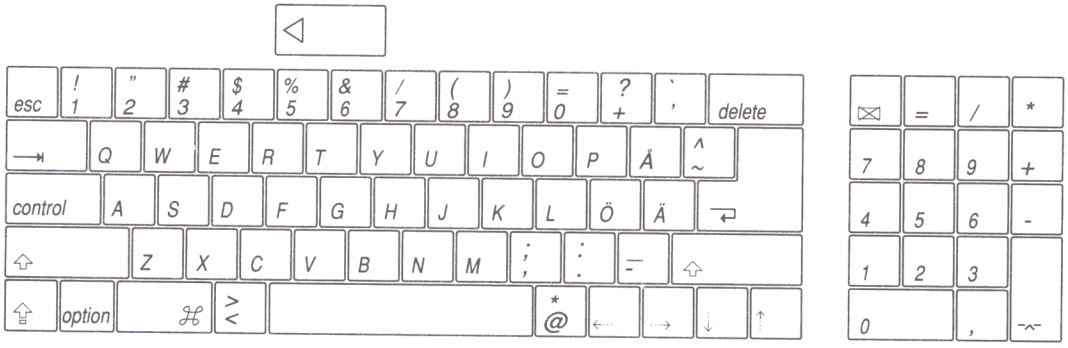
■ **Figure B-6** Italian keyboard



■ **Figure B-7** Spanish keyboard



■ **Figure B-8** Swedish keyboard



Appendix C **The Character Generator**

This appendix describes the hardware character generator for the 40-column and 80-column text displays. For information about text fonts in Super Hi-Res graphics displays, refer to the QuickDraw™ II tool set in *Apple IIGS Toolbox Reference,* volume 1.

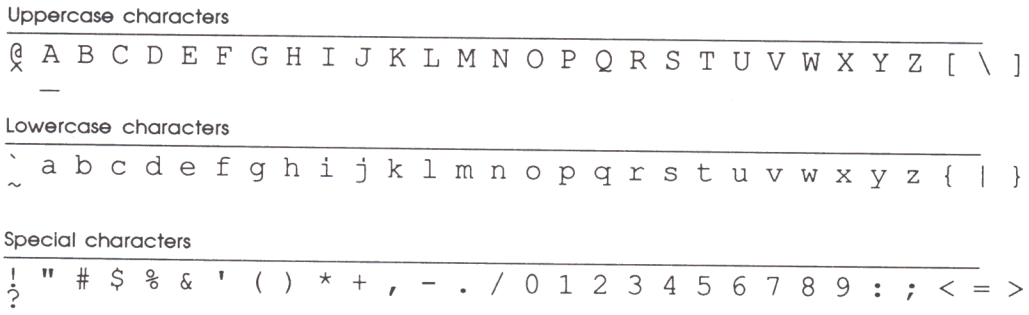
The character generator ROM

The ROM contains the dot patterns making up the characters in the 40-column and 80-column displays.

U.S. characters

Figure C-1 shows the characters for the U.S. versions of the computer.

■ **Figure C-1** U.S. characters



International characters

For other countries, localized versions of the Apple IIGS substitute appropriate characters for some of the special characters used in the text displays. Table C-1 shows those characters.

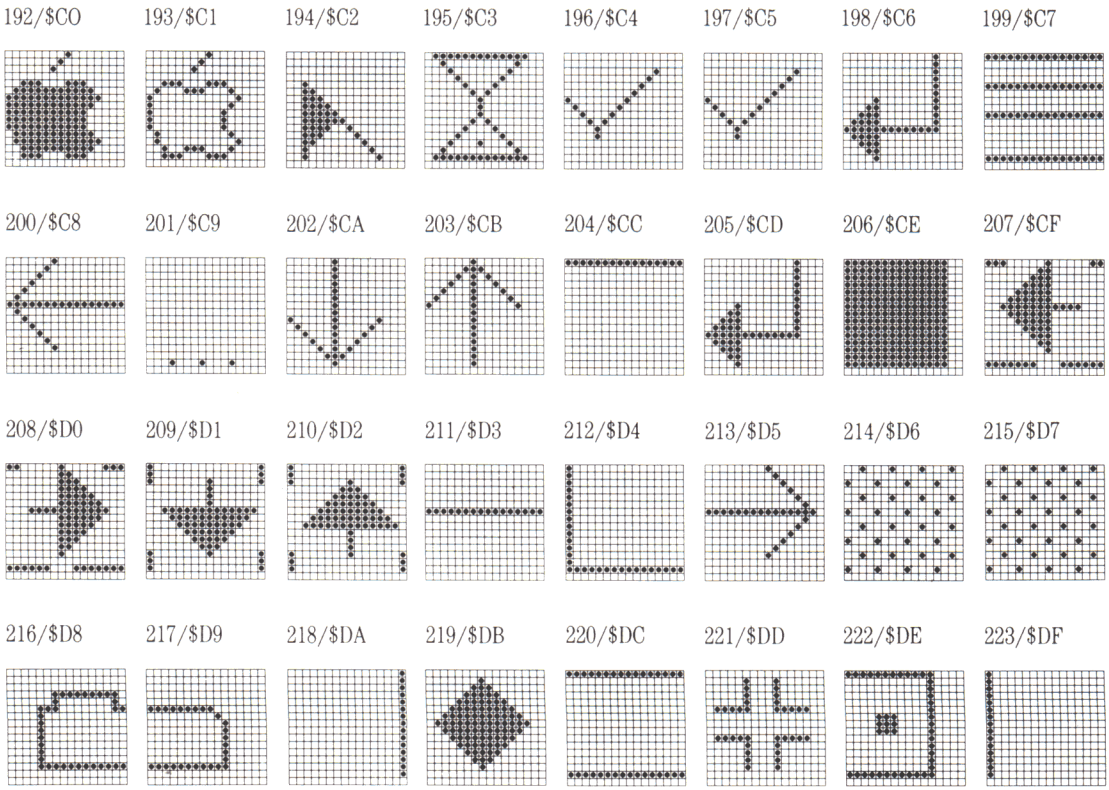
■ **Table C-1** International characters

Language	Equivalent characters									
U.S. English	#	@	[	\	]	`	{	|	}	~
U.K. English	£	@	[	\	]	`	{	|	}	~
French	£	à	"	ç	§	`	é	ù	è	¨
Danish	#	@	Æ	Ø	Å	`	æ	ø	å	~
Spanish	£	§	¡	Ñ	¿	`	"	ñ	ç	~
Italian	£	§	"	ç	é	ù	à	ò	è	ì
German	#	§	Ä	Ö	Ü	`	ä	ö	ü	ß
Swedish	#	@	Ä	Ö	Å	`	ä	ö	å	~

MouseText characters

The character ROM includes several graphic characters used in displaying the desktop user interface in text mode. Figure C-2 shows those characters.

■ **Figure C-2** MouseText characters



Appendix D **Conversion Tables**

This appendix briefly discusses bits and bytes and what they can represent, and peripheral identification numbers. It also contains conversion tables for hexadecimal to decimal and negative decimal, and a number of 8-bit codes.

These tables are intended for convenient reference. This appendix is not intended as a tutorial for the materials discussed. The brief section introductions are for orientation only.

Bits and bytes

This section discusses the relationships between bit values and their position within a byte. Here are some rules of thumb regarding the 65C816:

- A bit is a binary digit; it can be either a 0 or a 1.
- A bit can be used to represent any two-way choice. Some choices that a bit can represent in the Apple IIGS are listed in Table D-1.
- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of four bits.
- Four bits constitute a nibble (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 or a letter from A through F.
- Eight bits (two nibbles) make a byte.
- One byte can represent any of 16 times 16 or 256 values. The value can be specified by exactly two hexadecimal digits.
- Bits within a byte are numbered from bit 0 on the right to bit 7 on the left.
- The bit number is the same as the power of 2 that it represents, in a manner completely analogous to the digits in a decimal number.
- Each memory location in the Apple IIGS contains one 8-bit byte of data.
- How byte values are interpreted depends on whether the byte is an instruction in a language, part or all of an address, an ASCII code, or some other form of data. Tables D-6 through D-9 list some of the ways bytes are commonly interpreted.
- Two bytes make a word. The 16 bits of a word can represent any one of 256 times 256 or 65,536 different values.
- Three bytes make an address. The 24 bits of an address can represent any one of 256 times 65,536 or 16,777,216 different values.
- The 65C816 uses a 24-bit address to identify a memory location. It can therefore distinguish among 16,777,216 (16 MB) locations at any given time.
- A memory location is 1 byte of a 256-byte page. The low-order byte of an address specifies the location in the page. The middle byte specifies the memory page in a 65536-byte (64K) memory bank. The high-order byte specifies which 64K memory bank the byte is in.

■ **Table D-1** What a bit can represent

Context	Representing	0 =	1 =
Binary number	Place value	0	1 x place power
Logic	Condition	False	True
Any switch	Position	Off	On
Any switch	Position	Clear*	Set
Serial transfer	Beginning	Start	Carrier only
Serial transfer	Data	0 value	1 value
Serial transfer	Parity	SPACE	MARK
Serial transfer	End		Stop bit(s)
Serial transfer	Communication state	BREAK	Carrier
Program Status register bit N	Neg. result?	No	Yes
Program Status register bit v	Overflow?	No	Yes
Program Status register bit B	BRK command?	No	Yes
Program Status register bit d	Decimal mode?	No	Yes
Program Status register bit I	IRQ interrupts	Enabled	Disabled
Program Status register bit Z	Zero result?	No	Yes
Program Status register bit c	Carry required?	No	Yes

* Sometimes ambiguously termed *reset*.

Hexadecimal and decimal numbers

Use Table D-2 to find the binary equivalent of a known hexadecimal or decimal number.

■ **Table D-2** Binary, hexadecimal, and decimal equivalents

Binary	Hex	Dec	Binary	Hex	Dec
0000	$0	0	1000	$8	8
0001	$1	1	1001	$9	9
0010	$2	2	1010	$A	10
0011	$3	3	1011	$B	11
0100	$4	4	1100	$C	12
0101	$5	5	1101	$D	13
0110	$6	6	1110	$E	14
0111	$7	7	1111	$F	15

Use Table D-3 to determine the decimal weight of a hexadecimal digit in each of four places.

■ **Table D-3** Hexadecimal and decimal powers

Digit	$x000	$0x00	$00x0	$000x
F	61,440	3,840	240	15
E	57,344	3,584	224	14
D	53,248	3,328	208	13
C	49,152	3,072	192	12
B	45,056	2,816	176	11
A	40,960	2,560	160	10
9	36,864	2,304	144	9
8	32,768	2,048	128	8
7	28,672	1,792	112	7
6	24,576	1,536	96	6
5	20,480	1,280	80	5
4	16,384	1,024	64	4
3	12,288	768	48	3
2	8,192	512	32	2
1	4,096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up. For example:

```
$3C = ?          $FD47 = ?
$30 = 48         $F000 = 61440
$0C = 12         $ D00 =  3328
_________        $  40 =    64
$3C = 60         $   7 =     7

        $FD47 = 64839
```

To convert a decimal number to hexadecimal, subtract from the decimal number the largest decimal entry in the table that is less than it. Write down the hexadecimal digit (noting its place value) also. Now subtract the largest decimal number in the table that is less than the decimal remainder, and write down the next hexadecimal digit. Continue until you have 0 left. Add up the hexadecimal numbers. For example:

```
16215 = $ ?
16215 - 12288 = 3927     12288 = $7000
 3927 -  3840 =   87      3840 = $ F00
   87 -    80 =    7        80 = $  50
            7      7 = $    7

      16215 = $7F57
```

Hexadecimal and negative-decimal numbers

If a number is larger than decimal 32,767, Applesoft BASIC allows you to use the negative-decimal equivalent of the number. Table D-4 is set up to make it easy for you to convert a hexadecimal number directly to a negative-decimal number.

■ **Table D-4** Hexadecimal to negative-decimal conversion

Digit	$xooo	$$0x00	$$00x0	$$000x
	0	0	0	–1
E	–4,096	–256	–16	–2
D	–8,192	–512	–32	–3
C	–12,288	–768	–48	–4
B	–16,384	–1,024	–64	–5
A	–20,480	–1,280	–80	–6
9	–24,576	–1,536	–96	–7
8	–28,672	–1,792	–112	–8
7		–2,048	–128	–9
6		–2,304	–144	–10
5		–2,560	–160	–11
4		–2,816	–176	–12
3		–3,072	–192	–13
2		–3,328	–208	–14
1		–3,584	–224	–15
0		–3,840	–240	–16

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (0's included). Then add their values (ignoring their signs for a

moment). The resulting number, with a minus sign in front of it, is the desired negative-decimal number. For example:

```
$C010 = - ?
$C000:  -12288 $ 000:  - 3840 $  10:  -  224 $   0:  -   16

_______________
$C010   -16368
```

To convert a negative-decimal number directly to a positive-decimal number, add it to 65,536. (This addition ends up looking like subtraction.) For example:

```
-151 = + ?
65536 + (-151) = 65536 - 151 = 65385
```

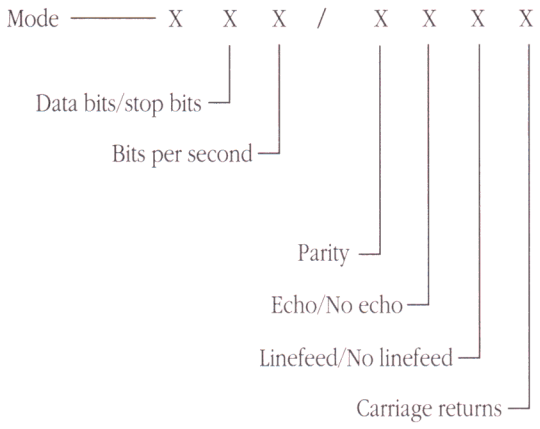
To convert a negative-decimal number to a hexadecimal number, first convert it to a positive-decimal number, then use Table D-3.

Peripheral identification numbers

Many Apple products now use peripheral identification numbers (called PIN numbers) as shorthand to designate serial device characteristics. The Apple II series *Universal Utilities* disk presents a menu from which to select the characteristics of, say, a printer or modem. From the selections made, it generates a PIN for the user. Other products have a ready-made PIN that the user can simply type in.

Figure D-1 shows the format of a PIN number. Table D-5 is a definition of the PIN number digits. When communication mode is selected, the seventh digit is ignored.

■ **Figure D-1** Format of PIN numbers



■ **Table D-5** Codes for PIN numbers

PIN number digit	Codes	PIN number digit	Codes
Mode	1 = Printer mode	Parity	1 = No parity
	2 = Communication mode*		2 = Even parity (total on = even)
			3 = Odd parity (total on = odd)
Data bits			4 = MARK parity (parity bit = 1)
Stop bits	1 = 6 data bits, 1 stop bit		5 = SPACE parity (parity bit = 0)
	2 = 6 data bits, 2 stop bit		
	3 = 7 data bits, 1 stop bit	Echo/	
	4 = 7 data bits, 2 stop bits	No echo	1 = Do not echo output on screen
	5 = 8 data bits, 1 stop bit		2 = Echo output on screen
	6 = 8 data bits, 2 stop bits		
		Line feed/	
Bits per		No line feed	1 = Do not generate LF after CR
second	1 = 110 bits per second		2 = Generate LF after CR
	2 = 300 bits per second		
	3 = 1200 bits per second	Carriage	
	4 = 2400 bits per second	returns	1 = Do not generate CR*
	5 = 4800 bits per second		2 = Generate CR after 40 characters
	6 = 9600 bits per second		3 = Generate CR after 72 characters
	7 = 19200 bits per second		4 = Generate CR after 80 characters
			5 = Generate CR after 132 characters

* If you select communication mode, then the seventh digit must be 1.

For example, 252/1111 means:

2 = Communication mode
5 = 8 data bits, 1 stop bit
2 = 300 baud (bits per second)
1 = No parity
1 = Do not echo output to display
1 = No line feed after carriage return
1 = Do not generate carriage returns

ASCII code conversion

Tables D-6 through D-9 show the first 128 ASCII characters ($0 through $7F). Note that only the low 7 bits of each character value are shown. The next 128 ASCII characters ($80 through $FF), not shown here, are identical to the one listed here, only with the high bit set. Unless otherwise noted, every ASCII character value above $7F (127 decimal) generates the same character as that value with the high bit off. Here is how to interpret these tables:

- The "Binary" column has the 7-bit code for each ASCII character.
- The "ASCII character" column gives the ASCII character name.
- The "Interpretation" column spells out the meaning of special symbols and abbreviations, where necessary.
- The "What to type" column indicates what keystrokes generate the ASCII character (where it is not obvious).
- The columns marked "Primary" and "Alternate" indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

  Note that the values $40 through $5F (and $C0 through $DF) in the alternate character set are displayed as MouseText characters (Figure C-2) if the firmware is set to do so, or if the firmware is bypassed .

◆ *Note:* The primary and alternate displayed character sets in Tables D-6 through D-9 are the result of firmware mapping. See "Text Displays," in Chapter 4, for more details on character sets.

■ **Table D-6** Control characters, high bit off

Binary	Dec	Hex	ASCII character	Interpretation	What to type	Primary	Alternate
0000000	0	$00	NUL	Blank (null)	Control-@	@	@
0000001	1	$01	SOH	Start of Header	Control-A	A	A
0000010	2	$02	STX	Start of Text	Control-B	B	B
0000011	3	$03	ETX	End of Text	Control-C	C	C
0000100	4	$04	EOT	End of Transm.	Control-D	D	D
0000101	5	$05	ENQ	Enquiry	Control-E	E	E
0000110	6	$06	ACK	Acknowledge	Control-F	F	F
0000111	7	$07	BEL	Bell	Control-G	G	G
0001000	8	$08	BS	Backspace	Control-H or Left-Arrow-H	H	H
0001001	9	$09	HT	Horizontal Tab	Control-I or Tab	I	I
0001010	10	$0A	LF	Line Feed	Control-J or Down-Arrow-J	J	J
0001011	11	$0B	VT	Vertical Tab	Control-K or Up-Arrow	K	K
0001100	12	$0C	FF	Form Feed	Control-L	L	L
0001101	13	$0D	CR	Carriage Return	Control-M or Return	M	M
0001110	14	$0E	SO	Shift Out	Control-N	N	N
0001111	15	$0F	SI	Shift In	Control-O	O	O
0010000	16	$10	DLE	Data Link Escape	Control-P	P	P
0010001	17	$11	DC1	Device Control 1	Control-Q	Q	Q
0010010	18	$12	DC2	Device Control 2	Control-R	R	R
0010011	19	$13	DC3	Device Control 3	Control-S	S	S
0010100	20	$14	DC4	Device Control 4	Control-T	T	T
0010101	21	$15	NAK	Neg. Acknowledge	Control-U or Right-Arrow	U	U
0010110	22	$16	SYN	Synchronization	Control-V	V	V
0010111	23	$17	ETB	End of Text Blk.	Control-W	W	W
0011000	24	$18	CAN	Cancel	Control-X	X	X
0011001	25	$19	EM	End of Medium	Control-Y	Y	Y
0011010	26	$1A	SUB	Substitute	Control-Z	Z	Z
0011011	27	$1B	ESC	Escape	Control-[or Escape	[	[
0011100	28	$1C	FS	File Separator	Control-\	\	\
0011101	29	$1D	GS	Group Separator	Control-]	]	]
0011110	30	$1E	RS	Record Separator	Control-^	^	^
0011111	31	$1F	US	Unit Separator	Control-_	_	_

■ **Table D-7** Special characters, high bit off

Binary	Dec	Hex	ASCII character	Interpretation	What to type	Primary	Alternate
0100000	32	$20	SP	Space	Space bar		
0100001	33	$21	!			!	!
0100010	34	$22	"			"	"
0100011	35	$23	#			#	#
0100100	36	$24	$			$	$
0100101	37	$25	%			%	%
0100110	38	$26	&			&	&
0100111	39	$27	'	Apostrophe		'	'
0101000	40	$28	(			(	(
0101001	41	$29	)			)	)
0101010	42	$2A	*			*	*
0101011	43	$2B	+			+	+
0101100	44	$2C	,	Comma		,	,
0101101	45	$2D	-	Hyphen		-	-
0101110	46	$2E	.	Period		.	.
0101111	47	$2F	/			/	/
0110000	48	$30	0			0	0
0110001	49	$31	1			1	1
0110010	50	$32	2			2	2
0110011	51	$33	3			3	3
0110100	52	$34	4			4	4
0110101	53	$35	5			5	5
0110110	54	$36	6			6	6
0110111	55	$37	7			7	7
0111000	56	$38	8			8	8
0111001	57	$39	9			9	9
0111010	58	$3A	:			:	:
0111011	59	$3B	;			;	;
0111100	60	$3C	<			<	<
0111101	61	$3D	=			=	=
0111110	62	$3E	>			>	>
0111111	63	$3F	?			?	?

■ **Table D-8** Uppercase characters, high bit off

Binary	Dec	Hex	ASCII character	Interpretation	What to type	Primary	Alternate
1000000	64	$40	@			@	
1000001	65	$41	A			*A*	
1000010	66	$42	B			*B*	
1000011	67	$43	C			*C*	
1000100	68	$44	D			*D*	
1000101	69	$45	E			*E*	
1000110	70	$46	F			*F*	
1000111	71	$47	G			*G*	
1001000	72	$48	H			*H*	
1001001	73	$49	I			*I*	
1001010	74	$4A	J			*J*	
1001011	75	$4B	K			*K*	
1001100	76	$4C	L			*L*	
1001101	77	$4D	M			*M*	
1001110	78	$4E	N			*N*	
1001111	79	$4F	O			*O*	
1010000	80	$50	P			*P*	
1010001	81	$51	Q			*Q*	
1010010	82	$52	R			*R*	
1010011	83	$53	S			*S*	
1010100	84	$54	T			*T*	
1010101	85	$55	U			*U*	
1010110	86	$56	V			*V*	
1010111	87	$57	W			*W*	
1011000	88	$58	X			*X*	
1011001	89	$59	Y			*Y*	
1011010	90	$5A	Z			*Z*	
1011011	91	$5B	[	Opening bracket		*[*	
1011100	92	$5C	\	Back slash		\	
1011101	93	$5D	]	Closing bracket		*]*	
1011110	94	$5E	^	Caret		^	
1011111	95	$5F	_	Underline		_	

Note: If the high bit is set, the MouseText character is replaced with the equivalent from the primary character set.

■ **Table D-9** Lowercase characters, high bit off

Binary	Dec	Hex	ASCII character	Interpretation	What to type	Primary	Alternate
1100000	96	$60	`	Grave accent		`	
1100001	97	$61	a			*!*	**a**
1100010	98	$62	b			*"*	**b**
1100011	99	$63	c			*#*	**c**
1100100	100	$64	d			*$*	**d**
1100101	101	$65	e			*%*	**e**
1100110	102	$66	f			*&*	**f**
1100111	103	$67	g			*'*	**g**
1101000	104	$68	h			*(*	**h**
1101001	105	$69	i			*)*	**i**
1101010	106	$6A	j			*	**j**
1101011	107	$6B	k			+	**k**
1101100	108	$6C	l			,	**l**
1101101	109	$6D	m			-	**m**
1101110	110	$6E	n			.	**n**
1101111	111	$6F	o			*/*	**o**
1110000	112	$70	p			*0*	**p**
1110001	113	$71	q			*1*	**q**
1110010	114	$72	r			*2*	**r**
1110011	115	$73	s			*3*	**s**
1110100	116	$74	t			*4*	**t**
1110101	117	$75	u			*5*	**u**
1110110	118	$76	v			*6*	**v**
1110111	119	$77	w			*7*	**w**
1111000	120	$78	x			*8*	**x**
1111001	121	$79	y			*9*	**y**
1111010	122	$7A	z			*:*	**z**
1111011	123	$7B	{	Opening brace		;	{
1111100	124	$7C	|	Vertical line		<	|
1111101	125	$7D	}	Closing brace		=	}
1111110	126	$7E	~	Overline (tilde)		>	~
1111111	127	$7F	DEL	Delete/rubout		*?*	**DEL**

Appendix E **Frequently Used Tables**

This appendix contains frequently used tables from throughout the manual. The original table number is given in a footnote to the table.

■ **Table E-1*** Language-card bank select switches

Name	Action	Location	Function
	R	$C080	Read this location to read RAM, write-protect RAM, and use $D000 bank 2.
ROMIN	RR	$C081	Read this location twice to read ROM, write-enable RAM, and use $D000 bank 2.
	R	$C082	Read this location to read ROM, write-protect RAM, and use $D000 bank 2.
LCBANK2	RR	$C083	Read this location twice to read RAM, write-enable RAM, and use $D000 bank 2.
	R	$C088	Read this location to read RAM, write-protect RAM, and use $D000 bank 1.
	RR	$C089	Read this location twice to read ROM, write-enable RAM, and use $D000 bank 1.
	R	$C08A	Read this location to read ROM, write-protect RAM, and use $D000 bank 1.
	RR	$C08B	Read this switch twice to read RAM, write-enable RAM, and use $D000 bank 1.
RDLCBNK2	R7	$C011	Read this location and test bit 7 for switch status: $D000 bank 2 (1) or bank 1 (0).
RDLCRAM	R7	$C012	Read this location and test bit 7 for switch status: RAM (1) or ROM (0).
SETSTDZP	W	$C008	Write this location to use main bank, page 0 and page 1.
SETALTZP	W	$C009	Write this location to use auxiliary bank, page 0 and page 1.
RDALTZP	R7	$C016	Read this location and test bit 7 for switch status: auxiliary (1) or main (0) bank.

* Table 3-1

■ **Table E-2*** Auxiliary-memory select switches

Name	Function	Location		Notes
		Hex	Dec	
RDCARDRAM	Read auxiliary memory	$C003	49155	Write
RDMAINRAM	Read main memory	$C002	49154	Write
RDRAMRD	Read switch status	$C013	49171	Read and test bit 7 (1=auxiliary, 0=main)
WRCARDRAM	Write auxiliary memory	$C005	49157	Write
WRMAINRAM	Write main memory	$C004	49156	Write
RDRAMWRT	Read switch status	$C014	49172	Read and test bit 7 (1=auxiliary, 0=main)
SET80COL	Access display page	$C001	49153	Write
CLR80COL	Use RAM switches ($C002-5,13,14)	$C000	49152	Write
RD80COL	Read switch status	$C018	49176	Read and test bit 7 (1=80-column access on, 0=80-column access off)
TXTPAGE2	Text Page 2 on (auxiliary memory)	$C055	49237	Read or write
TXTPAGE1	Text Page 1 on (main memory)	$C054	49236	Read or write
RDPAGE2	Read switch status	$C01C	49180	Read and test bit 7 (1=Page 2, 0=Page 1)
HIRES	Access Hi-Res pages	$C057	49239	Read or write
LORES	Use RAM switches ($C002-5,13,14)	$C056	49238	Read or write
RDHIRES	Read switch status	$C01D	49181	Read and test bit 7 (1=HIRES on, 0=off)
SETALTZP	Auxiliary stack and direct page	$C009	49161	Write
SETSTDZP	Main stack and direct page	$C008	49160	Write
RDALTZP	Read switch status	$C016	49174	Read and test bit 7 (1=auxiliary, 0=main)

* Table 3-3

■ **Table E-3*** Standard Apple II video display specifications

Display modes	40-column text; map: Figure 4-5. 80-column text; map: Figure 4-6. Lo-Res color graphics; map: Figure 4-7. Hi-Res color graphics; map: Figure 4-8. Double Hi-Res color graphics; map: Figure 4-9.
Text capacity	24 lines by 80 columns (character positions).
Character set	128 ASCII characters. (See Appendix C for a list of display characters.)
Display formats	Normal, inverse, flashing, MouseText (Table 4-10).
Lo-Res color graphics	16 colors (Table 4-15): 40 horizontal by 48 vertical; map: Figure 4-7.
Hi-Res color graphics	6 colors (Table 4-14): 140 horizontal by 192 vertical (restricted). Black-and-White: 280 horizontal by 192 vertical; map: Figure 4-8.
Double Hi-Res color graphics	16 colors (Table 4-16): 140 horizontal by 192 vertical (no restrictions). Black-and-White: 560 horizontal by 192 vertical; map: Figure 4-9.

* Table 4-4

■ **Table E-4*** Video display locations

Display mode	Display page	Lowest address		Highest address	
		Hex	Dec	Hex	Dec
40-column text,	1	$0400	1024	$07FF	2047
Lo-Res graphics	2†	$0800	2048	$0BFF	3071
80-column text	1	$0400	1024	$07FF	2047
	2†	$0800	2048	$0BFF	3071
Hi-Res graphics	1	$2000	8192	$3FFF	16383
	2	$4000	16384	$5FFF	24575
Double High-Res	1‡	$2000	8192	$3FFF	16383
graphics	2‡	$4000	16384	$5FFF	24575

* Table 4-6

† Lo-Res graphics on Page 2 is not supported by firmware; for instructions on how to switch pages, refer to the section "Display Mode Switching" in Chapter 4.

‡ See the section "Double Hi-Res Graphics" in Chapter 4.

■ **Table E-5*** Display soft switches

Name	Action†	Location		Function
CLR80COL	W	$C000	(49152)	Disable 80-column store.
SET80COL	W	$C001	(49153)	Enable 80-column store.
CLR80VID	W	$C00C	(49164)	Disable 80-column hardware.
SET80VID	W	$C00D	(49165)	Enable 80-column hardware.
CLRALTCHAR	W	$C00E	(49166)	Normal lowercase character set; flashing uppercase character set.
SETALTCHAR	W	$C00F	(49167)	Normal, inverse character set; no flashing.
RD80COL	R7	$C018	(49176)	Read CLR/SET80COL switch: 1 = 80-column store enabled.
RDVBL BAR	R7	$C019	(49177)	Read vertical blanking(VBL): 1 = not VBL.
RDTEXT	R7	$C01A	(49178)	Read TXTCLR/TXTSET switch: 1 = text mode enabled.
RDMIX	R7	$C01B	(49179)	Read MIXCLR/MIXSET switch: 1 = mixed mode enabled.
RDPAGE2	R7	$C01C	(49180)	Read TXTPAGE1/TXTPAGE2 switch: 1 = text Page 2 selected.
RDHIRES	R7	$C01D	(49181)	Read HIRES switch: 1 = Hi-Res mode enabled.
ALTCHARSET	R7	$C01E	(49182)	Read CLRALTCHAR/SETALTCHAR switch: 1 = alternate character set in use.
RD80VID	R7	$C01F	(49183)	Read CLR80VID/SET80VID switch: 1 = 80-column hardware in use.
RDDHIRES	R7	$C07F	(49279)	Read SETAN3/CLRAN3 switch: 1 = Double Hi-Res graphics mode selected.
TXTCLR	R/W	$C050	(49232)	Select standard Apple II graphics mode or, if MIXSET on, mixed mode.
TXTSET	R/W	$C051	(49233)	Select text mode only.
MIXCLR	R/W	$C052	(49234)	Clear mixed mode.
MIXSET	R/W	$C053	(49235)	Select mixed mode.
TXTPAGE1	R/W	$C054	(49236)	Select text Page 1.
TXTPAGE2	R/W	$C055	(49237)	Select text Page 2, or, if SET80COL on, text Page 1 in auxiliary memory.
LORES	R/W	$C056	(49238)	Select Lo-Res graphics mode.
HIRES	R/W	$C057	(49239)	Select Hi-Res graphics mode, or, if SETAN3 is on, select Double Hi-Res graphics mode.
SETAN3	R/W	$C05E	(49246)	Enable Double Hi-Res graphics mode.
CLRAN3	R/W	$C05F	(49247)	Disable Double Hi-Res graphics mode.

* Table 4-7 † *W* means write anything to the location, *R* means read the location, *R/W* means read or write, and *R7* means read the location and then check bit 7.

■ Table E-6* Text window memory locations

Window parameter	Location		Minimum value		Normal values				Maximum values			
					40-column		80-column		40-column		80-column	
	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
Left edge	32	$20	00	$00	00	$00	00	$00	39	$27	79	$4F
Width	33	$21	00	$00	40	$28	80	$50	40	$28	80	$50
Top edge	34	$22	00	$00	00	$00	00	$00	23	$17	23	$17
Bottom edge	35	$23	01	$01	24	$18	24	$18	24	$18	24	$18

* Table 4-9

■ Table E-7* Display character sets

Hex values	Primary character set		Alternate character set	
	Character type	Format	Character type	Format
$00–$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
$20–$3F	Special characters	Inverse	Special characters	Inverse
$40–$5F	Uppercase letters	Flashing	MouseText	Inverse
$60–$7F	Special characters	Flashing	Lowercase letters	Inverse
$80–$9F	Uppercase letters	Normal	Uppercase letters	Normal
$A0–$BF	Special characters	Normal	Special characters	Normal
$C0–$DF	Uppercase letters	Normal	Uppercase letters	Normal
$E0–$FF	Lowercase letters	Normal	Lowercase letters	Normal

* Table 4-10

■ Table E-8* Lo-Res graphics colors

Nibble value			Nibble value		
Dec	**Hex**	**Color**	**Dec**	**Hex**	**Color**
0	$00	Black	8	$08	Brown
1	$01	Deep red	9	$09	Orange
2	$02	Dark blue	10	$0A	Light gray
3	$03	Purple	11	$0B	Pink
4	$04	Dark green	12	$0C	Light green
5	$05	Dark gray	13	$0D	Yellow
6	$06	Medium blue	14	$0E	Aquamarine
7	$07	Light blue	15	$0F	White

Note: Colors may vary, depending on the controls on the monitor or television set.

* Table 4-14

■ Table E-9* Hi-Res graphics colors

Bits 0–6	Bit 7 off	Bit 7 on
Adjacent columns off	Black 1	Black 2
Even columns on	Purple	Blue
Odd columns on	Green	Orange
Adjacent columns on	White 1	White 2

Note: Colors may vary, depending on the controls on the monitor or television set.

* Table 4-15

■ **Table E-10*** Double Hi-Res graphics colors

Repeated color pattern	ab0	mb1	ab2	mb3	Bit
Black	$00	$00	$00	$00	0000
Deep red	$08	$11	$22	$44	0001
Brown	$44	$08	$11	$22	0010
Orange	$4C	$19	$33	$66	0011
Dark green	$22	$44	$08	$11	0100
Dark gray	$2A	$55	$2A	$55	0101
Green	$66	$4C	$19	$33	0110
Yellow	$6E	$5D	$3B	$77	0111
Dark blue	$11	$22	$44	$08	1000
Purple	$19	$33	$66	$4C	1001
Light gray	$55	$2A	$55	$2A	1010
Pink	$5D	$3B	$77	$6E	1011
Medium blue	$33	$66	$4C	$19	1100
Light blue	$3B	$77	$6E	$5D	1101
Aquamarine	$77	$6E	$5D	$3B	1110
White	$7F	$7F	$7F	$7F	1111

* Table 4-16

■ **Table E-11*** Palette and color starting addresses

Palettenumber	Color $0	Color $1	...	Color $E	Color $F
$0	$9E00–01	$9E02–03	...	$9E1C–1D	$9E1E–1F
$1	$9E20–21	$9E22–23	...	$9E3C–3D	$9E3E–3F
$2	$9E40–41	$9E42–43	...	$9E5C–5D	$9E5E–5F
.	.	.		.	.
.	.	.		.	.
.	.	.		.	.
$F	$9FE0–E1	$9FE2–E3	...	$9FFC–FD	$9FFE–FF

* Table 4-20

■ **Table E-12*** GLU registers

GLU registers	Address	Type
Sound Control register	$C03C	R/W
Data register	$C03D	R/W
Address Pointer register, low byte	$C03E	R/W
Address Pointer register, high byte	$C03F	R/W

* Table 5-1

■ **Table E-13*** DOC register addresses

Oscillator number	Frequency Low register	Frequency High register	Volume register	Data register	Wavetable Pointer register	Control register	Wavetable Size register
$00	$00	$20	$40	$60	$80	$A0	$C0
$01	$01	$21	$41	$61	$81	$A1	$C1
$02	$02	$22	$42	$62	$82	$A2	$C2
$03	$03	$23	$43	$63	$83	$A3	$C3
$04	$04	$24	$44	$64	$84	$A4	$C4
$05	$05	$25	$45	$65	$85	$A5	$C5
$06	$06	$26	$46	$66	$86	$A6	$C6
$07	$07	$27	$47	$67	$87	$A7	$C7
$08	$08	$28	$48	$68	$88	$A8	$C8
$09	$09	$29	$49	$69	$89	$A9	$C9
$0A	$0A	$2A	$4A	$6A	$8A	$AA	$CA
$0B	$0B	$2B	$4B	$6B	$8B	$AB	$CB
$0C	$0C	$2C	$4C	$6C	$8C	$AC	$CC
$0D	$0C	$2D	$4D	$6D	$8D	$AD	$CD
$0E	$0E	$2E	$4E	$6E	$8E	$AE	$CE
$0F	$0F	$2F	$4F	$6F	$8F	$AF	$CF
$10	$10	$30	$50	$70	$90	$B0	$D0
$11	$11	$31	$51	$71	$91	$B1	$D1
$12	$12	$32	$52	$72	$92	$B2	$D2
$13	$13	$33	$53	$73	$93	$B3	$D3
$0B	$0B	$2B	$4B	$6B	$8B	$AB	$CB
$0C	$0C	$2C	$4C	$6C	$8C	$AC	$CC
$0D	$0C	$2D	$4D	$6D	$8D	$AD	$CD
$0E	$0E	$2E	$4E	$6E	$8E	$AE	$CE
$0F	$0F	$2F	$4F	$6F	$8F	$AF	$CF
$10	$10	$30	$50	$70	$90	$B0	$D0
$11	$11	$31	$51	$71	$91	$B1	$D1
$12	$12	$32	$52	$72	$92	$B2	$D2
$13	$13	$33	$53	$73	$93	$B3	$D3
$14	$14	$34	$54	$74	$94	$B4	$D4
$15	$15	$35	$55	$75	$95	$B5	$D5
$16	$16	$36	$56	$76	$96	$B6	$D6
$17	$17	$37	$57	$77	$97	$B7	$D7
$18	$18	$38	$58	$78	$98	$B8	$D8
$19	$19	$39	$59	$79	$99	$B9	$D9

(Continued)

■ **Table E-13*** DOC register addresses (Continued)

Oscillator number	Frequency Low register	Frequency High register	Volume register	Data register	Wavetable Pointer register	Control register	Wavetable Size register
$1A	$1A	$3A	$5A	$7A	$9A	$BA	$DA
$1B	$1B	$3B	$5B	$7B	$9B	$BB	$DB
$1C	$1C	$3C	$5C	$7C	$9C	$BC	$DC
$1D	$1D	$3D	$5D	$7D	$9D	$BD	$DD
$1E†	$1E	$3E	$5E	$7E	$9E	$BE	$DE
$1F†	$1F	$3F	$5F	$7F	$9F	$BF	$DF

* Table 5-4

† These oscillators are reserved for system use. Use of these oscillators by the user may result in a system crash.

■ **Table E-14*** Disk-port soft switches

Address	Description
$C0E0	Stepper motor phase 0 low
$C0E1	Stepper motor phase 0 high
$C0E2	Stepper motor phase 1 low
$C0E3	Stepper motor phase 1 high
$C0E4	Stepper motor phase 2 low
$C0E5	Stepper motor phase 2 high
$C0E6	Stepper motor phase 3 low
$C0E7	Stepper motor phase 3 high
$C0E8	Spindle motor enabled
$C0E9	Spindle motor disabled
$C0EA	Drive 0 select
$C0EB	Drive 1 select
$C0EC	Q6 select bit low
$C0ED	Q6 select bit high
$C0EE	Q7 select bit low
$C0EF	Q7 select bit high

* Table 7-3

■ Table E-15* The IWM states

Q7	Q6	Spindle motor	Operation
0	0	1	Read Data register
0	1	x	Read Status register
1	0	x	Read Handshake register
1	1	0	Write Mode register
1	1	1	Write Data register

* Table 7-4

■ Table E-16* SCC Command and SCC data register addresses

Register	Channel A	Channel B
SCC Command	$C039	$C038
SCC data	$C03B	$C03A

* Table 7-10

■ Table E-17* Annunciator memory locations

Annunciator			Address	
Number	Pin†	State	Hex	Dec
0	15	Off	$C058	49240
		On	$C059	49241
1	14	Off	$C05A	49242
		On	$C05B	49243
2	13	Off	$C05C	49244
		On	$C05D	49245
3	12	Off	$C05E	49246
		On	$C05F	49247

* Table 7-14

† Pin numbers given are for the 16-pin IC connector on the circuit board.

■ Table E-18* Secondary I/O memory locations

Soft switch	Address		Definition
	Dec	Hex	
SPKR	49200	$C030	Toggle speaker (read only).
CLRAN0	49240	$C058	Clear annunciator 0.
SETAN0	49241	$C059	Set annunciator 0.
CLRAN1	49242	$C05A	Clear annunciator 1.
SETAN1	49243	$C05B	Set annunciator 1.
CLRAN2	49244	$C05C	Clear annunciator 2.
SETAN2	49245	$C05D	Set annunciator 2.
CLRAN3	49246	$C05E	Clear annunciator 3.
SETAN3	49247	$C05F	Set annunciator 3.
BUTN3	49248	$C060	Read switch 3 (read only).
BUTN0	49249	$C061	Read switch 0 (read only).
BUTN1	49250	$C062	Read switch 1 (read only).
BUTN2	49251	$C063	Read switch 2 (read only).
PADDL0	$C064	49252	Read analog-input 0.
PADDL1	$C065	49253	Read analog-input 1.
PADDL2	$C066	49254	Read analog-input 2.
PADDL3	$C067	49255	Read analog-input 3.
PTRIG	49264	$C070	Analog-input reset.

* Table 7-15

■ Table E-19* Peripheral-card RAM memory locations

Base address	Slot number						
	1	2	3	4	5	6	7
$0478	$0479	$047A	$047B	$047C	$047D	$047E	$047F
$04F8	$04F9	$04FA	$04FB	$04FC	$04FD	$04FE	$04FF
$0578	$0579	$057A	$057B	$057C	$057D	$057E	$057F
$05F8	$05F9	$05FA	$05FB	$05FC	$05FD	$05FE	$05FF
$0678	$0679	$067A	$067B	$067C	$067D	$067E	$067F
$06F8	$06F9	$06FA	$06FB	$06FC	$06FD	$06FE	$06FF
$0778	$0779	$077A	$077B	$077C	$077D	$077E	$077F
$07F8	$07F9	$07FA	$07FB	$07FC	$07FD	$07FE	$07FF

* Table 8-5

■ **Table E-20*** Peripheral-card I/O base addresses

Base address	Slot number						
	1	2	3	4	5	6	7
$C080	$C090	$C0A0	$C0B0	$C0C0	$C0D0	$C0E0	$C0F0
$C081	$C091	$C0A1	$C0B1	$C0C1	$C0D1	$C0E1	$C0F1
$C082	$C092	$C0A2	$C0B2	$C0C2	$C0D2	$C0E2	$C0F2
$C083	$C093	$C0A3	$C0B3	$C0C3	$C0D3	$C0E3	$C0F3
$C084	$C094	$C0A4	$C0B4	$C0C4	$C0D4	$C0E4	$C0F4
$C085	$C095	$C0A5	$C0B5	$C0C5	$C0D5	$C0E5	$C0F5
$C086	$C096	$C0A6	$C0B6	$C0C6	$C0D6	$C0E6	$C0F6
$C087	$C097	$C0A7	$C0B7	$C0C7	$C0D7	$C0E7	$C0F7
$C088	$C098	$C0A8	$C0B8	$C0C8	$C0D8	$C0E8	$C0F8
$C089	$C099	$C0A9	$C0B9	$C0C9	$C0D9	$C0E9	$C0F9
$C08A	$C09A	$C0AA	$C0BA	$C0CA	$C0DA	$C0EA	$C0FA
$C08B	$C09B	$C0AB	$C0BB	$C0CB	$C0DB	$C0EB	$C0FB
$C08C	$C09C	$C0AC	$C0BC	$C0CC	$C0DC	$C0EC	$C0FC
$C08D	$C09D	$C0AD	$C0BD	$C0CD	$C0DD	$C0ED	$C0FD
$C08E	$C09E	$C0AE	$C0BE	$C0CE	$C0DE	$C0EE	$C0FE
$C08F	$C09F	$C0AF	$C0BF	$C0CF	$C0DF	$C0EF	$C0FF

* Table 8-6

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. Converts data from parallel to serial form and vice versa. An ACIA handles serial transmission and reception and RS-232-C signals under the control of its internal registers, which can be set and changed by firmware or software. Compare **Serial Communications Controller.**

ADB: See **Apple Desktop Bus.**

address: A number that specifies the location of a single byte of memory. Addresses can be given as decimal or hexadecimal integers. The Apple IIGS has addresses ranging from 0 to 16,777,215 (in decimal) or from $000000 to $FFFFFF (in hexadecimal). A complete address consists of a 4-bit bank number ($00 to $FF) followed by a 16-bit address within that bank ($0000 to $FFFF).

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

Apple Desktop Bus (ADB): An input bus and protocol for connecting keyboards, mouse devices, and graphics tablets to the Apple IIGS. Chapter 6 provides details of the ADB.

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

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

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. Compare **standard Apple II.**

Apple IIGS Programmer's Workshop (APW): The development environment for the Apple IIGS computer. It consists of a set of programs that facilitate the writing, compiling, and debugging of Apple IIGS application programs.

Apple IIGS toolbox: A collection of built-in routines on the Apple IIGS that programs can call to perform many commonly needed functions. Functions within the toolbox are grouped into tool sets.

application program or **application:** (1) A program that performs a specific task useful to the computer user, such as word processing, database management, or graphics. Compare **controlling program, software, system software.**

application program or **application:** (2) On the Apple IIGS, a program (such as the APW Shell) that accesses ProDOS 16 and the toolbox directly and that can be called or exited via the Quit call. ProDOS 16 applications are file type $B3.

APW: See **Apple IIGS Programmer's Workshop.**

ASCII: Acronym for *American Standard Code for Information Interchange.* (2) The standard alphabetic, numeric, control, and special characters represented by hexadecimal values $0 through $FF. *ASCII data* refers to these universal 256 codes.

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 program that produces object files (programs that contain machine-language code) from source files written in assembly language. The opposite of *disassembler.*

auxiliary slot: The special expansion slot within 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.

bank: A 64K (65,536-byte) portion of the Apple IIGS internal memory. An individual bank is specified by the value of one of the 65C816 microprocessor's bank registers.

bank-switched memory: On Apple II computers, that part of the language-card memory in which two 4K portions of memory share the same address range ($D000–$DFFF).

binary: A method of numeric representation using a base-2 system. Valid digits are 0 and 1. Compare **hexadecimal, decimal.**

bit: Short for *binary digit.* In the binary system, the smallest unit of information, consisting of 0 or 1.

bit map: A set of bits that represents the positions and states of a corresponding set of items. In graphics, video pixels are represented by a bit or bits in video display memory. See also **graphics.**

block: (1) A unit of data storage or transfer, typically 512 bytes. (2) A contiguous, page-aligned region of computer memory of arbitrary size, allocated by the Memory Manager. Also called a *memory block.*

block device or **block I/O device:** A device that transfers data to or from a computer in multiples of one block (512 bytes) of characters at a time. Disk drives are block devices.

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* : Starting up is often accomplished by first loading a small program, which then reads a larger program into memory. The program is said to "pull itself up by its own bootstraps."

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 or a ProDOS 16 I/O buffer.

byte: A unit of information consisting of a sequence of 8 bits. A byte can take any value between 0 and 255 ($0 and $FF hexadecimal). The value can represent an instruction, number, character, or logical state.

carriage return (\r): A control code (ASCII 13) generated by the Return key; in APW C, equal to newline (\n).

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, such as a television picture tube, that produces images on a phosphor-coated screen.

central processing unit (CPU): The part of the computer that performs the actual computations in machine language. See also **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. Most characters are represented in the computer as 1-byte values.

character device or **character I/O device:** A device that transfers data to or from a computer as a stream of individual characters. Keyboards and printers are character devices.

character generator: The IC responsible for providing all text and special characters to the computer that may be displayed on the video monitor.

CMOS: Acronym for *complementary metal oxide semiconductor,* 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.

color fringing: The rainbow-like effect that appears around text characters when they are displayed in color on most color video monitors. This fringing is unavoidable because the color-detection circuitry of most composite video color monitors cannot respond fast enough to the changing of the color information during the text portion of the display. Displaying text in monochrome makes it more readable.

command: (1) In the Standard C Library, a parameter that tells a function which of several actions to perform. (2) In the APW Shell, a word that tells APW which utility to execute.

Command key: A modifier key on the Apple IIGS keyboard, marked with both an Apple icon and a propeller, the icon used on the equivalent key on some Macintosh keyboards. It performs the same functions as the Open Apple key on standard Apple II machines.

composite video: A standard video signal that includes all color and timing information that is needed by a composite video monitor. Several video standards are in use around the world: NTSC video is used in northern America and Japan; PAL video is used in much of Europe; SECAM is used in the USSR and many other countries. The Apple IIGS is capable of generating both NTSC and PAL video. Compare **RGB.**

controlling program: A program that loads and runs other programs, without itself relinquishing control. A controlling program is responsible for shutting down its subprograms and freeing their memory space when they are finished. A shell, for example, is a controlling program.

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 computers that usually causes an Applesoft BASIC program or command to stop immediately.

CPU: See **central processing unit** and **microprocessor.**

cursor: A graphic icon displayed by the operating system or application program that indicates where the next input from the user is expected. Different styles of cursors are used with the Apple IIGS: an arrow, an underbar, a vertical bar, and an inverse video block.

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

data block: A 512-byte portion of a ProDOS 16 standard file that consists of whatever kind of information the file may contain.

decimal: A method of numeric representation using a base-10 system. Valid digits are 0 through 9. Compare **hexadecimal, binary.**

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.

desk accessories: Small, special-purpose programs that are available to the user regardless of which application is running. The Control Panel is an example of a desk accessory.

desktop user interface: The visual appearance of a program and the way in which it interacts with the user. 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.

device: A piece of equipment (hardware) used in conjunction with a computer and under the computer's control. Also called a *peripheral device* because such equipment is often physically separate from, but attached to, the computer.

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

digital: Descriptive of that which uses signals representing characters or numbers, signals being of discrete rather than continuously variable values, or those produced by pulses of one current or voltage value.

Digital Oscillator Chip (DOC): An integrated circuit in the Apple IIGS that contains 32 digital oscillators, each of which can generate a sound from stored digital waveform data in a wavetable.

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

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

direct memory access (DMA): A means of fast data transfer into or out of computer memory to or from a computer peripheral. A peripheral device, usually a card in a peripheral I/O expansion slot, puts the 65C816 microprocessor in an idle state, and takes control of the

computer for a short period of time. Data in memory may be directly accessed without the time-consuming usual **handshaking** and **protocol.**

directory: A file that contains a list of the names and locations of other files stored on a disk. Directories are either volume directories or subdirectories. A directory is sometimes called a *catalog.*

directory file: A directory. One of the two principal categories of ProDOS 16 files. Directory files contain specially formatted entries that give the names and disk locations of other files.

direct page: A page (256 bytes) of bank $00 of Apple IIGS memory, any part of which can be addressed with a short (1-byte) address because its high-address byte is always $00 and its middle-address byte is the value of the 65C816 direct register. Coresident programs or routines can have their own direct pages at different locations. The direct page corresponds to the 6502 processor's zero page. The term *direct page* is often used informally to refer to any part of the lower portion of the direct-page/stack space.

direct-page/stack space: A portion of bank $00 of Apple IIGS memory reserved for a program's direct page and stack. Initially, the 65C816 processor's direct register contains the base address of the space, and its stack register contains the highest address. In use, the stack grows downward from the top of the direct-page/stack space, and the lower part of the space contains direct-page data.

direct register: A hardware register in the 65C816 processor that specifies the start of the direct page.

disk drive: A computer peripheral device that stores digital data on a revolving magnetic surface. Disk drives may be floppy disk drives (which use a removable, flexible mylar disk as the medium) or hard disk drives (which use a fixed aluminum platter as the medium). Disk drives retain the information after the computer is turned off, but are capable of altering the data as requested by the computer program.

disk operating system: An operating system whose principal function is to manage files and communication with one or more disk drives. DOS and ProDOS are two families of Apple II disk operating systems.

Disk II drive: A type of disk drive made and sold by Apple Computer, Inc. for use with the Apple II, Apple II Plus, and Apple IIe. It uses 5.25-inch disks.

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.

DMA: See **direct memory access.**

DOC: See **Digital Oscillator Chip.**

DOS: Acronym for *disk operating system.* An Apple II disk operating system.

Double Hi-Res: A high-resolution graphics display mode on Apple II computers with at least 128K of RAM, consisting of an array of points 560 wide by 192 high with 16 colors.

dynamic ROM: A form of read-only memory (ROM) in which data is retained in memory while the computer power is off, but is lost as soon as the system is turned on.

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 also **m flag, x flag.**

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: The 8-bit configuration of the 65C816 processor in which it functions like a 6502 processor in all respects except clock speed.

event-driven program: A program that waits in a loop until it detects an event such as a click of the mouse button. The occurrence of the event causes the program to take a specific action based on the event.

external device: See **device.**

external reference: A reference to a symbol that is defined in another segment. External references must be to global symbols.

fatal error: An error serious enough that the computer must halt execution.

firmware: Programs stored permanently in read-only memory (ROM); most provide an interface to system hardware. Such programs (for example, the Control Panel 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. Compare **hardware, software.**

FPI: Abbreviation for *Fast Processor Interface.* A custom integrated circuit that incorporates most of the memory organization and address-decoding functions of the Apple IIGS. One of this IC's functions is to slow the system clock to 1.024 MHz whenever access to banks $E0 and E1 is detected.

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.

fringing: Also known as *color fringing.* The undesirable effect of rainbow-like colors obscuring text on the video monitor. Occurs when 80-column text is displayed in color.

General Logic Unit: A class of custom integrated circuits used as interfaces between different parts of the computer.

GLU: See **General Logic Unit.**

graphics: The display by a computer on a video monitor of data in memory, to visually represent figures, charts, graphs, or icons. In the Apple IIGS, each pixel on the monitor screen is directly controllable by bits in the screen. See also **bit map.**

hand control: A hand-held device with a knob and pushbutton that provides the user with a means for inputting stimuli to the computer for the purpose of controlling the application program. Usually used in conjunction with game software. Compare **joystick.**

handshaking: The exchange of status information between a DCE (Data Communications Equipment) and a DTE (Data Terminal Equipment), usually a computer and a peripheral device, 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 also **XOFF, XON.**

hardware: Collectively, electronic circuit components and associated fittings and attachments. In computers, the computer itself (the processor), disk drives, and other peripheral equipment. The saying goes, "If you can touch it, it's hardware. If you can't, it's software." Compare **software.**

hertz (Hz): The unit of frequency of vibration or oscillation, defined as the number of *cycles per second*. Named for the physicist Heinrich Hertz. See also **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. In Apple manuals, 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: Referring to the most significant part of a numerical quantity. In normal representation, the *high-order bit* of a binary value is in the leftmost position; likewise, the *high-order byte* of a binary word or long word consists of the leftmost 8 bits.

Hi-Res: A high-resolution graphics display mode on the Apple II family of computers, consisting of an array of pixels 140 wide by 192 high in six colors or 280 wide by 192 high in monochrome.

IC: See **integrated circuit.**

image: A representation of the contents of memory. A code image consists of machine-language instructions or data that may be loaded unchanged into memory.

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 index register* and the *Y index register.*

input device: The keyboard is the main input device for the Apple IIGS. Other possible input devices are the mouse and the graphics tablet. Almost any device may be used as an ADB input device, as long as it conforms to the Apple Desktop Bus protocol. Chapter 6 provides details on the ADB.

input/output (I/O): The transmitting and receiving of data to and from peripheral or built-in devices.

integrated circuit (IC): A miniature electronic circuit consisting of many thousands of transistors and other electronic components by processing a chip of semiconductor material. This chip is then cast in a plastic or ceramic package with metal leads or "legs" used to connect it to a circuit board. Categories of ICs are labeled due to their construction process: *monolothic, hybrid,* and *thin-film* are some. Almost any electronic circuit may be miniaturized and made into an integrated circuit.

interrupt: A request made of the microprocessor by a device, either built-in or external, to receive urgent data or respond to a recent event. Disk drives make interrupt requests of the microprocessor, as do the real-time clock and the mouse firmware in the Apple IIGS.

interrupt handler: A program, associated with a particular external device, that executes whenever that device sends an interrupt signal to the computer. The interrupt handler performs its tasks during the interrupt, then returns control to the computer so it may resume program execution.

interrupt vector table: A table maintained in memory by ProDOS 16 that contains the addresses of all currently active (allocated) interrupt handlers.

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

I/O expansion slots: The seven rectangular connectors located on Apple IIGS main logic board. These slots will accept standard Apple II peripheral cards and allow the computer to communicate with peripherals such as printers and disk drives. See also **peripheral card.**

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

joystick: A type of hand control that has a movable stick to provide x-axis and y-axis inputs to the computer.

jump table: A table constructed in memory by the System Loader from all jump-table segments encountered during a load. The jump table contains all references to dynamic segments that may be called during execution of the program.

K: Abbreviation for the prefix *kilo-,* meaning 1024. A kilobyte (expressed 1K) of memory is 1024 memory locations.

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

kilobyte: 1024 bytes, usually used to describe a range or size of memory. Compare **kilobit.**

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

language card: Memory with addresses between $D000 and $FFFF on any Apple II–family computer. It includes two RAM banks in the $Dxxx space, called *bank-switched memory.* The language card was originally a peripheral card for the 48K Apple II or Apple II Plus that expanded its memory capacity to 64K and provided space for an additional dialect of BASIC.

local area network: A high-speed data communication channel that provides connections between computers, disk drives, printers, and other peripherals in a limited geographic area, such as within a single building or campus.

long word: A double-length word. For the Apple IIGS, a long word is 32 bits (4 bytes) long.

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.

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

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: Referring to the least significant part of a numerical quantity. In normal representation, the low-order bit of a binary number is in the rightmost position; likewise, the low-order byte of a binary word or long word consists of the rightmost 8 bits.

main logic board: The main circuit board in the computer, which holds the major electronic components and connectors.

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

megabyte: 1,048,576 bytes or 1024 kilobytes, usually used to describe a range or size of memory. Compare **megabit.**

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

Mega II: A custom, large-scale integrated circuit that incorporates most of the timing and control circuits of the standard Apple II; an Apple II on a chip. 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).

memory: Locations within a computer or other electronic device that retain or "remember" data as needed by the microprocessor. Two types of memory are utilized in the Apple II computers: random-access (or read/write) memory (RAM), and read-only memory (ROM).

memory expansion card: A slot card that contains additional RAM and ROM memory. In the Apple IIGS, this optional expansion card is to be used only in the memory expansion slot. Memory expansion cards for use in the Apple IIe are not to be used in this computer.

memory expansion slot: The single slot located on the Apple IIGS main logic board that accepts an Apple IIGS memory expansion card. Memory expansion cards designed for other Apple II computers will not work in this slot.

Memory Manager: A program in the Apple IIGS Toolbox that manages memory use. The Memory Manager keeps track of how much memory is available, and allocates memory blocks to hold program segments or data.

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.

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 also **e flag, x flag.**

microcomputer: Any small computer whose central processing element is contained on a single small circuit board or within a single integrated circuit.

microprocessor: The heart of a microcomputer. Usually, a single-chip computer element that contains the control unit, central processing circuitry, and arithmetic and logic functions and is suitable for use as the central processing unit of a microcomputer or a dedicated automatic control system. In the Apple IIGS computer, the microprocessor is the 65C816. Previous Apple II computers utilize the 6502 and 65C02 microprocessors. Some microprocessors used in other computers are the 68000, the 8080, the Z80, and the 8086.

modem: Acronym for *modulator-demodulator.* A computer peripheral device that allows computers to transfer digital information over conventional telephone lines. Modems usually connect to the computer's serial port, but may instead plug into a peripheral expansion slot.

monitor: See **video monitor.**

Monitor program: A 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.

monochrome: Displaying video in one color and the background in another, frequently black and white, but not necessarily. The Apple IIGS monochrome default is white characters on a medium blue background.

MOS: Acronym for *metal oxide semiconductor,* 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 within graphics programs.

move: To change the location of a group of data bytes in memory. The Memory Manager may move blocks to consolidate memory space.

native mode: The 16-bit operating configuration of the 65C816 processor.

NMOS: One of several methods of making integrated circuits out of silicon; a metal-oxide semiconductor device made on a p-type substrate using n-type source and drain contacts.

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 video* because it combines all the video information, including color, into a single signal.

null: Zero.

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 keyboards. On the Apple IIGS keyboard, the equivalent key is called the *Command key;* it is marked with both an Apple icon and a propeller, the icon used on some Macintosh keyboards.

operand: (1) In assembly language, the part of an instruction that follows the operation code. The operand is used as a value or an address, or to calculate a value or an address. (2) In object module format, an operation code that is followed by a single value that constitutes part of an expression. The value following the operand opcode is acted on by an operator.

operating system: A general-purpose program that organizes the actions of the various parts of the computer and its peripheral devices. See also **disk operating system.**

operation code: The part of a machine-language instruction that specifies the operation to be performed. Often called *opcode.*

operator: In object module format, an operation code that specifies an arithmetic or logical operation in an expression to be performed on one or two variables that precede it. The variables acted on by an operator are identified by operand opcodes that precede them.

oscillator: Traditionally, an electronic circuit that generates a constant or variable frequency signal. As used in the **Digital Oscillator Chip** in the Apple IIGS, the oscillators are actually address generators, pointing to data bytes in sound memory.

page: (1) A portion of memory 256 bytes long and beginning at an address that is an even multiple of 256. Memory blocks whose starting addresses are an even multiple of 256 are said to be *page-aligned.* (2) (Usually capitalized.) An area of main memory containing text or graphics information being displayed on the screen.

PAL: Acronym for *phase alternating lines.* A video standard originated in England and used in other European countries.

parallel: (1) The simultaneous occurrence of more than one phenomenon. (2) The connection of a group of wires for the purpose of conducting bits of data simultaneously, rather than one at a time (via a serial connection).

peripheral card: A removable printed-circuit board that plugs into one of the seven I/O expansion slots, allowing the computer to use a peripheral device or to perform some subsidiary or peripheral function. These cards may be self-contained (such as a clock card) or an interface card (such as a disk interface card) with a cable connecting the card and the peripheral.

peripheral device: An input or output (or input/output) device, usually external to the computer (but which may reside on a card in a peripheral-expansion slot), that performs some secondary function for the computer. Printers, disk drives, modems, and video monitors are examples of peripheral devices.

peripheral expansion slot: The seven slots at the rear of the main logic board that will accept most Apple II peripheral expansion cards. Each slot has 50 pins, which carry required power and signals to and from the peripheral cards.

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 ø0 and ø1. (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 bit map. In the Super Hi-Res graphics display on the Apple IIGS, each pixel is represented by either two or four bits; the display is not a bit map, but rather a pixel map.

pixel map: A set of values that represents the positions and states of the set of **pixels** making up an image. Compare **bit map.**

pointer: An item of information consisting of the memory address of some other item. For example, the 65C816 stack register contains a pointer to the top of the stack.

pop: See **pull.**

power supply: The large metal box inside the Apple IIGS that divides and conditions the household current, supplying the voltages required by the main logic board and some peripheral devices.

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

ProDOS 8: A disk operating system developed for standard Apple II computers. It runs on 6502-series microprocessors. It also runs on the Apple IIGS when the 65C816 processor is in 6502 emulation mode.

ProDOS 16: A disk operating system developed for 65C816 native mode operation on the Apple IIGS. It is functionally similar to ProDOS 8 but more powerful.

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

PROM: See **programmable read-only memory**.

protocol: An agreed-upon formal hardware or software handshaking between two electronic devices (usually a computer and a peripheral I/O device).

Pull: To remove the top entry from a stack, this instruction moves the stack pointer to the entry below it. Synonymous with *pop*. Compare **push.**

Push: To place a new entry at the top of the stack, this instruction moves the stack pointer to the entry above it. Compare **pull.**

radio frequency (RF): Broadcast frequency over which radio and television operate. Generally defined as the radio spectrum between 3 MHz and 3000 MHz.

radio-frequency modulator: A device used to raise video signals to a frequency that may be received and displayed by a television, as a substitute for a standard video monitor when one is not available.

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

RAM disk: A portion of memory (RAM) that appears to the operating system to be a disk volume. Files in a RAM disk can be accessed much faster than the same files on a floppy disk or hard disk.

random-access device: See **block device.**

random-access memory (RAM): Volatile memory within a computer or other electronic device. Data are retained as long as power is supplied to the memory chip. Once the power is disconnected, the data are lost. Compare **read-only memory.**

read-only memory (ROM): Nonvolatile, permanent memory. ROM ICs may be written once, usually in the development of the computer. Data are retained in the memory even after power is disconnected. Special ROM ICs allow you to change the data in them under specific conditions such as ultraviolet light (EPROMs [erasable programmable read-only memory]), or high voltages (EEPROMs [electrically erasable programmable read-only memory]). Normally, however, ROM ICs are written once.

read/write memory: See **random-access memory.**

real-time clock (RTC): A custom IC that, once set, retains the current time of day, day, month, and year. Chapter 7 provides details of the RTC and other built-in I/O devices.

RF: See **radio frequency.**

RF modulator: See **radio-frequency modulator.**

RGB: Abbreviation for *red, green,* and *blue.* A method of displaying color video by transmitting these three 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. The Apple IIGS uses analog RGB; connect only RGB monitors using analog RGB to the RGB video connector at the rear of the computer. Compare **composite video.**

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

ROM disk: A feature of some operating systems making it possible to use read-only memory (ROM) as a disk volume. Often used for making applications permanently resident. See also **RAM disk.**

RS-232-C: 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-C devices.

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

SCC: See **Serial Communications Controller.**

schematic diagram: A diagram using special figures to represent ICs, logic functions, and interconnecting wires, to describe a circuit. Schematic diagrams for the Apple IIGS main logic board are located in the Addendum.

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 $0400 to $07FF; the screen holes are locations in that area that are neither displayed nor modified by the display firmware.

SECAM: A French acronym meaning "sequential color with memory." A video standard originating in France and used in the USSR and other countries.

sector: A division of a track on a disk. When a disk is formatted, its surface is divided into tracks and sectors.

semiconductor: A class of materials whose resistivity lies between that of conductors and insulators, for example, germanium and silicon. Solid-state electronics is based on the use of semiconductors.

serial: A single-wire connection for the purpose of transferring bits of data one at a time, usually between a computer and a peripheral device. Compare **parallel.**

Serial Communications Controller (SCC): A type of communications IC used in the Apple IIGS. The SCC can run synchronous data transmission protocol and thus transmit data at faster rates than the ACIA. Compare **ACIA.**

serial port: The two connectors located at the back of the Apple IIGS main logic board that provide a means for communicating with peripherals (such as printers and local area networks) using a serial interface.

shadowing: The process whereby any changes made to one part of the Apple IIGS memory are automatically and simultaneously copied into another part. When shadowing is on, information written to bank $00 or $01 is automatically copied into equivalent locations in bank $E0 or $E1. Likewise, any changes to bank $E0 or $E1 are immediately reflected in bank $00 or $01.

silicon: The semiconductor used in a majority of solid-state electronic components and integrated circuits.

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

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

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

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

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.

SmartPort: A set of firmware routines supporting multiple block devices connected to the Apple IIGS disk port.

soft switch: A location in memory that produces some specific effect whenever its contents are read or written.

software: A group of instructions to the microprocessor, instructing it to perform certain functions, such as performing computations, displaying data on a monitor, reading data from and writing data to a disk. The group of instructions is known collectively as a program. Compare **application program.**

special memory: On an Apple IIGS, all of banks $00 and $01, and all display memory in banks $E0 and $E1. So called because it is the memory directly accessed by standard Apple II programs running on the Apple IIGS.

stack: A list in which entries are added (pushed) and removed (pulled) at one end only (the top of the stack), causing them to be removed in last-in, first-out (LIFO) order. The term *the stack* usually refers to the particular stack pointed to by the 65C816's stack register.

stack register: A hardware register in the 65C816 processor that contains the address of the top of the processor's stack.

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.

start up: To get the system running. Starting up involves loading system software from disk, and then loading and running an application. Also called **boot.**

Super Hi-Res: A high-resolution graphics display mode on the Apple IIGS, consisting of an array of points 320 wide by 200 high with 16 colors or 640 wide by 200 high with 16 colors (with restrictions).

sync signal: A signal that exists for the purpose of synchronizing two devices. Frequently generated by a video generator and used by a video monitor to synchronize the video display to the separate video information. In the Apple IIGS, the sync signal is mixed with the video information resulting in the composite video signal.

synthesizer: A hardware device capable of producing sound by creating it digitally, and then converting it into an analog waveform that you can hear.

system disk: A disk that contains the operating system and other system software needed to run applications.

system software: The components of a computer system that support application programs by managing system resources such as memory and I/O devices.

text window: That portion of the screen that is reserved for text. After starting the computer, the firmware uses the entire display for text. However, if you wish, you can restrict the text video activity to any rectangular portion of the display.

tool: See **tool set.**

toolbox: See "*Apple IIGS Toolbox.*"

tool set: A group of related routines (usually in firmware), available to applications and system software, that perform necessary functions or provide programming convenience. The Memory Manager, the System Loader, and QuickDraw II are tool sets.

track: One of a series of concentric circles on a disk. When a disk is formatted, its surface is divided into tracks and sectors.

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. Stands for *transistor-transistor logic, red, green, blue.* Compare **analog RGB.**

unbuffered: A style of input and output that does not use a buffer for I/O; reading and writing is done one character at a time.

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.

VGC: See **Video Graphics Controller.**

video: An electrical signal containing information that may be obtained visually when displayed on a video monitor. Information organized or transmitted in video form. See also **NTSC, PAL, SECAM.**

Video Graphics Controller (VGC): The custom IC on the Apple IIGS main logic board responsible for generating all video used in the Apple IIGS.

video monitor: A display device that receives video signals by direct connection only.

wavetable: A group of data bytes in memory used as data by the DOC to generate sound. The wavetable is built by using the DOC to digitize an analog-input signal and placing the resulting data bytes in sound RAM memory.

word: A group of bits that is treated as a unit. For the Apple IIGS, a word is 16 bits (2 bytes) long.

write-only memory: A form of computer memory into which information can be stored but never, ever retrieved.

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.**

XOFF: A special character (ASCII value $11) used for controlling the transfer of data between a microcomputer and a serial peripheral device. When one piece of equipment receives an XOFF character from the other, it stops transmitting characters until it receives an XON. See **handshaking, XON.**

XON: A special character (ASCII value $13) used for controlling the transfer of data between a microcomputer and a serial peripheral device. See **handshaking, XOFF.**

zero page: The first page (256 bytes) of memory in a standard Apple II computer (or in the Apple IIGS computer when running a standard Apple II program). Because the high-order byte of any address in this part of memory is zero, only a single byte is needed to specify a zero-page address. Compare **direct page.**

Index

Addendum **Schematic Diagrams**

This Addendum contains schematic diagrams for the main circuit board of the original Apple IIGS and the 1 MB Apple IIGS.

THE APPLE PUBLISHING SYSTEM

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

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

> $26.95 FPT
USA

The Official Publication from Apple Computer, Inc.

Apple IIGS® Hardware Reference, second edition

Written and produced by the Developer Technical Publications group at Apple Computer, this reference is for software developers and hardware developers creating products for the Apple IIGS and the 1MB Apple IIGS.

The *Apple IIGS Hardware Reference*, second edition, provides an extensive description of all aspects of internal operation and presents the latest information on the machine's hardware—including the interface requirements of input devices, video display, disk drives, and serial ports. This information is essential to the hardware designers and programmers writing drivers for peripheral devices. The *Apple IIGS Hardware Reference* is the definitive reference for these particular Apple II computers.

The manual describes

- the architecture of both the Apple IIGS and 1 MB Apple IIGS computers
- memory organization, including standard RAM (256K in the Apple IIGS and 1 MB in the 1 MB Apple IIGS), standard ROM (128K in the Apple IIGS and 256K in the 1 MB Apple IIGS, and up to 8 MB of expansion RAM
- the video display modes, including Super Hi-Res graphics mode
- the sound capabilities of the computers, including the 15-voice synthesizer
- the Apple Desktop Bus™ for connecting keyboards and other input devices (including the new keyboard mouse and sticky keys support for disabled users)
- the built-in input/output ports and real-time clock
- the seven input/output expansion slots for connecting peripheral devices, including timing parameters and diagrams
- the sixteen-bit 65C816 microprocessor

The appendixes contain supplementary tables and frequently used information about the Apple IIGS computers. Also included is an addendum containing schematic diagrams for both versions of the computer.

Printed in U.S.A.

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
408 996 1010
TLX 171 576

Addison-Wesley Publishing Company, Inc.

ISBN 0-201-52389-2