APPLE
PROGRAMMER'S
AND DEVELOPER'S
ASSOCIATION

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

# Apple IIGS Hardware Reference Manual

**APDA Draft**
**21 Nov., 1986**

APDA#: K2SHWR

# Apple IIGS Hardware Reference

**APDA Draft**

21 November, 1986

Apple Technical Publications

*This document contains preliminary material. It does not include*

- *final editorial corrections*
- *final artwork*
- *an index*

*It may not include final technical changes.*

® APPLE COMPUTER, INC.

*Apple IIGS Hardware Reference*

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

Simultaneously published in the United States and Canada.

# Contents

# List of Illustrations and Tables

## List of Ilustrations

## List of tables

Chapter 4, *Video*, goes in depth into how to make the new Super Hi-Res graphics work for you.

Chapter 5, *Peripheral Expansion*, provides descriptions of the I/O slots and the signals available at an expansion slot. DMA and interrupts are described here also.

Chapter 6, *Sound*, shows you how to control the 32 digital oscillators and generate sound.

Chapter 7, *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 device.

The built-in disk drive port is described in Chapter 8.

Chapter 9 goes into detailed description of the memory expansion slot and how to design and access a memory expansion card for this special slot.

Chapter 10 briefly describes the Apple IIGS power supply and lists its specifications.

The new 65C816 microprocessor is covered in Chapter 11.

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 all nine international keyboard layouts.

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

Appendix D contains the schematic diagrams showing all of the electrical components of the main circuit board.

Appendix E has tables that show what a bit and a byte can represent. Conversion tables between hexadecimal, decimal, and negative decimal, as well as 8-bit ASCII, are provided.

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

A glossary follows the appendixes.

## Some terminology

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

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

# 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 processor 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 a photograph of the Apple IIGS.

PHOTOGRAPH OF THE APPLE IIGS

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

## Removing the cover

The Apple IIGS uses a 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.

PHOTO OF SNAP-LOCKS AT THE REAR OF THE COVER.

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

AERIAL PHOTO OF APPLE IIGS, COVER REMOVED. CALLOUTS.

**Figure 1-3**. The Apple IIGS with cover removed

## Peripheral expansion slots

The Apple IIGS, like the Apple IIe, has seven **expansion slots** at the rear of the **main logic board**. These 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 5, "Peripheral 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.

PHOTO OF REAR PANEL OF APPLE IIGS WITH CALLOUTS

**Figure 1-4**. The Apple IIGS connectors

**Two 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**). Figure 1-5 shows the pin configuration of the serial ports.

To read about how to use the firmware in the Apple IIGS ROM, refer to the Apple IIGS *Firmware Reference* manual.

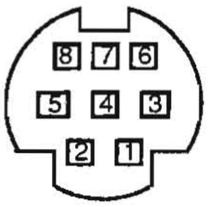


**Figure 1-5.** Pin configuration of a serial port connector

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

**Table 1-1.** Serial port signal description

**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 8, "The Disk Port".

**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, "Video," for more information.

**Composite video connector: Composite** color **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, "Video" 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 7, "Apple DeskTop Bus," for more information on using this connector.

**Game connector:** Connect a standard Apple II game paddle 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 2 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 first subsystem 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 peripheral devices and external I/O slots

This subsystem is referred to as the Mega II portion of the system, after the controlling device.

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

The second subsystem consists of components of the computer that are new to the Apple II family. These are

- the 65C816 microprocessor
- the Fast Processor Interface (FPI) custom IC
- 256K (expandable to 8Mb) of dynamic **random-access memory (RAM)**
- 128K (expandable to 1Mb) of **read-only memory (ROM)**

This subsystem is referred to as the FPI portion of the system, because of the controlling device, the FPI.

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 1-6 shows the Apple IIGS computer in block form. Note the dotted division 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.

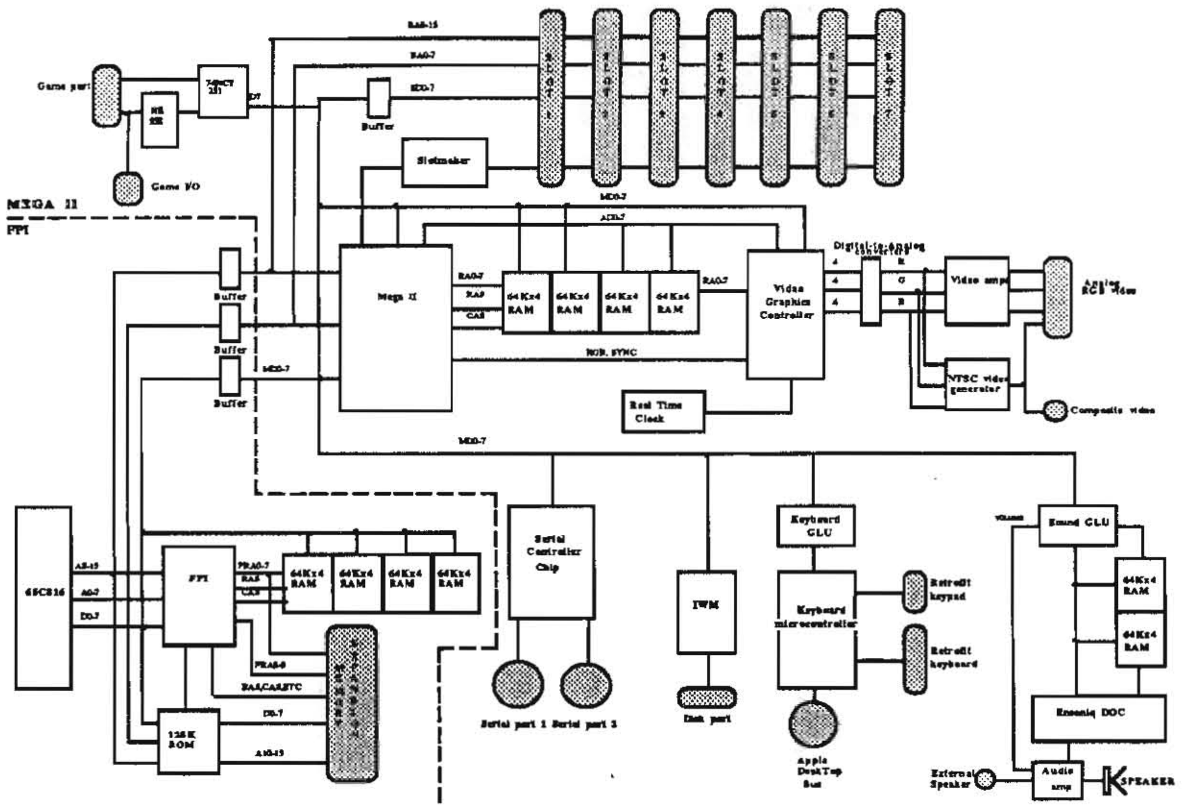


**Figure 1-6**. Apple IIGS system block diagram

The rest of this chapter describes the FPI and Mega II subsystems in more detail and explains how they function together.

## Apple II compatibility

The Apple IIGS is compatible with the Apple II family of processors. Here are some of the features the Apple IIGS shares with the Apple IIe and IIc:

- 6502 processor compatibility, which is maintained by the 65C816 microprocessor used by the Apple IIGS
- Apple II graphics, which includes **Lo-Res** mode, **Hi-Res** mode, and **Double Hi-Res** mode color video graphics
- 128K of main RAM memory
- built-in Applesoft BASIC
- two built-in serial ports
- seven peripheral expansion slots, compatible with the IIe
- a built-in disk interface port that will accept either 5.25-inch or 3.5-inch disk drives

- built-in Apple II Monitor firmware
- 40-column and 80-column text display capability
- a game I/O port for joysticks and game controllers like the IIe

## New 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.

**Complementary Metal Oxide Semiconductor (CMOS)** is the silicon material that this microprocessor is made from. This material allows the device to be faster and to require less power.

To learn more about the 6502 and the 65C02 microprocessors, refer to the *Apple IIe Technical Reference* manual and the *Apple IIc Technical Reference* manual, respectively.

- high processing speed, which is selectable between 1.024 MHz or 2.8 MHz
- Super Hi-Res video graphics mode, which offers 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
- 128K of RAM, which may be expanded to a maximum of 8Mb that can be achieved by using an optional expansion card in the memory expansion slot; a maximum of 1Mb of ROM can be utilized by using an expansion card
- 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 IC with 32 independant oscillators and 64K of dedicated RAM
- a detachable, full 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

- a control panel screen, which provides users with means for setting system parameters

## Terminology

The terms *Mega II subsystem* and *Apple II-side* refer to the portion of the Apple IIgs that provides the Apple II compatibility. The terms *FPI subsystem* and *16-bit side* refer to the portion of the computer system that provides those features which are new to the Apple II family: Everything that makes the Apple IIGS not an Apple IIe or IIc.

**Important:** Throughout this manual you will need to manipulate bits within registers and soft switch locations in order to achieve some 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." In order to manipulate the desired bits 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 allow you to modify any one bit and leave the others untouched.

To read about using the TSB and TRB instructions, refer to "Programming the 65816" by David Eyes.

## Memory allocation

Note that the Apple IIGS has three separate quantities of RAM: 256K available to the FPI; 128K (half of the on-board RAM) available to the Mega II; 64K dedicated to the DOC. Figure 1-7 shows the system memory map.

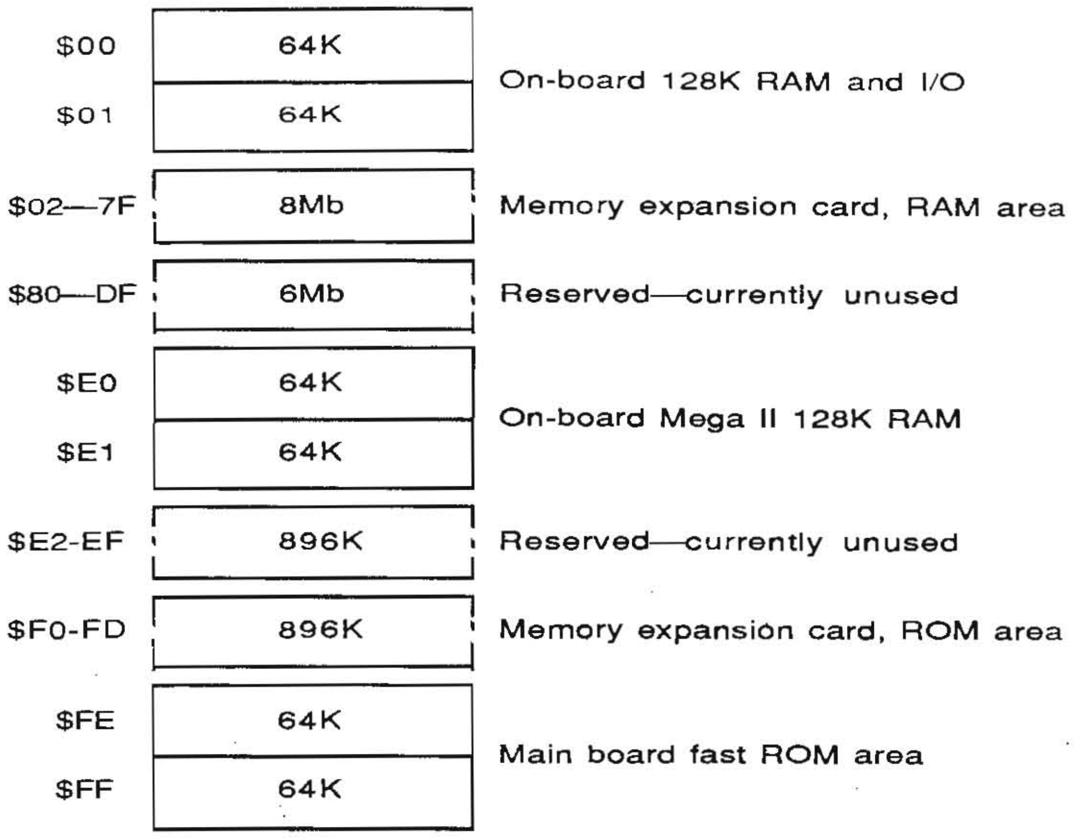

$00
64K
$01
64K
On-board 128K RAM and I/O
$02—7F
8Mb
Memory expansion card, RAM area
$80—DF
6Mb
Reserved—currently unused
$E0
64K
$E1
64K
On-board Mega II 128K RAM
$E2-EF
896K
Reserved—currently unused
$F0-FD
896K
Memory expansion card, ROM area
$FE
64K
$FF
64K
Main board fast ROM area

**Figure 1-7.** Bank memory map

A minimum Apple IIGS system includes 256K of RAM and 128K of ROM. The 128K ROM space is expandable to 1 megabyte.

*Note*: 128K of the main RAM (banks $00 and $01) correspond to the main 64K and auxiliary 64K memory banks of the Apple IIe and Apple IIc with some exceptions: they lack the standard Apple II language-card space, I/O space, and video display buffers. To provide these memory spaces, the Apple IIGS *shadows*, or duplicates, accesses to the language-card, I/O, and video locations in banks $00 and $01 into the equivalent locations in banks $E0 and $E1. For more information on how shadowing works, see Chapter 3, "New Features."

# Chapter 2

# Maintaining Apple II Compatibility

The Apple IIGS maintains compatibility with the rest of the Apple II computers by virtue of the Mega II custom IC. This chip contains most of the components from the Apple IIe (and many from the IIc) and makes it possible for the Apple IIGS to run application programs written for the IIe.

This chapter describes the function of the Mega II IC and therefore the Apple IIe. Although each topic (video, I/O, memory, and so forth) can also be found elsewhere in this manual, only those standard Apple II-compatible features are covered in this chapter. To read about the new features of the Apple IIGS, see the chapter for each feature.

This chapter also contains detailed descriptions of all of the hardware and firmware that make up the Apple IIe, and therefore, all that makes the Apple IIGS compatible with the Apple IIe. Table 2-1 lists come comparisons of the features found on the Apple IIGS, Apple IIe, and the Apple IIc.

This chapter contains a lot of information about the way the Apple IIGS works, but it doesn't tell you how to use a Apple IIGS. For that information, you should read the other Apple IIGS manuals, especially the Apple IIGS *Owner's Guide*.

To read about the new Apple IIGS sound synthesizing capability, see Chapter 6, "Apple IIGS Sound."

To read about the new Apple IIGS Super Hi-Res graphics, see Chapter 4, "Video."

**Table 2-1.** Apple II-compatible features

| **Feature** | **Apple IIe** | **Apple IIc** | **Apple IIGS** |
|---|---|---|---|
| Memory | 64K | 128K | 256K* |
| Serial ports | Expansion card only | 2 built in | 2 built in |
| Disk port | Expansion card only | 1 built in | 1 built in |
| Text display | 40-column (80 optional) | 40- and 80-column | 40- and 80-column |
| Graphics | Standard Apple II graphics modes | Standard Apple II graphics modes | Standard and new graphics modes |
| Keyboard | Built-in | Built in | Detachable input bus device |
| Peripheral expansion slots | Seven | none | seven |
| Memory expansion slot | One, for use with with 80-column card/ | none | one, for use with memory expansion card |

*while the Apple IIGS has 256K on-board memory, programs written for previous Apple II models will utilize a maximum of 128K.

## The Mega II Custom IC

The Mega II is a custom integrated circuit made up of several circuits previously found in the Apple IIe. The following make up the Mega II:

- memory management unit (MMU) custom IC
- input/output unit (IOU) custom IC
- character generator ROMs (eight character sets)
- video circuitry

The Mega II has virtually all the functions of an Apple IIe on a chip; it supports a slotted microcomputer architecture as well as the new peripheral devices built into the Apple IIGS. Apple II system components not found on the Mega II are the microprocessor, RAM and ROM memory, the slots, and the 16-pin game I/O connector. Figure 2-1 shows the Mega II and its relationship to the other parts of the Apple IIGS.

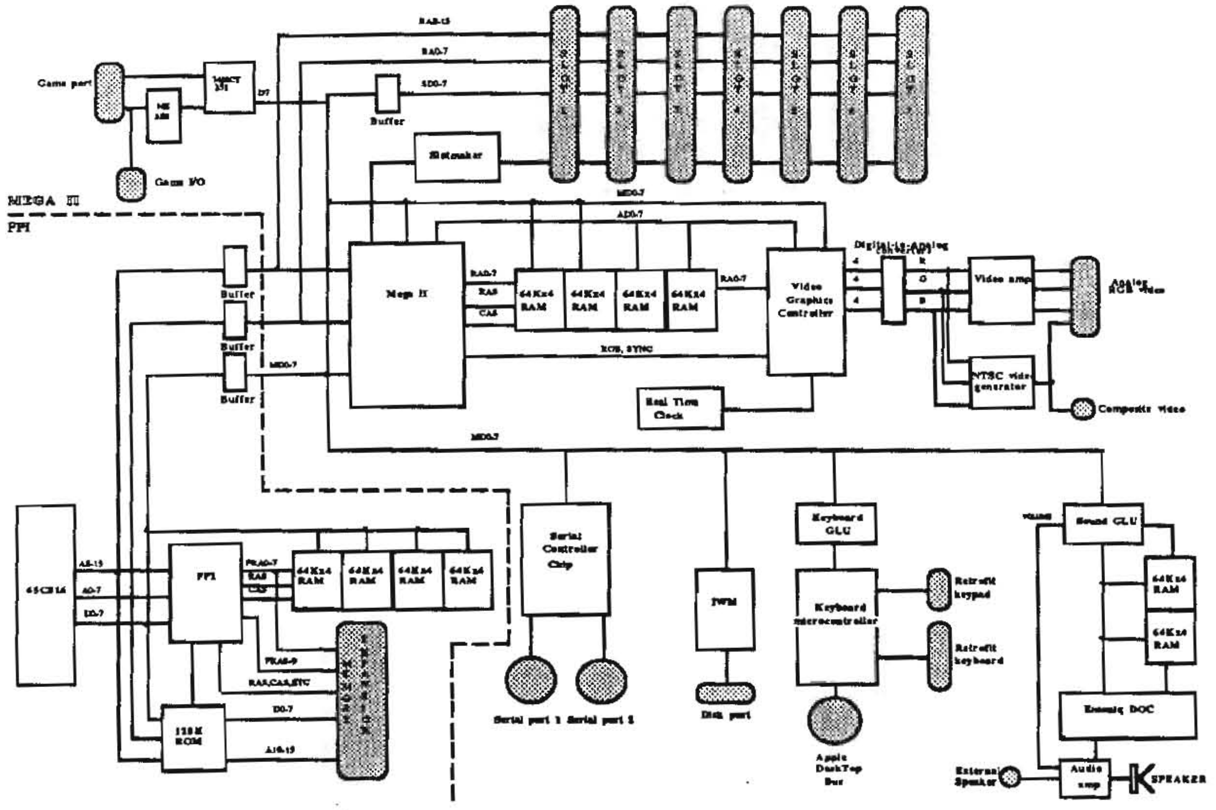


Figure 2-1. Relationship of the Mega II within the Apple IIGS

## The Apple IIe

- The Apple IIGS emulates many features of the Apple IIe:
- the 6502 microprocessor running at 1.024 MHz
- 128K of RAM
- 40-column text and 80-column text display
- video graphics, including:
  - Lo-Res graphics
  - Hi-Res graphics
  - Double Hi-Res graphics
- peripheral card I/O including direct memory access (DMA)

Although the Apple IIGS is capable of running the processor at a higher clock speed, some standard Apple II application programs must be executed at the 1.024MHz clock speed for timing reasons.

## RAM upgrade

The Apple IIe is a 64K machine, expandable to 128K through the use of auxiliary memory cards like the **Extended 80-Column Text Card**. The Apple IIGS has 256K of **main memory**, mounted on the circuit board.

The RAM ICs on the IIe circuit board have been replaced by ten 64K x4 ICs on the IIGS main logic board. This memory is divided into the 128K accessable by the Mega II in which standard Apple II programs are run, and 128K of fast system memory available for programs that were developed for the Apple IIGS. Also note that there is 64K of RAM dedicated to sound generation that is not directly accessable by application programs.

## Apple II I/O

This section describes the I/O devices built into the Apple IIGS in terms of their functions and the way they are used by programs. The built-in I/O devices are:

- the keyboard
- the video-display generator
- the speaker
- the game input and output

At the lowest level, programs use the built-in I/O devices by reading and writing to dedicated memory locations. This section lists these locations for each I/O device. It also gives the locations of the internal soft-switches that select the different display modes of the IIGS.

❖ *Built-in I/O routines:* This method of input and output—loading and storing directly to specific locations in memory—is not the only method you can use. For many of your programs, it may be more convenient to call the built-in I/O routines stored in the Apple IIGS firmware.

To read more about built-in I/O routines in the Apple IIGS, see the *Apple IIGS Firmware Reference* manual.

### The keyboard

The keyboard uses the Apple DeskTop Bus (ADB) 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. You may also hold down any number of keys and still press another key; this is known as N-key rollover.

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

## Reading the keyboard

The keyboard encoder and ROM generate all 128 **ASCII** (American Standard Code for Information Interchange) codes, so all the special character codes in the ASCII character set are available from the keyboard. Application programs obtain character codes from the keyboard by reading a byte from the keyboard data location shown in Table 2-2.

**Table 2-2**
Keyboard memory locations

| Location | | |
|---|---|---|
| **Hex** | **Decimal** | **Description** |
| $C000 | 49152 -16384 | Keyboard data and **strobe** |
| $C010 | 49168 -16368 | Any-key-down flag and clear-strobe switch |

Your programs can get the code for the last key pressed by reading the keyboard data location. Table 2-2 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 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, Apple, 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 byte. After any key has been pressed, the strobe bit is high. It remains high until you reset it by reading or writing at the clear-strobe location. This location is a combination flag and switch; the flag tells whether any key is down, and the switch clears the strobe bit. The switch function of this memory location is called a **soft switch** because it is controlled by software. 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. Similar soft switches, described later, are used for controlling other functions in the Apple computer.

See the *Apple IIGS Firmware Reference* manual for information on firmware for reading the keyboard.

**Important** 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 C 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, the state of these modifier keys is available by reading a register within the ADB microcontroller.

To learn how to read the registers within the ADB microcontroller, see Chapter 7, "Apple DeskTop Bus."

The Control-Apple-Reset key combination is different from all other keys on the ADB keyboard only in that it generates a special key code. When the ADB microcontroller detects the reset code, the program currently running in memory is halted. If the Control-Reset-Esc key combination are pressed, the system halts whatever program it's running, 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* manual.

To read more about the ADB, see Chapter 8, "Apple DeskTop Bus."

## Apple II video

The Apple IIGS 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 graphics mode, Super Hi-Res graphics, is discussed in Chapter 4, "Video."

## Video output

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 graphics modes, you can use a television set for your video display. If the TV 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.

**Important** 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 2-3.

The video signal produced by the Apple IIGS is NTSC-compatible composite color video. It is available at two places: 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. To read about Super Hi-Res graphics, see Chaper 4, "Video."

**Table 2-3**
Standard Apple II Video display specifications

| | |
|---|---|
| Display modes: | 40-column text; map: Figure 2-5<br>80-column text; map: Figure 2-6<br>Low-Res color graphics; map: Figure 2-7<br>Hi-Res color graphics; map: Figure 2-8<br>Double Hi-Res color graphics; map: Figure 2-9 |
| 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 2-4) |
| Lo-Res color graphics: | 16 colors (Table 2-5) 40 horizontal by 48 vertical; map: Figure 2-7 |
| Hi-Res color graphics: | 6 colors (Table 2-6) 140 horizontal by 192 vertical (restricted);<br>Black and white: 280 horizontal by 192 vertical; map: Figure 2-8 |
| Double Hi-Res color graphics: | 16 colors (Table 2-7) 140 horizontal by 192 vertical (no restrictions);<br>Black and white: 560 horizontal by 192 vertical; map: Figure 2-8 |

The Apple IIGS can produce seven different kinds of standard Apple II video display:

- text, 24 lines of 40 characters
- text, 24 lines of 80 characters
- Lo-Res graphics, 40 by 48, in 16 colors
- Hi-Res graphics, 140 by 192, in 6 colors
- Hi-Res graphics, 280 by 192, in black and white
- Double Hi-Res graphics, 140 by 192, in 16 colors
- Double Hi-Res graphics, 560 by 192, in black and white

The two text modes can display all 128 ASCII **characters**: uppercase and lowercase letters, numbers, and symbols. The Apple IIGS can also display MouseText characters.

Any of the graphics displays can have 4 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. In mixed mode, the 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 dots, 280 wide by 192 high. There are 6 colors available in Hi-Res displays, but a given dot can use only 4 of the 6 colors. If color is used, the display is 140 dots wide by 192 high. In mixed mode, the 4 lines of text replace the bottom 32 rows of dots, leaving 160 rows of 280 dots each.

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

## Text modes

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

The normal display has white dots on a medium blue background. (Other color text on other color backgrounds is also possible. See Chapter 4, "Video," for more information.) Characters can also be displayed as blue dots on a white background; this is called *inverse format.*

### Text character sets

The Apple IIGS can display either of two 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 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 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 set by means of the alternate-text soft switch, SETALTCHAR, described later in this chapter in the section "Display Mode Switching." Table 2-4 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 2-4**
Display character sets

| **Hex** | **Primary character set** | | **Alternate character set** | |
|---|---|---|---|---|
| **Values** | **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 dots 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 2-2 and 2-3. 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.

```
]LIST 0,100
10  REM  APPLESOFT CHARACTER DEMO
20  TEXT : HOME
30  PRINT : PRINT "Applesoft character Demo"
40  PRINT : PRINT "Which character set--"
50  PRINT : INPUT "Primary (P) or Alternate (A) ?";A$
60  IF  LEN (A$) < 1 THEN 50
65  LET A$ =  LEFT$ (A$,1)
70  IF A$ = "P" THEN  POKE 49166, 0
80  IF A$ = "A"THEN  POKE 49167, 0
90  PRINT : PRINT "...printing the same line, first"
100 PRINT " in NORMAL, then INVERSE ,then FLASH:": PRINT
]
```

**Figure 2-2**
40-column text display

```
]LIST
10 REM APPLESOFT CHARACTER DEMO
20 TEXT : HOME
30 PRINT : PRINT "Applesoft Character Demo"
40 PRINT : PRINT "Which character set--"
50 PRINT : INPUT "Primary (P) or Alternate (A) ?";A$
60 IF  LEN (A$) < 1 THEN 50
65 LET A$ = LEFT$ (A$,1)
70 IF A$ = "P" THEN POKE 49166,0
80 IF A$ = "A" THEN POKE 49167,0
90 PRINT : PRINT "printing the same line, first"
100 PRINT " in NORMAL, then INVERSE ,then FLASH:": PRINT
150 NORMAL : GOSUB 1000
160 INVERSE : GOSUB 1000
170 FLASH : GOSUB 1000
180 NORMAL : PRINT : PRINT : PRINT "Press any key to repeat."
190 GET A$
200 GOTO 10
1000 PRINT : PRINT "SAMPLE TEXT: Now is the time--12:00"
1100 RETURN
]
```

**Figure 2-3**
80-column text display

## Graphics modes

The Apple IIGS can produce standard Apple II video graphics in three different modes. 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.

### 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 3 shades of gray. There are no blank dots 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 2 Lo-Res graphics blocks. The 2 blocks are displayed one atop the other in a display space the same size as a 40-column text character, 7 dots wide by 8 dots 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 1 of 16 colors appears on the screen. The colors and their corresponding nibble values are shown in Table 2-5. 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 2-5**
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 TV set.

As explained later 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 later 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.

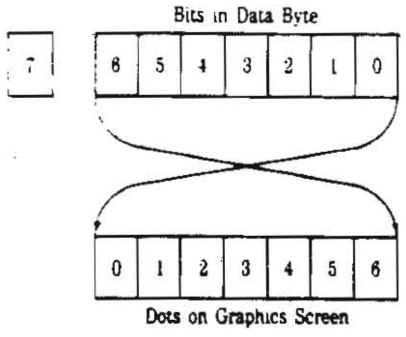
## Hi-Res graphics

In the Hi-Res graphics mode, the Apple IIGS displays an array of colored dots in 192 rows and 280 columns. The colors available are black, white, purple, green, orange, and blue, although the colors of the individual dots are limited, as described later in this section. Adjacent dots of the same color merge to form a larger colored area.

Data for the Hi-Res graphics displays are stored in either of two 8192-byte areas in memory. These areas are called *Hi-Res Page 1* and *Page 2;* think of them as buffers where you can put data to be displayed. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes to display; this section describes the way the resulting graphics data are stored in memory.

The Hi-Res graphics display is bit-mapped: each dot on the screen corresponds to a bit in memory. The 7 low-order bits of each display byte control a row of 7 adjacent dots on the screen, and 40 adjacent bytes in memory control a row of 280 (7 times 40) dots. The least significant bit of each byte is displayed as the leftmost dot in a row of 7, followed by the second least significant bit, and so on, as shown in Figure 2-4. The eighth bit (the most significant) of each byte is not displayed; it selects 1 of 2 color sets, as described later.

**Figure 2-4**
Hi-Res display bits



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

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

Call the leftmost column of dots 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 dots in even-numbered columns (0, 2, 4, and so forth) are on, the dots are purple; if the bits that control odd-numbered columns are on, the dots are green—but only if the dots on both sides of a given dot are black. If two adjacent dots are both on, they are both white.

You select the other two colors, blue and orange, by turning the high-order bit (bit 7) of a data byte on (1). The colored dots controlled by a byte with the high-order bit on are either blue or orange: The dots in even-numbered columns are blue, and the dots in odd-numbered columns are orange—again, only if the dots on both sides are black. Within each horizontal line of seven dots controlled by a single byte, you can have black, white, and one pair of colors. To change the color of any dot 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 dots controlled by the byte.

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

- Dots in even columns can be black, purple, or blue.
- Dots in odd columns can be black, green, or orange.
- If adjacent dots in a row are both on, they are both white.
- The colors in each row of seven dots 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 2-6. The blacks and whites are numbered to remind you that the high-order bit is different.

**Table 2-6**
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 dots 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 dots at this spacing cause a color monitor or TV set to produce color, but 2 or more white dots together do not. Effective horizontal resolution with color is 140 dots per line (280 divided by 2).

## Double Hi-Res graphics

In the Double Hi-Res graphics mode, the Apple IIGS displays an array of colored dots 560 columns wide and 192 rows deep. There are 16 colors available for use with Double Hi-Res graphics (see Table 2–7).

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–$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 dots 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 dots along a line. Think of a 4-dot-wide window moving across the screen: At any given time, the color displayed will correspond to the 4-bit value from Table 2-7 that corresponds to the window's position (Figure 2-9). Effective horizontal resolution with color is 140 (560 divided by 4) dots per line.

To use Table 2-7, 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***.

**Table 2-7**
Double Hi-Res graphics colors

| Repeated Color | ab0 | mb1 | ab2 | mb3 | bit pattern |
|---|---|---|---|---|---|
| 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 |

## Video display pages

The Apple IIGS generates its video displays 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, the object is two stacked colored blocks; and in Hi-Res and Double Hi-Res modes, it is a line of seven adjacent dots.

The 40-column text and Lo-Res graphics modes use 2 display pages of 1024 bytes each. These are called *text Page 1* and *text Page 2,* and they are located at 1024–2047 (\$0400–\$07FF) and 2048–3071 (\$0800–\$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 (4 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 located on the 80-column text card. 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 Chapter 3, take care of this extra addressing automatically; that is one reason to use those routines for all your normal text output.

The Hi-Res graphics mode also has 2 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 dots wide by 8 dots high. In Hi-Res graphics mode each byte controls an area 7 dots wide by 1 dot high. Thus, a Hi-Res display requires 8 times as much data storage, as shown in Table 2-8.

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

**Table 2-8**
Video display page locations

| | Display | Lowest address | | Highest address | |
|---|---|---|---|---|---|
| **Display mode** | **page** | **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 |

* This 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," earlier 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 2-9 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 2-9 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 2-9**
Display soft switches

| Name | Action | Hex | Function |
|---|---|---|---|
| CLR80COL | W | $C000 | Disable 80-column store |
| SET80COL | W | $C001 | Enable 80-column store |
| CLR80VID | W | $C00C | Disable 80-column hardware |
| SET80VID | W | $C00D | Enable 80-column hardware |
| CLRALTCHAR | W | $C00E | Normal lower case character set; flashing upper case char. set |
| SETALTCHAR | W | $C00F | Normal, inverse character set; no flash |
| RD80COL | R7 | $C018 | Read CLR/SET80COL switch ($C000/1) 1 = 80-column store enabled |
| RDVBL BAR | R7 | $C019 | Read vertical blanking: 1 = not VBL |
| RDTEXT | R7 | $C01A | Read TEXT switch: 1 = text mode enabled |
| RDMIX | R7 | $C01B | Read MIXED switch: 1 = mixed mode enabled |
| RDPAGE2 | R7 | $C01C | Read PAGE2 switch: 1 = text page 2 selected |
| RDHIRES | R7 | $C01D | Read HIRES switch: 1 = Hi-Res mode enabled |
| ALTCHARSET | R7 | $C01E | Read ALTCHAR switch: 1 = alternate character set in use |
| RD80VID | R7 | $C01F | Read 80COL switch: 1 = 80-column hardware in use |
| RDDHIRES | R7 | $C07F | Read DHIRES switch: 1 = DHIRES mode selected |
| TXTCLR | R/W | $C050 | Select standard Apple II graphics mode, or if MIXSET on, mixed mode |
| TXTSET | R/W | $C051 | Select text mode only |
| MIXCLR | R/W | $C052 | Clear mixed mode |
| MIXSET | R/W | $C053 | Select mixed mode |
| TXTPAGE1 | R/W | $C054 | Select text Page 1 |

| | | | |
|---|---|---|---|
| TXTPAGE2 | R/W | $C055 | Select text Page 2 or, if 80STORE on, Page 1 in auxiliary memory |
| LORES | R/W | $C056 | Select Lo-Res graphics mode |
| HIRES | R/W | $C057 | Select Hi-Res graphics mode or, if DHIRES is on, select Double Hi-Res graphics mode |
| DHIRES (AN3) | R/W | $C05E | Turn Double Hi-Res graphics mode on |
| DHIRES (AN3) | R/W | $C05F | Turn Select Double Hi-Res graphics mode off |

*Note:* 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.

❖ *By the way:* You may not need to deal with these functions by reading and writing directly to the memory locations in Table 2-9. 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.

## 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 2-5, 2-6, 2-7, 2-8, and 2-9. 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 3 rows that are 8 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, like the Apple II, 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 8 times as many bytes to store, because 8 rows of dots occupy the same space on the display as 1 row of characters. The subset consisting of all the first rows from the groups of 8 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 8 is stored in the second 1024 bytes, and so on for a total of 8 times 1024, or 8192 bytes. In other words, each block of 1024 bytes in the Hi-Res display page contains 1 row of dots out of every group of 8 rows. The individual rows are stored in sets of 3 40-byte rows, the same way as the text display.

All of the display modes except 80-column mode and Double Hi-Res graphics mode can use either of 2 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 obtain addresses for Hi-Res Page 2, add 8192 ($2000).

The 80-column display and Double Hi-Res graphics mode work a little differently. Half of the data are stored in the normal text Page 1 memory, and the other half are stored in memory on the 80-column text card using the same addresses. The display circuitry fetches bytes from these 2 memory areas simultaneously and displays them sequentially: first the byte from the 80-column text display memory, then the byte from the main memory. The main memory stores the characters in the odd columns of the display, and the 80-column text display 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 49153 ($C001). With SET80COL on, the page-select switch, TXTPAGE2, selects between the portion of the 80-column display memory in Page 1 of main memory and the portion stored in the 80-column text display memory. To enable the 80-column text display, turn the TXTPAGE2 soft switch on by reading or writing at location 49237 ($C055).

**Figure 2-5**
Map of 40-column text display

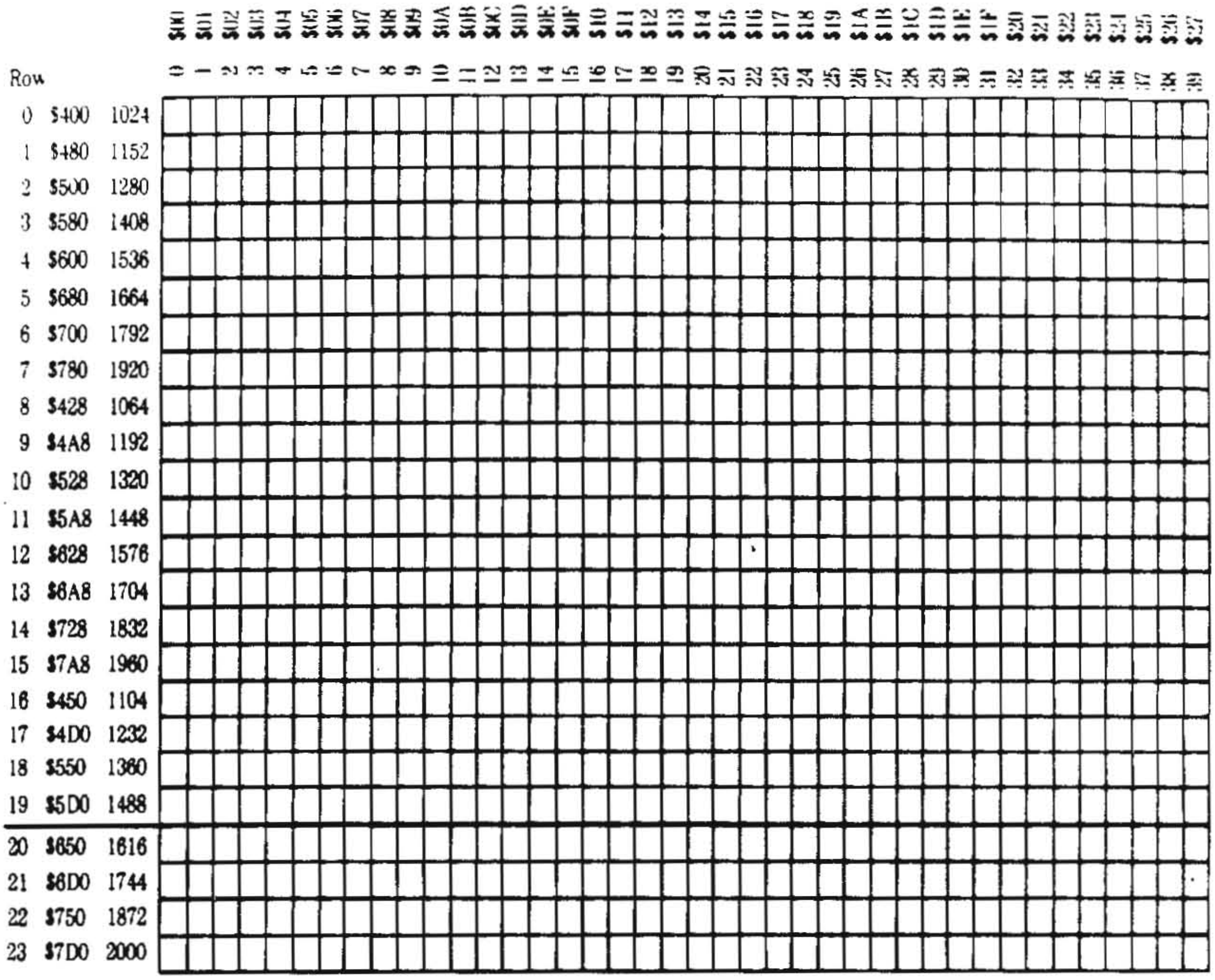


**Figure 2-6**
Map of 80-column text display

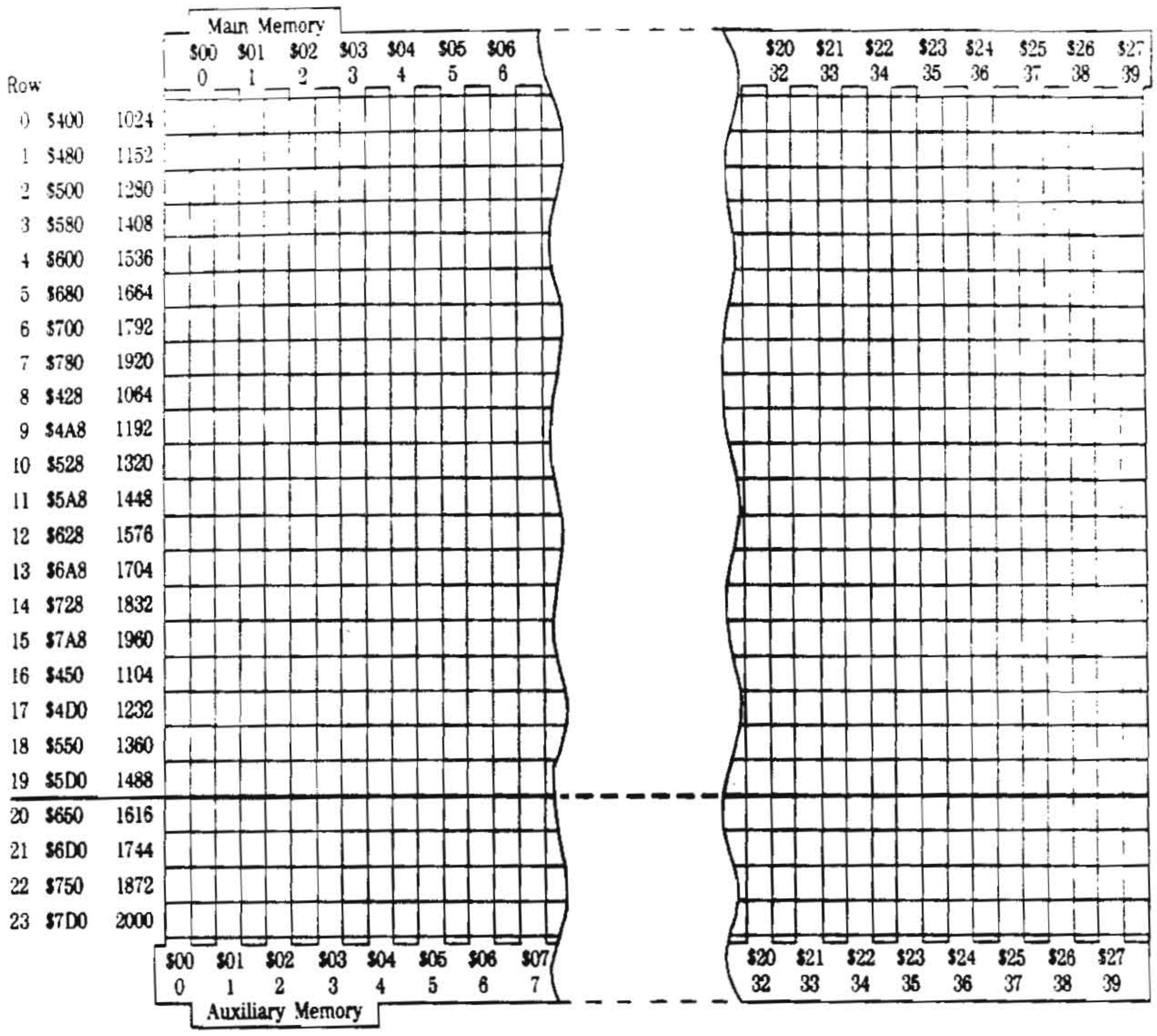


**Figure 2-7**
Map of Lo-Res graphics display

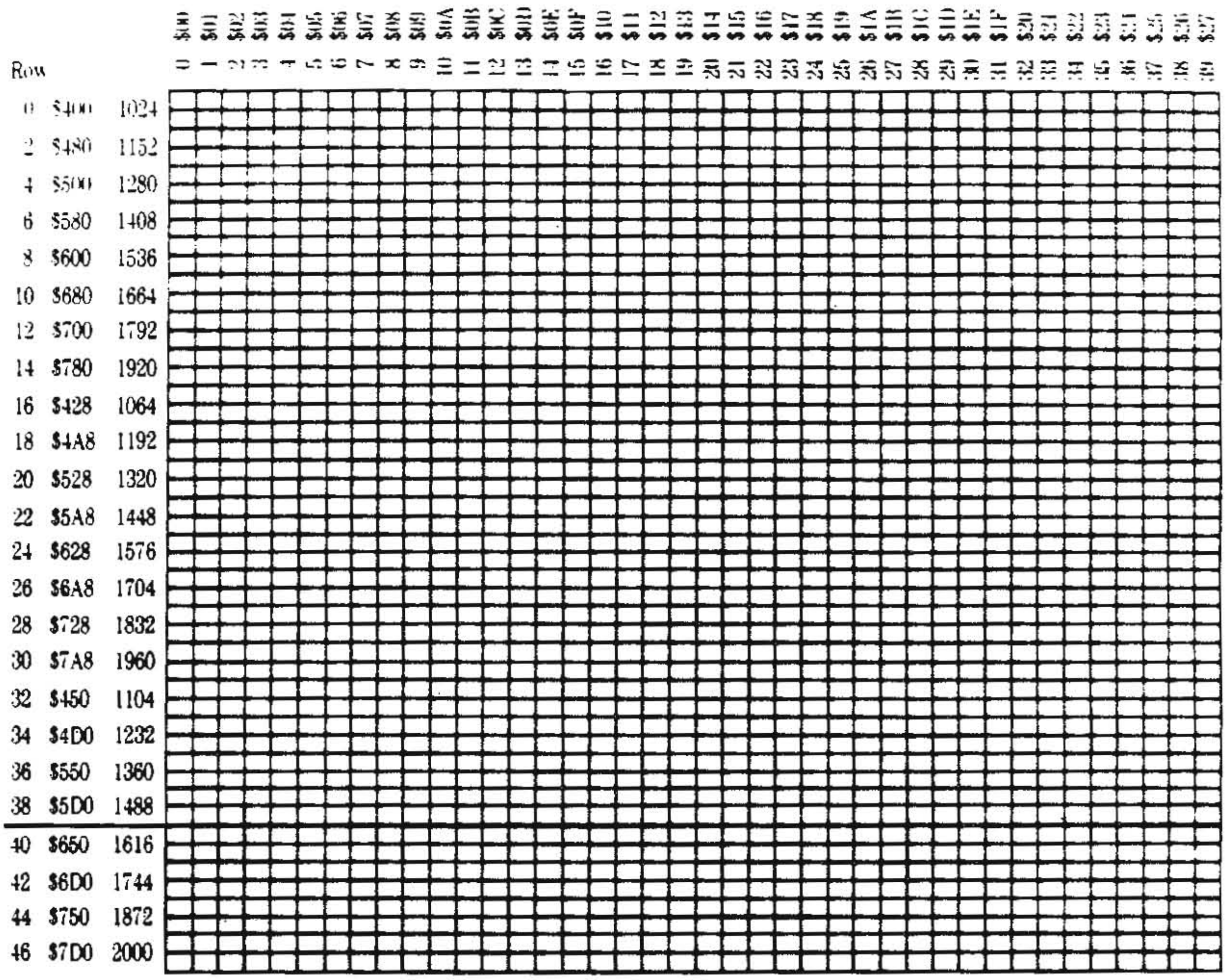


**Figure 2-8**
Map of Hi-Res graphics display

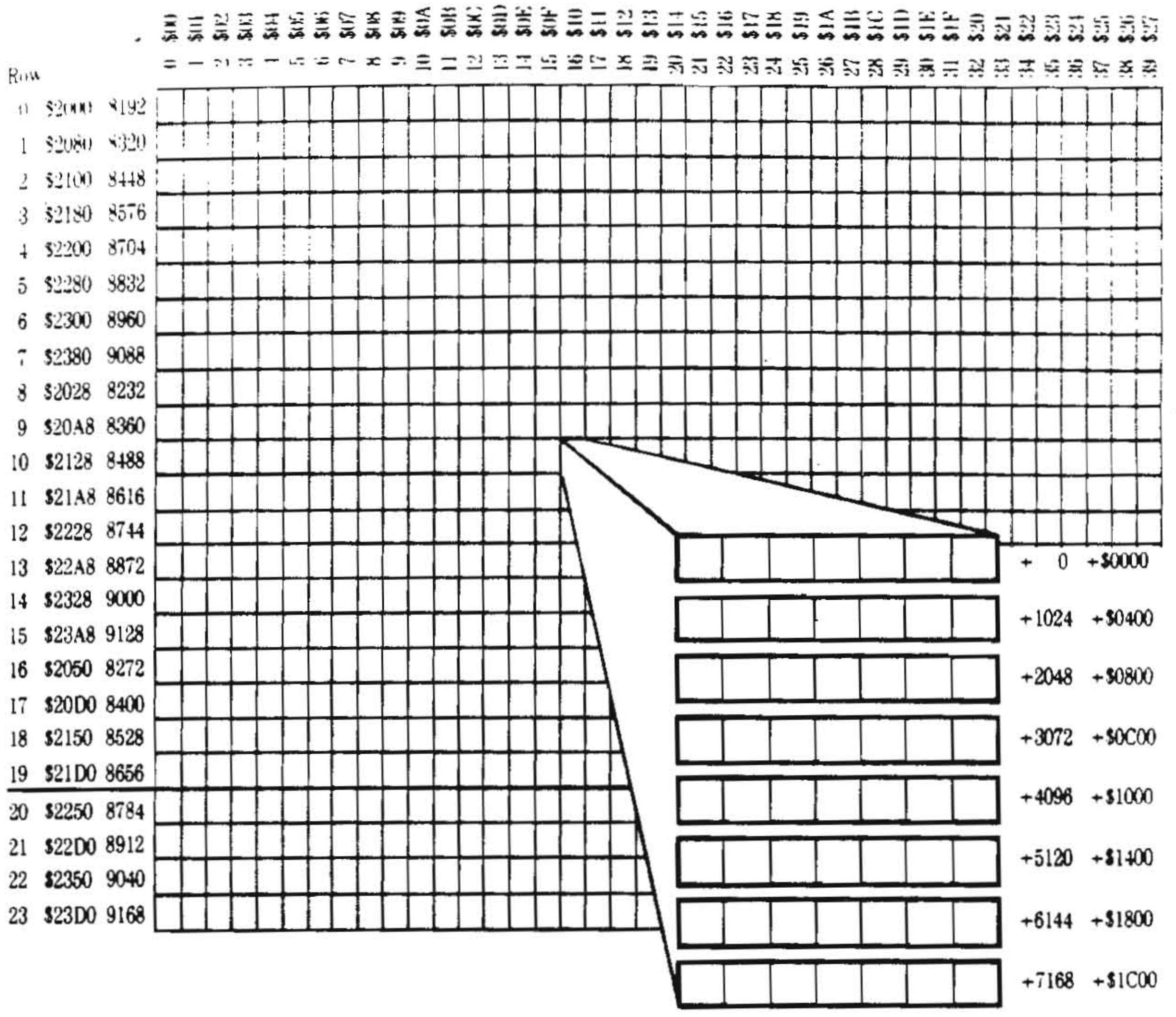
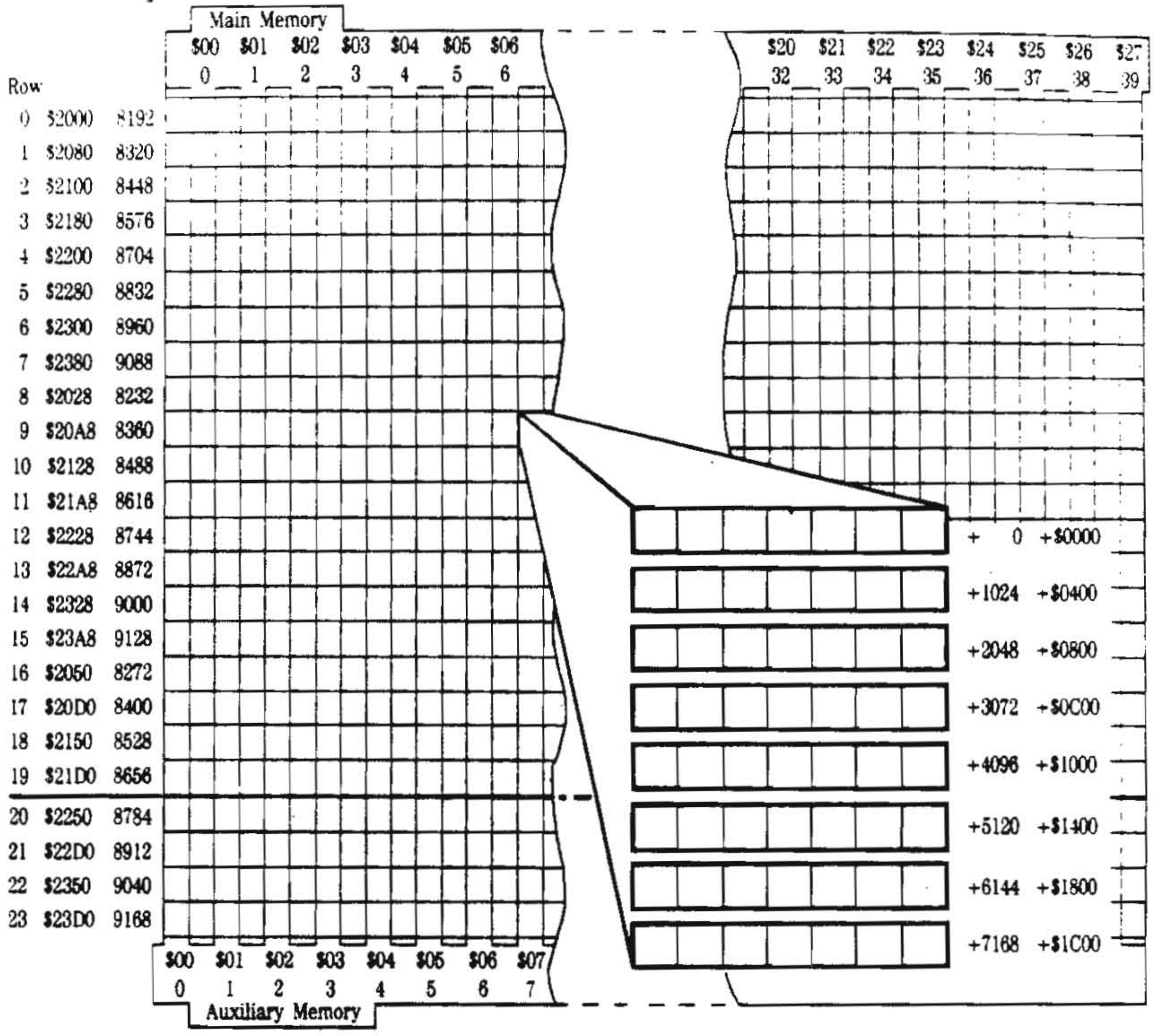


**Figure 2-9**
Map of Double Hi-Res graphics display



## The text window

After you have started up the computer or after a reset, the firmware uses the entire 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 32 ($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 39 ($27); in an 80-column display, the maximum value is 79 ($4F).

Memory location 33 ($21) holds the width of the text window. For a 40-column display, it is normally 40 ($28); for an 80-column display, it is normally 80 ($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, possibly destroying programs or data.

Memory location 34 ($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 23 ($17).

Memory location 35 ($23) contains the number of the bottom line of the screen, plus 1. It is normally 24 ($18) for the bottom line of the display. Its minimum value is 1.

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** position (stored at CH ($24) and CV ($25)) is inside the new window. If it is outside, it is possible to put characters into memory locations outside the display page, possibly destroying programs or data.

Table 2-10 summarizes the memory locations and the possible values for the window parameters.

**Table 2-10**
Text window memory locations

| Window parameter | Location | | Minimum value | | Normal values 40 col. | | Normal values 80 col. | | Maximum values 40 col. | | Maximum values 80 col. | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 |

## Secondary inputs and outputs

In addition to the primary I/O devices—the keyboard and display—there are several secondary input and output devices in the Apple IIGS. These devices are

- the speaker (output)
- the annunciator (output)
- the switch (input)
- analog hand controls (input)

These devices are similar in operation to the soft switches described in the preceding section: You control them by reading or writing to dedicated memory locations. Action takes place any time your program reads or writes to one of these locations; information written is ignored.

**Important** Some of these devices toggle—change state—each time they are accessed. If you write using an assembly-language indexed store operation, the microprocessor activates the address bus twice during successive clock cycles, causing a device that toggles each time it is addressed to end up back in its original state. For this reason, you should read, rather than write, to such devices.

### The speaker

The Apple IIGS has a small speaker mounted toward the front of the bottom plate. The speaker is connected to a soft switch that toggles; it has two states, off and on, and it changes from one to the other each time it is accessed. (At low frequencies, less than 400 Hz or so, the speaker clicks only on every other access.)

If you switch the speaker once, it emits a click; to make longer sounds, you access the speaker repeatedly. You should always use a read operation to toggle the speaker.

**Important** If you write to this soft switch using an assembly language indexed-write 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.

The soft switch for the speaker, SPKR, uses memory location 49200 ($C030). You can make various tones and buzzes with the speaker by using combinations of timing loops in your program.

## Game I/O

The Mega II supports paddles 0, 1, 2, and 3, and switches 0, 1, 2, and 3, which are available through the 16-pin DIP game connector located below slot 4 and the 9-pin connector 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. Figure 2-10 shows the two Apple IIGS game connectors, J9 and J21. Connector J21 is located on the main logic board just forward of expansion slot 4. Connector J9 is located at the rear of the main logic board. Table 2-11 lists the locations of the game I/O signals at the two connectors.

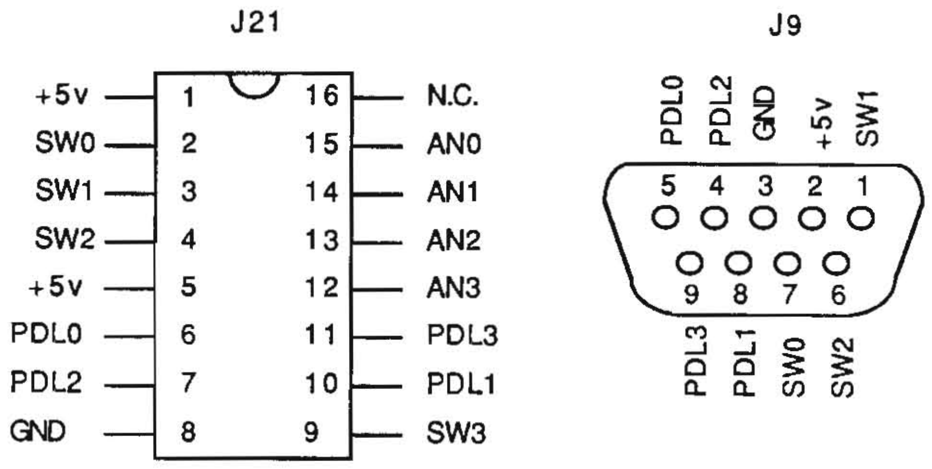


**Figure 2-10.** Game I/O connectors

**Table 2-11**
Game I/O signals

| **Pin numbers** | | |
|---|---|---|
| **J21** | **J9** | **Signal** |
| 1 | 2 | +5 volts |
| 2 | 7 | SW0; switch input 0 |
| 3 | 1 | SW1; switch input 1 |
| 4 | 6 | SW2; switch input 2 |
| 5 | - | +5 volt pull-up |
| 6 | 5 | PDL0; analog input 0 |
| 7 | 4 | PDL2; analog input 2 |
| 8 | 3 | 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. |

## The hand control signals

Several inputs and outputs are available at the 16-pin IC connector on the main logic board: 3 1-bit inputs, or switches, and 4 analog inputs, along with 4 one-bit outputs. You can access all these signals from your application program. Note that the SW3 signal 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 2 1-bit inputs. However, you can also use these inputs and outputs for many other jobs. For example, 2 analog inputs can be used with a 2-axis joystick. Figure 2-10 shows the connector pin numbers.

The Apple DeskTop Bus will accept ADB-type hand controls, joysticks, and graphics tablets as well as keyboards and mouse devices that have been designed specifically 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 7, "Apple DeskTop Bus," for more information.

### *Annunciator outputs*

The four 1-bit outputs 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 2-12. Any reference to the first location of a pair turns the corresponding annunciator off; a reference to the second location turns the annunciator on. There is no way to read the state of an annunciator.

**Table 2-12**
Annunciator memory locations

| Annunciator | | | Address | |
|---|---|---|---|---|
| **No.** | **Pin*** | **State** | **Decimal** | **Hex** |
| 0 | 15 | Off | 49240 | $C058 |
| | | On | 49241 | $C059 |
| 1 | 14 | Off | 49242 | $C05A |
| | | On | 49243 | $C05B |
| 2 | 13 | Off | 49244 | $C05C |
| | | On | 49245 | $C05D |
| 3 | 12 | Off | 49246 | $C05E |
| | | On | 49247 | $C05F |

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

### *Switch inputs*

The four 1-bit inputs can be connected to the output of another electronic device or to a pushbutton. 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 memory locations for these switches are 49249 through 49251 ($C060 through $C063), as shown in Table 2-13.

### *Analog inputs*

The 4 analog inputs are designed for use with 150K ohm variable resistors or potentiometers. The variable resistance is connected between the +5V 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 ($C070) does this. As soon as you reset the timing circuits, the high bits of the bytes at locations 49252 through 49255 ($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 4 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 2-13 shows the memory locations for all of the built-in I/O devices except the keyboard and display. As explained earlier, some soft switches should be accessed only by means of read operations; those switches are marked.

**Table 2-13**
Secondary I/O memory locations

| | Address | | |
|---|---|---|---|
| **Soft switch** | **Decimal** | **Hex** | **Definition** |
| SPKR | 49200 | $C030 | Toggle speaker (read only) |
| SETAN0 | 49241 | $C059 | Set annunciator 0 |
| CLRAN0 | 49240 | $C058 | Clear annunciator 0 |
| SETAN1 | 49243 | $C05B | Set annunciator 1 |
| CLRAN1 | 49242 | $C05A | Clear annunciator 1 |
| SETAN2 | 49245 | $C05D | Set annunciator 2 |
| CLRAN2 | 49244 | $C05C | Clear annunciator 2 |
| SETAN3 | 49247 | $C05F | Set annunciator 3 |
| CLRAN3 | 49246 | $C05E | Clear 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) |
| PTRIG | 49264 | $C070 | Analog input reset |
| PADDL0 | 49252 | $C064 | Read only |
| PADDL1 | 49253 | $C065 | Read only |
| PADDL2 | 49254 | $C066 | Read only |
| PADDL3 | 49255 | $C067 | Read only |

## Standard Apple II Memory

The 65C816 microprocessor in the Apple IIGS can emulate the 6502 microprocessor. Within the context of standard Apple II programs however, we will assume that the microprocessor addresses only 65,536 (64K) memory locations, and will limit the discussion of memory in this chapter to the main and auxiliary RAM banks. For information on the 65C816's ability to address locations in other than the main or auxiliary banks, see Chapter 3, "New features".

All input and output are ***memory mapped.*** This means that all devices connected to the Apple IIGS appear to be a set of memory locations to the computer. In this chapter, the I/O memory spaces are described simply as blocks of memory.

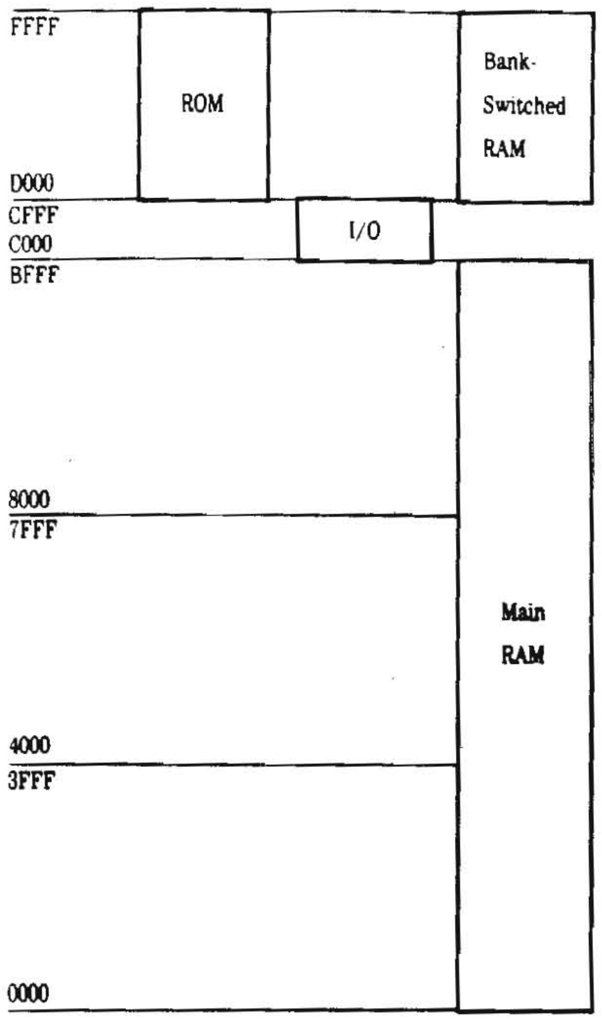
All the RAM, ROM, and I/O devices are allocated locations in the 64K address range. Since each device or function requires a certain block of memory, there are more devices and functions than there are legal addresses, which means that the legal addresses must be shared. This sharing is accomplished through a technique called ***bank-switching***, which is explained under the "Bank-Switched Memory" and "Auxiliary Memory" sections in this chapter.

Programmers often refer to the Apple IIGS memory in 256-byte blocks called ***pages***. One reason for this is that a 1-byte address counter or index register can specify 1 of 256 different locations. Thus, ***page 0*** consists of memory locations from 0 to 255 ($00 to $FF), inclusive; ***page 1*** consists of locations 256 to 511 ($0100 to $01FF). Note that the page number is the high-order part of the hexadecimal address. Don't confuse this kind of page with the display buffers in Apple II computers, which are sometimes referred to as Page 1 and Page 2.

## Main memory map

The map of the main memory address space in Figure 2-11 shows the functions of the major areas of memory.

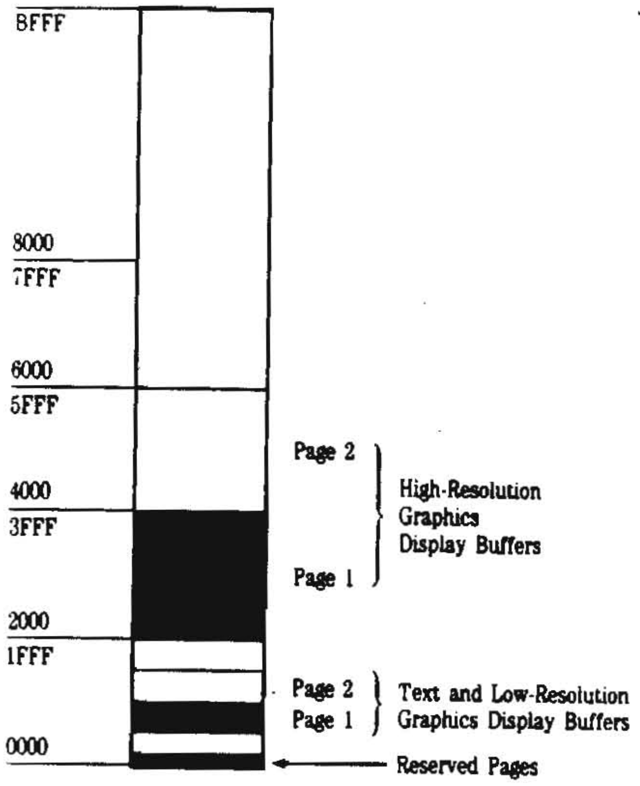
**Figure 2-11**
Memory map for bank $E0.



## RAM memory allocation

As Figure 2-11 shows, the major portion of the Apple IIGS memory space is allocated to program storage (RAM). Figure 2-12 shows the areas allocated to RAM. The main RAM memory extends from location 0 to location 49151 (hex $BFFF), and occupies pages 0 through 191 (hex $BF). There is also RAM storage in the bank-switched space from 53248 to 65535 (hex $D000 to $FFFF), described in the section "Bank-Switched Memory" later in this chapter, and auxiliary RAM, described in the section "Auxiliary Memory" later in this chapter.

**Figure 2-12**
RAM allocation map


BFFF
8000
7FFF
6000
5FFF
4000
3FFF
2000
1FFF
0000
Page 2
Page 1
High-Resolution Graphics Display Buffers
Page 2
Page 1
Text and Low-Resolution Graphics Display Buffers
Reserved Pages

## Reserved memory pages

Most of the Apple IIGS RAM is available for storing your programs and data. However, a few RAM pages are reserved for the use of the Monitor firmware and the BASIC interpreters. 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.

### *Direct page*

Several of the 6502 microprocessor's addressing modes require the use of addresses in memory page zero, also called *direct* *page.* The Monitor firmware, the BASIC interpreters, DOS 3.3, and ProDOS® all make extensive use of direct page.

To use indirect addressing in your assembly-language programs, you must store base addresses in direct page. At the same time, you must avoid interfering with the other programs that use direct page—the Monitor program, the BASIC interpreters, and the disk operating systems. The best way to avoid conflicts is to save and restore only those direct page locations you use.

### *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 (via PUSH and PULL instructions).

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 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 now lost. This writing over old data is called *stack overflow,* and when it happens, the program continues to run normally until the lost information is needed, whereupon the program terminates catastrophically.

In native mode, the stack pointer is 16 bits long, and the stack can hold 64K of information at a time. To read more about using the 65C816 stack in native mode, see Chapter 11, "65C816 Microprocessor".

**Important** Wraparound does not occur in all addressing modes.

### *The input buffer*

The GETLN input routine, which is used by the Monitor program and the BASIC interpreters, uses page 2 as its keyboard-input buffer. The size of this buffer sets the maximum size of input strings. (Note: Applesoft 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.

### *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.

### *The display buffers*

The primary text and Lo-Res graphics display buffer occupies memory pages 4 through 7 (locations 1024 through 2047, hexadecimal $0400 through $07FF). This entire 1024-byte area is called *text Page 1,* and it is not usable for program and data storage. There are 64 locations in this area that are not displayed on the screen; these locations are reserved for use by the peripheral cards.

Text Page 2, the alternate text and Lo-Res graphics display buffer, occupies memory pages 8 through 11 (locations 2048 through 3071, hexadecimal $0800 through $0BFF). Most programs do not use text Page 2 for displays, so you can use this area for program or data storage.

The primary Hi-Res graphics display buffer, called *Hi-Res graphics Page 1,* occupies memory pages 32 through 63 (locations 8192 through 16383, hexadecimal $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 occupies memory pages 64 through 95 (locations 16384 through 24575, hexadecimal $4000 through $5FFF). Most programs use this area for program or data storage.

The primary Double Hi-Res graphics display buffer, called *Double Hi-Res graphics Page 1,* occupies memory pages 32 through 63 (locations 8192 through 16383, hexadecimal $2000 through $3FFF) in both main and auxiliary memory. If your program doesn't use Hi-Res or Double Hi-Res graphics, this area of main memory is usable for programs or data.

## Bank-switched memory

The memory address space from 52K to 64K ($D000 through $FFFF) is doubly allocated: It is used for both ROM and RAM. The 12K of ROM in this address space contain the Monitor program and the Applesoft BASIC interpreter. Alternatively, there are 16K of RAM in this space. The RAM is normally used for storing the operating system (purchased separately).

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 interpreter in ROM, but the Apple IIGS, like an Apple II with a language card, 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 52K to 56K ($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 52K to 64K ($D000 through $FFFF) is switched between ROM and RAM, and second, the address space from 52K to 56K ($D000 to $DFFF) is switched between two different blocks of RAM. (See Figure 2-13.)

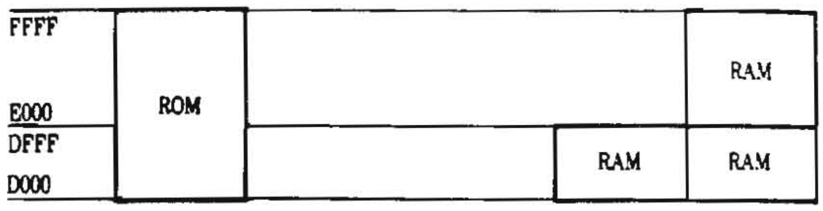


**Figure 2-13**
Bank-switched memory map

## Setting bank switches

You switch banks of memory in the same way you switch other functions in a standard Apple II: by using soft switches. Read operations to these soft switches 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 2-14 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.

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

**Table 2-14**
Language card bank select switches

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

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

❖ *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
LDA #PAT4K
JSR YOURSUB
```

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 read 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. The byte-wide format of the soft switch 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 2-14.)

**Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in Chapter 1, "Introduction to the Apple IIGS."

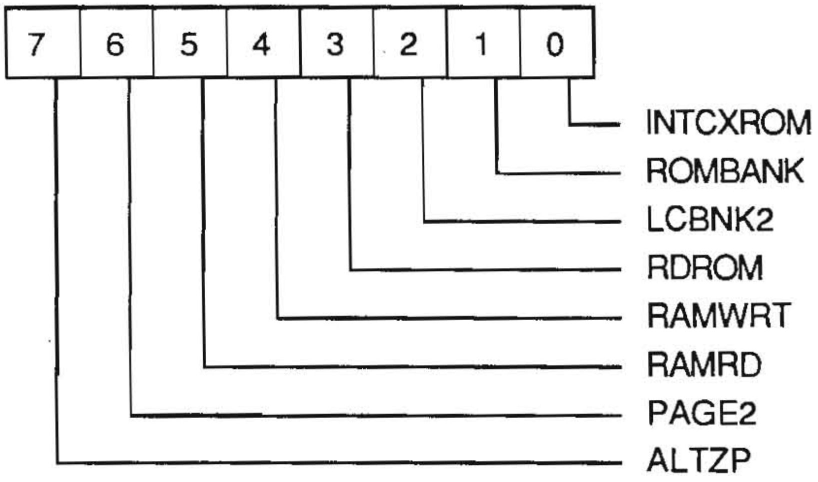


**Figure 2-14** . State register at $C068

| Bit | Value | Description |
|---|---|---|
| 7 | 1 | ALTZP: If this bit is 1, then bank-switched memory, stack and zero page are in main memory. |
| | 0 | If this bit is 0, bank-switched memory, stack and zero 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, the language-card RAM bank 2 is selected. |
| 1 | - | ROMBANK: 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 slot card ROM at Cx00 is selected. |

## Auxiliary memory

The auxiliary bank has a 1K area of memory that serves the purpose of expanding the text display to 80 columns. The other 63K can be used as auxiliary program and data storage. If you use only 40-column displays, the entire 64K bytes is available for programs and data.

**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 zero 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 2-15, the auxiliary memory is broken 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: As described in the section "Text Modes" earlier in this chapter, space in auxiliary memory is used for one-half of the 80-column text display. You can switch to the auxiliary memory for this entire memory space, or you can switch just the display pages: See the next section, "Memory Mode Switching."

- ❖ *Soft switches:* If the only reason you are using auxiliary memory is for the 80-column display, note that you can store into the display page in auxiliary memory by using the SET80COL and TXTPAGE2 soft switches described in the section "Display Mode Switching" earlier in this chapter.

The other large section of auxiliary memory 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 "Bank-Switched Memory." If you use the auxiliary RAM in this space, the soft switches have the same effect on the auxiliary RAM that they do on the main RAM: The bank switching is independent of the auxiliary RAM switching.

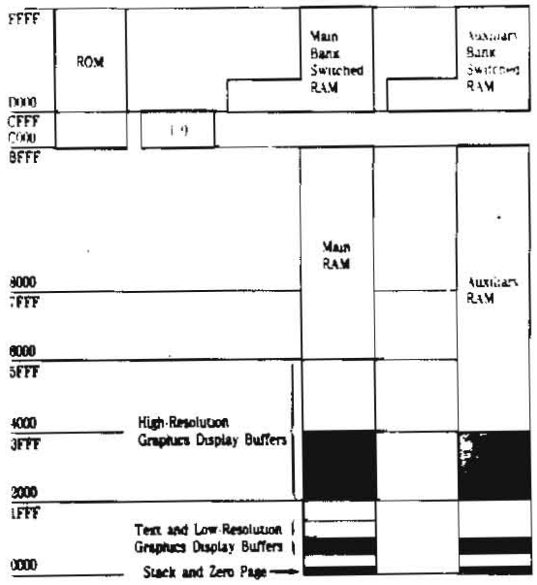


**Figure 2-15**
**Memory map with auxiliary memory**

- ❖ ***Bank switches:*** Note that the soft switches for the bank-switched memory, described in the previous section, do not change when you switch to auxiliary RAM. In particular, if ROM is enabled in the bank-switched memory space before you switch to auxiliary memory, the ROM will still be enabled after you switch. Any time you switch the bank-switched 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 bank-switched space, you also switch the first two pages, from 0 to 511 ($0000 through $01FF). This part of memory contains page zero, which is used for important data and base addresses, and page 1, which is the 6502 stack. The stack and zero page are switched this way so that system software running in the bank-switched memory space can maintain its own stack and zero page while it manipulates the 48K address space (from $0200 to $BFFF) in either main memory or auxiliary memory.

## Memory mode switching

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 ones named WRMAINRAM and WRCARDRAM select main or auxiliary memory for writing. As shown in Table 2-15, 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 the Apple IIGS. For example, if you switch to expansion memory with no memory expansion card in the memory expansion slot, the program that is running will crash.

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.

Auxiliary memory corresponding to text Page 1 and Hi-Res graphics Page 1 can be used as part of the address space from $0200 to $BFFF by using RAM read and RAM write soft switches as described above. These areas in auxiliary RAM can also be controlled separately by using the switches described in the section "Display Mode Switching" earlier in this chapter. Those switches are named SET80COL, TXTPAGE2, and HIRES.

As shown in Table 2-15, 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, from $0400 to $07FF; with HIRES on, TXTPAGE2 switches both text Page 1 and Hi-Res graphics Page 1, from $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, 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 zero page area between main and auxiliary memory. As shown in Table 2-15, writing to location $C009 turns ALTZP on and selects auxiliary memory stack and zero page; writing to the soft switch at location $C008 turns ALTZP off and selects main memory stack and zero page for both reading and writing.

**Table 2-15**
Auxiliary-memory select switches

| | | Location | | |
|---|---|---|---|---|
| **Name** | **Function** | **Hex** | **Decimal** | **Notes** |
| RDCARDRAM | Read auxiliary memory | $C003 | 49155 | Write |
| RDMAINRAM | Read main memory | $C002 | 49154 | Write |
| RDRAMRD | Read switch status | $C013 | 49171 | Read (1=aux, 0=main) |
| WRCARDRAM | Write auxiliary memory | $C005 | 49157 | Write |
| WRMAINRAM | Write main memory | $C004 | 49156 | Write |
| RDRAMWRT | Read switch status | $C014 | 49172 | Read (1=aux, 0=main) |
| SET80COL | On: access display page | $C001 | 49153 | Write |
| CLR80COL | Off: use RAM switches, above | $C000 | 49152 | Write |
| RD80COL | Read switch status | $C018 | 49176 | Read (1=80-col. store on) |
| TXTPAGE2 | Page 2 on (aux. memory) | $C055 | 49237 | Write |
| TXTPAGE1 | Page 1 on (main memory) | $C054 | 49236 | Write |
| RDPAGE2 | Read switch status | $C01C | 49180 | Read (1=page 2, 0=page 1) |
| HIRES | On: access Hi-Res pages | $C057 | 49239 | Write |
| LORES | Off: use RAM switches, above | $C056 | 49238 | Write |
| RDHIRES | Read switch status | $C01D | 49181 | Read (1=HIRES on, 0=off) |
| SETALTZP | Aux. stack & zero page | $C009 | 49161 | Write |
| SETSTDZP | Main stack & zero page | $C008 | 49160 | Write |
| RDALTZP | Read switch status | $C016 | 49174 | Read (1=aux, 0=main) |

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

There are three more locations associated with the auxiliary memory switches. The high-order bits of the bytes you read at these locations tell you the settings of the three soft switches described above. The byte you read at location $C013 has its high bit set to 1 if the auxiliary memory is read-enabled, or 0 if the 48K block of main memory is read-enabled. The byte at location $C014 has its high bit set to 1 if auxiliary memory is write-enabled, or 0 if the 48K block of main memory is write-enabled. The byte at location $C016 has its high bit set to 1 if ALTZP is on (the bank-switched area, stack, and page zero in the auxiliary memory are selected), or 0 if ALTZP is off (these areas in main memory are selected).

❖ *Sharing memory:* In order to have enough memory locations for all the soft switches and remain compatible with the Apple II and Apple II Plus, the soft switches listed in Table 2-15 share their memory locations with the keyboard functions listed in Table 2-2. The operations—read or write—(shown in Table 2-15) for controlling the auxiliary memory are just the ones that are not used for reading the keyboard and clearing the strobe.

## Peripheral expansion

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 peripheral device is assigned to a slot, and cards are plugged into any of the seven peripheral slots. This allows devices, such as a serial port, to be built onto the main logic board; however, only one device (either the built-in peripheral device or the slot peripheral device) can be selected at a time.

### The Slot register

The Control Panel (accessible by pressing the Apple-Control-Esc keys simultaneously) allows the user to select the appropriate device for each logical slot. Enabling and disabling of internal peripheral devices may also be achieved under software control by setting the bits in the Slot Select register at location $C02D. The bit representations are given in figure 2-16.

**Warning** To prevent the operating system from crashing, do not manipulate the bits within the Slot Select register under software control.

**Warning** Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in Chapter 1, "Introduction to the Apple IIGS".

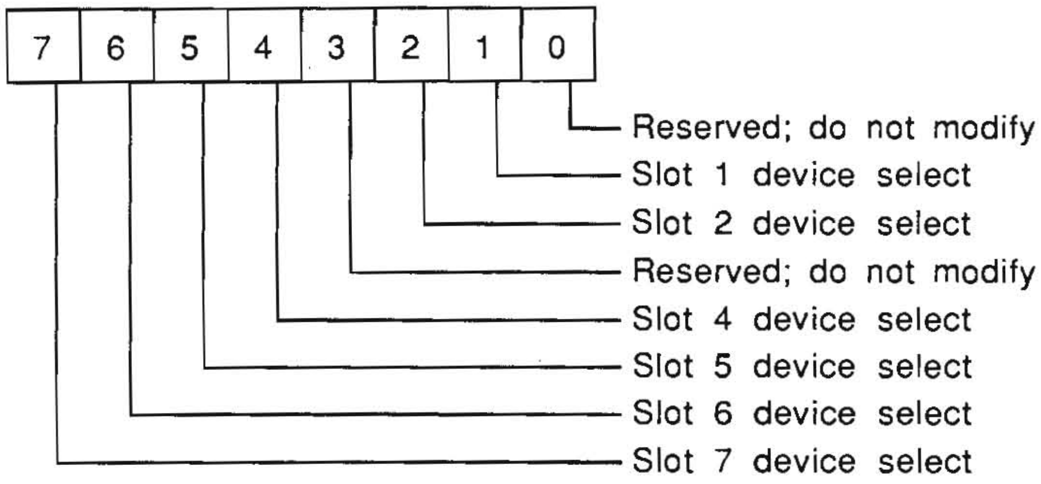


**Figure 2-16**. The Slot register at $C02D

| 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) 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) 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 ($C0C0 to $C0CF) and 4 ($C0D0 to $C0DF) 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. Those memory spaces are used for small dedicated programs such as I/O drivers. Peripheral cards that contain their own driver routines in firmware like this 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 2-16.

These memory locations are used for different I/O functions, depending on the design of each peripheral card. Whenever the Apple IIGS addresses 1 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 4 low-order address lines to determine which of its 16 I/O locations is being accessed.

**Table 2-16**
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 with 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 2-17 and Figure 2-16. 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 8 low-order address lines determine which of the 256 memory locations is being accessed.

**Table 2-17**
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 2-11.) 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 have expansion ROM on it, 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); second, it enables the expansion ROM devices when the /IOSTRB signal, pin 20 on the slot, becomes active (low). Figure 2-17 shows a typical ROM enable circuit.

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. 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 used to enable the expansion-ROM devices on a peripheral card. (See Figure 2-17.)

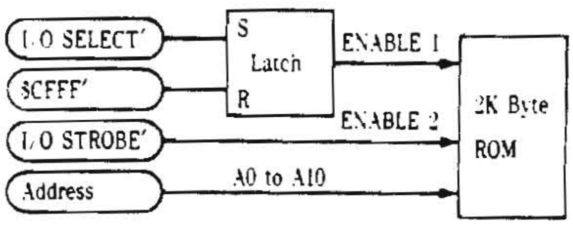


**Figure 2-17**
Expansion ROM enable circuit

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 reenable 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.

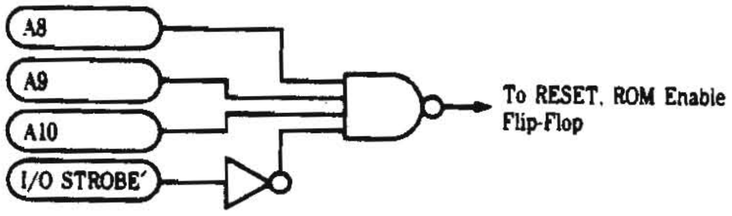


**Figure 2-18**
ROM disable address decoding

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 2-18.

### Peripheral-card RAM space

There are 56 bytes of main memory allocated to the peripheral cards, 8 bytes per card, as shown in Table 2-18. These 56 locations are actually in the RAM memory reserved for the text and Lo-Res graphics displays, but these particular locations 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 2-18**
Peripheral-card RAM memory locations

| **Base** | **Slot number** | | | | | | |
|---|---|---|---|---|---|---|---|
| **address** | **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. 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 2-19 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 4 left shifts, then loads it into an index register and uses the base address to specify 1 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
```

❖ *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 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.

**Table 2-19**
Peripheral-card I/O base addresses

| **Base** | **Connector number** | | | | | | |
|---|---|---|---|---|---|---|---|
| **address** | **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 |

## RAM addressing

A program on a peripheral card can use the eight base addresses shown in Table 2-19 to access the eight RAM locations allocated for its use. The program does this by putting its slot number into the Y Index register 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), 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 since 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 in this computer there are built-in functions that also use this memory space. Figure 2-19 shows the division of address space to the built-in devices. The soft switches that control the allocation of this memory space are described in the next section.

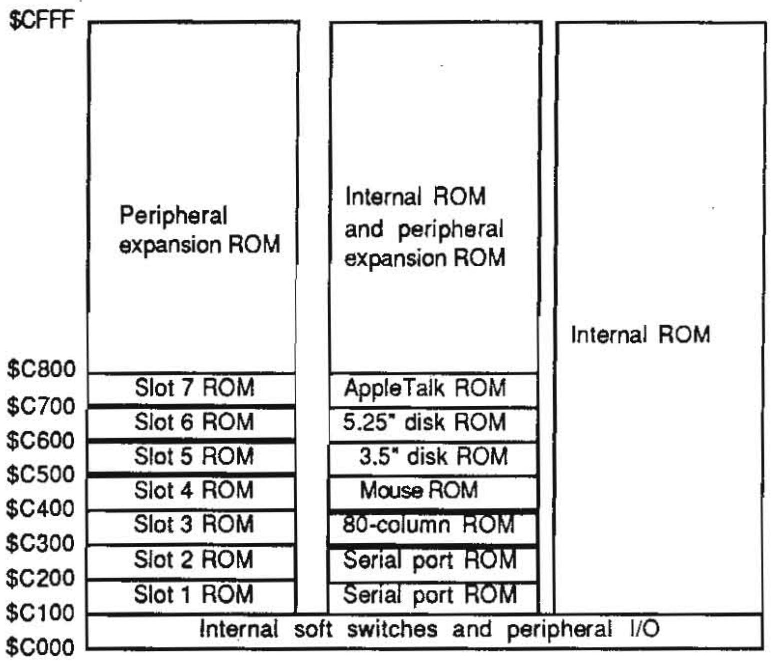


**Figure 2-19**
I/O memory map

## Switching I/O memory

The built-in firmware uses two 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 2-20.

❖ *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 2-20.

**Table 2-20**
I/O memory switches

| | | Location | | |
|---|---|---|---|---|
| **Name** | **Function** | **Hex** | **Decimal** | **Notes** |
| SETSLOTC3ROM | Slot ROM at $C300 | $C00B | 49163 | Write |
| SETINTC3ROM | Internal ROM at $C300 | $C00A | 49162 | Write |
| RDC3ROM | Read SLOTC3ROM switch | $C017 | 49175 | Read (1 = slot ROM enabled, 0 = internal ROM enabled) |
| SETSLOTCXROM | Slot ROM at $Cx00 | $C006 | 49159 | Write |
| SETINTCXROM | 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 slot 3 address space because slot 3 is normally used with a terminal interface, so the built-in firmware 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 SLOTCXROM signal is active (high), the I/O memory space from $C100 to $C7FF is allocated to the expansion slots, as described previously. Setting SLOTCXROM inactive (low) disables the peripheral-card ROM and selects built-in ROM in all of the I/O memory space except the part from $C000 to $C0FF (used for soft switches and data I/O), as shown in Figure 6-3.

❖ *Note:* Setting SLOTCXROM low 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 in if there was an 80-column 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 routines at $C311 and $C314.
- Don't use unpublished entry points into the internal $Cn00 firmware, because there is no guarantee that they will stay the same.
- 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* manual for information on firmware protocol.

## 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. This is done by saving the machine's state at the time of the interrupt, placing the Apple 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* manual.

# Chapter 3

# New Features

The Fast Processor Interface (FPI) is one of the two major subsystems that make up the Apple IIGS. It provides these new features for the Apple II family:

- faster processor speed
- support of additional RAM
- I/O shadowing

Figure 3-1 shows the relationship of the FPI to other parts of the Apple IIGS. This chapter describes the FPI subsystem and the new functions.

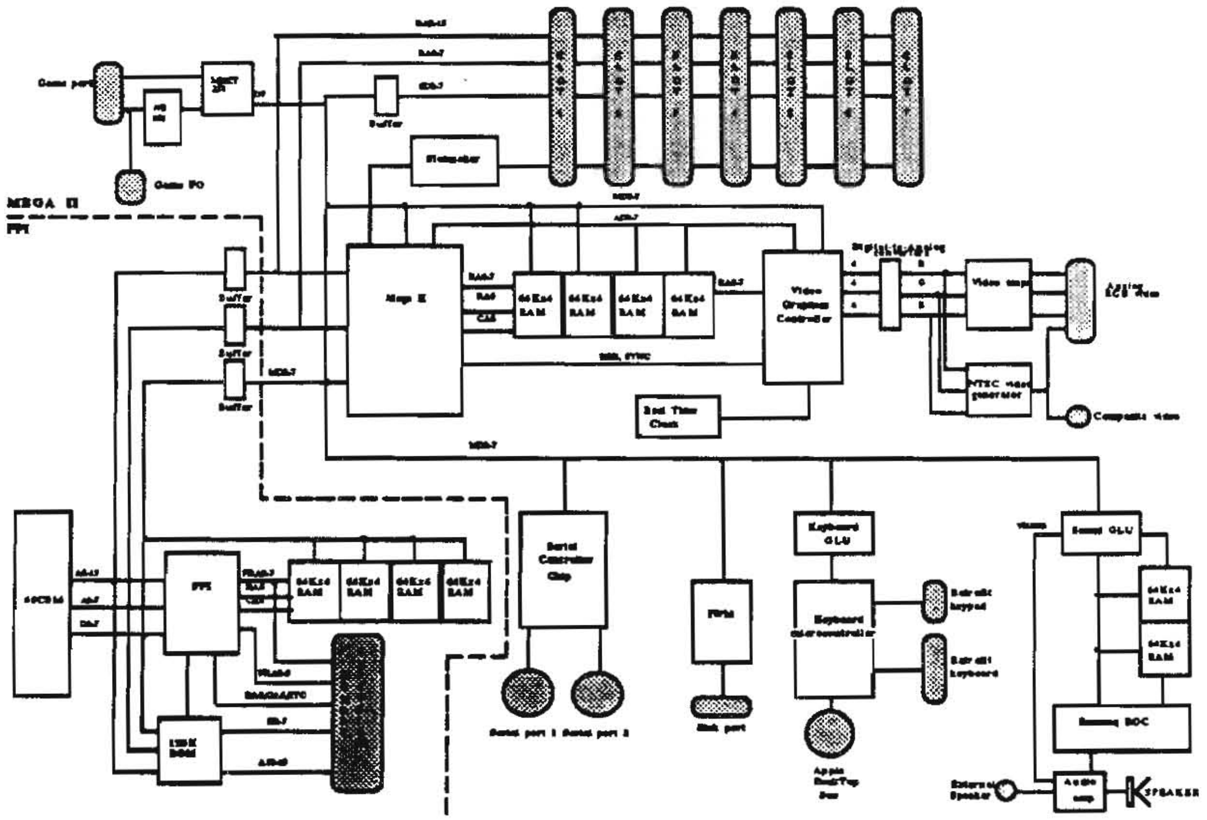


**Figure 3-1.** Relationship of the FPI subsystem to other system components

## The FPI subsystem

During normal operation, the FPI side of the system runs at 2.8 MHz and the Mega II side runs at 1.024 MHz. This allows faster processing without disturbing the standard 1.024 MHz speed for I/O, video timing necessary for compatibility, and existing peripherals.

Data must be transferred and soft switches accessed between the Mega II and the FPI. When a program running on the FPI side of the system must access an I/O or Mega II RAM location, the system slows down briefly to 1.024 MHz and synchronizes itself with the Mega II timing so that the access can be accomplished. When the access is finished, the FPI side returns to the normal 2.8 MHz operating speed, and the Mega II side continues at 1.024 MHz.

A combination of existing Apple II soft switches, Mega II soft switches, and FPI control registers controls the various functions of the FPI. The control registers include:

- the State register
- the Shadow register
- the Speed register

## Memory allocation

The FPI controller can access a minimum of 128K of RAM and is expandable to 8Mb. This RAM is separate from the 128K RAM available to the Mega II and is used for text and graphics display buffers and system software. The FPI also has access to 128K of ROM, expandable to 1Mb. The application program is free to use the remaining locations in banks $0 and $1 and those in the RAM expansion banks. Figure 3-2 shows the system memory map.

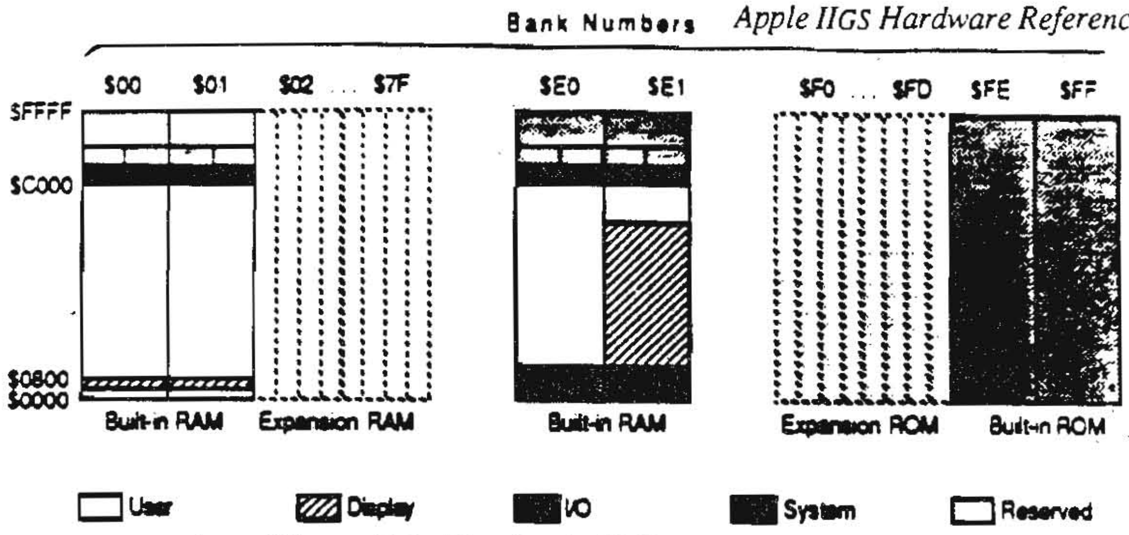


**Figure 3-2.** The Apple IIGS memory map

## The State register

The State register is duplicated in the FPI IC, allowing access to eight of the commonly used soft switches in a single transfer, as shown in Figure 3-3. This means that reading the State register at location $C068 allows you to read the state of the soft switches without slowing the system down. Write operations to the State register will slow the system down momentarily. Figure 3-3 shows the soft switches in the State register.

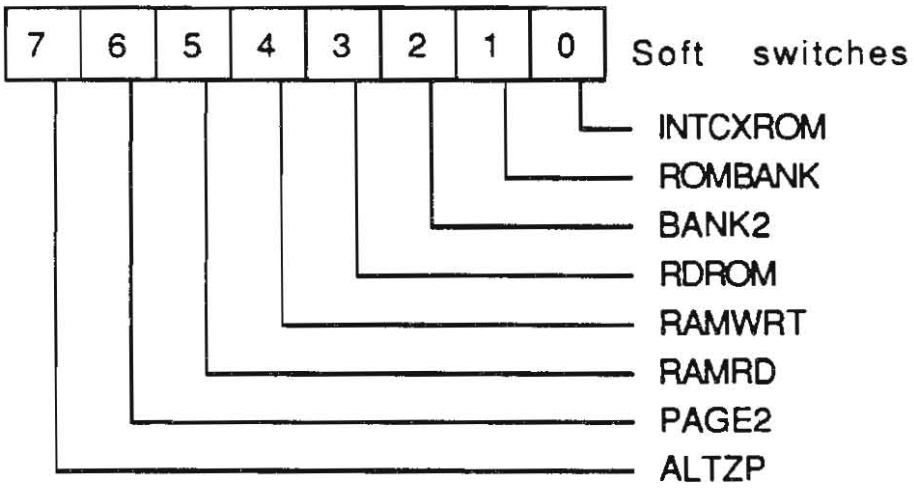


**Figure 3-3.** State register at $C068

## Shadowing

Shadowing is the process of duplicating the I/O addresses in another bank of RAM. Shadowing may be enabled in any RAM memory bank. The I/O addresses are located in banks $E0 and $E1, which is the low-speed Mega II main address space. By shadowing these locations in the high-speed FPI address space, only writes to the I/O location require the system to slow down.

An I/O write actually writes to the I/O address in both banks, the Mega II bank, $E0 or $E1, and the high-speed bank, $00 or $01 (when shadowing is enabled in $00 and $01 only). Shadowing, therefore, helps minimize the impact of video display updates on the overall system speed: Only I/O writes are done at low system speed; I/O reads are done at full system speed.

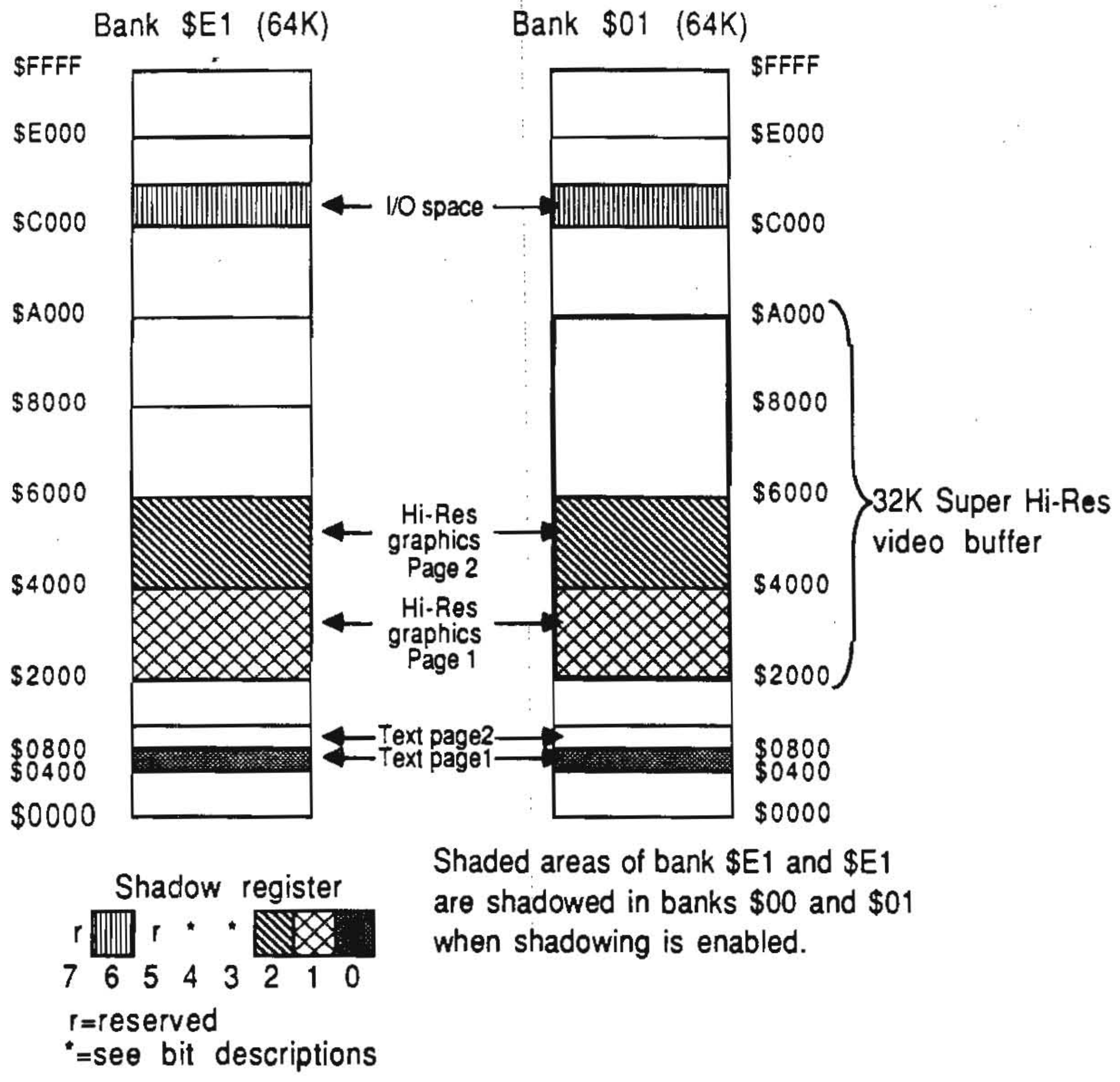
The shadowing options are

- enable shadowing for banks $00 and $01 only
- enable shadowing for all RAM banks (not recommended)

*Note*: Shadowing of 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.

Choose which banks you want shadowing enabled in by setting or clearing bit 4 in the Speed register (described later in this chapter). This will duplicate the I/O locations and any portions of the video buffers you select (via the Shadow register) in those shadow-enabled RAM banks. Reads and writes can now be done from and to these I/O locations in the shadowed banks. Direct access to I/O and the video buffers is not inhibited and may still be accesed through banks $E0 and $E1. Figure 3-4 shows banks $E1 and $01, or any other odd-numbered shadow-enabled bank.

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



The Shadow register above shows which bits control shadowing of which areas.

**Figure 3-4.** Shadowed memory map

## The Shadow register

The Shadow register, located at $C035, controls which address ranges of each shadowed high-speed RAM bank are duplicated in the Mega II RAM display areas. The Shadow register also determines whether or not the I/O space/language-card (IOLC) areas for each bank are implemented. Figure 3-5 shows the format of the Shadow register, followed by a list of each 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 Chapter 1, "Introduction to the Apple IIGS".

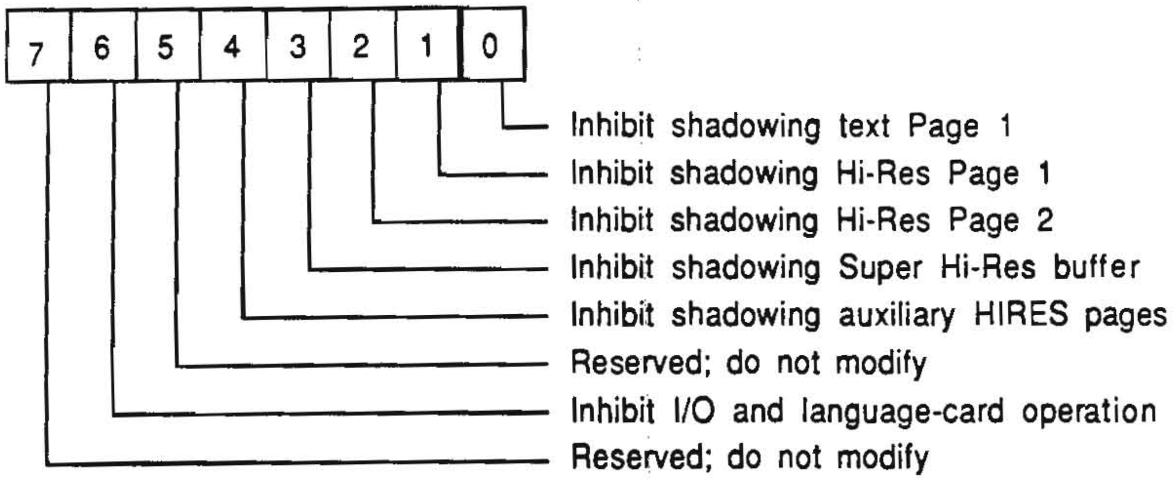

7
6
5
4
3
2
1
0
Inhibit shadowing text Page 1
Inhibit shadowing Hi-Res Page 1
Inhibit shadowing Hi-Res Page 2
Inhibit shadowing Super Hi-Res buffer
Inhibit shadowing auxiliary HIRES pages
Reserved; do not modify
Inhibit I/O and language-card operation
Reserved; do not modify

**Figure 3-5.** Shadow register at $C035

| 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 acts as RAM or I/O. When this bit is 0, I/O is enabled in the $Cxxx space and the RAM that would normally occupy that space becomes a second $Dxxx RAM space, forming a language card. |
|  | 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 the language card, see Chapter 2, "The Mega II: Maintaining Compatibility."

| Bit | Value | Description |
|---|---|---|
| 5 | - | Reserved; do not modify. |
| 4 | 1 | Inhibit shadowing for auxiliary Hi-Res graphics pages: When this bit is 1, all shadowing enabled for Hi-Res graphics pages 1 and 2 (as determined by bits 0 through 3 in this register) is disabled for all shadowed auxiliary (odd) banks. Shadowing of main bank Hi-Res graphics pages remains uneffected. |
|  | 0 | When this bit is 0, all shadowing enabled for Hi-Res graphics pages (as determined by bits 0 through 3 in this register) is enabled for auxiliary bank Hi-Res graphics pages as well. |
| 3 | 1 | Super Hi-Res 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 buffer. |
| 2 | 1 | Hi-Res graphics Page 2 inhibit: When this bit is 1, shadowing is disabled for Hi-Res video Page 2 and auxiliary Hi-Res video 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 page Hi-Res 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 video 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 Page Hi-Res graphics 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. |

*Note*: Text Page 2 ($0800 through $0BFF) is never shadowed. If you need a text display area or a code storage area, use Mega II banks $E0 and $E1. These banks are limited to 1.024 MHz operation, however.

Areas within each shadow-enabled 64K bank may be shadowed or not by setting the corresponding bit or bits in the Shadow register. Shadowing may be turned off (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 and reclaim the free memory space in the unused video buffers in Mega II banks $E0 and $E1.

## 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. Figure 3-6 shows the format of the Speed register, followed by 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 Chapter 1, "Introduction to the Apple IIGS".

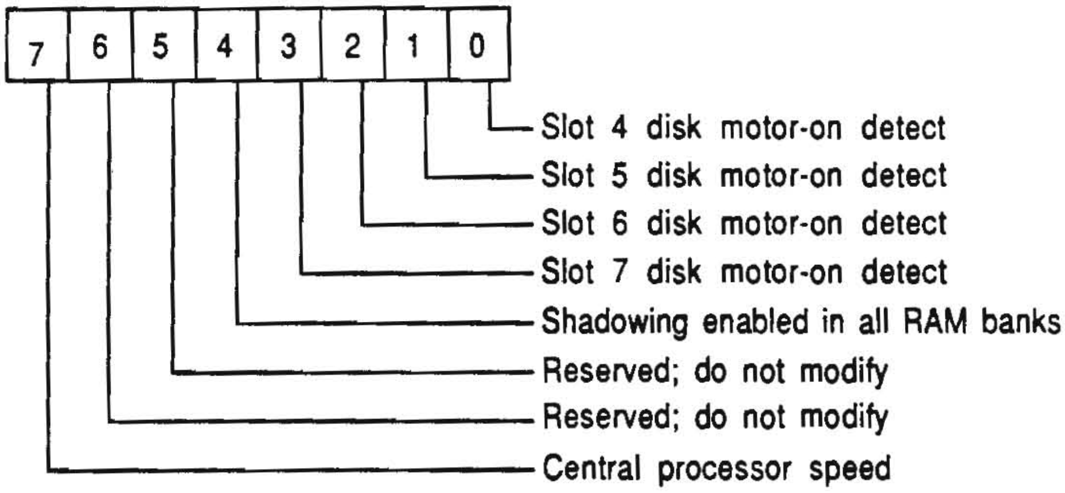

7
6
5
4
3
2
1
0
Slot 4 disk motor-on detect
Slot 5 disk motor-on detect
Slot 6 disk motor-on detect
Slot 7 disk motor-on detect
Shadowing enabled in all RAM banks
Reserved; do not modify
Reserved; do not modify
Central processor speed

**Figure 3-6. Speed register at $C036**

| 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 an Apple II). |
| 5–6 | - | 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 portions, 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. |
| 0–3 | 1 | Disk II motor address detectors: To retain Apple II peripheral compatibility, the motor-on detectors slow the system 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 slow mode (1.024 MHz) when address $C0D9 is accessed, and returns to normal speed (2.8 MHz) following a $C0D8 access. (See list of addresses below.) |
| | 0 | When this bit is 0, the disk II motor detector is turned off. |

Bits 0 through 3 detect the following address:

| Slot | Motor-on | Motor-off |
|---|---|---|
| 4 | $C0C9 | $C0C8 |
| 5 | $C0D9 | $C0D8 |
| 6 | $C0E9 | $C0E8 |
| 7 | $C0F9 | $C0F8 |

*Note*: 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 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.

*Note*: Drives designed for previous Apple II computers will function as Apple IIGS peripherals only if the system speed is slowed before disk access is attempted.

*Note*: For compatability with future Apple products, use firmware calls only to manipulate bits 0 to 3 of the Speed register.

## RAM control

The FPI alone controls the high-speed RAM. This high-speed memory consists of a minimum of 128K RAM on the main logic board and additional expansion RAM on the extended memory card for a total of 8Mb.

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 µs 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.

## ROM

The FPI provides control for 128K of on-board ROM and additional expansion-card ROM for a total of 1Mb. The Apple IIGS on-board system ROM is located in banks $FE through $FF. Banks $F0 through $FD are reserved for ROM expansion. ROM that occupies this address space may reside on the extended memory card only, along with additional expansion RAM.

## I/O processing

Normally, all I/O write accesses are to the designated I/O space in bank $E0 or $E1, and are written to all shadowed I/O space in the FPI. However, when FPI internal registers (the direct memory access [DMA] Bank register, the Speed register, and the Shadow register) are accessed, or when the interrupt ROM addresses ($C071 through $C07F) are read, only the high-speed I/O space is written to. All reads access only the high-speed, shadowed I/O space, eliminating the need to slow the system speed

Direct memory access (DMA) is a means of providing fast I/O. A peripheral card in one of the expansion slots can require DMA. For more information on DMA, see Chapter 5, "Peripheral Expansion Slots."

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.

### The Slot register

The built-in Slot register, located at $C02D, is used to select which device is enabled for each of the seven logical slots. That device can be either the internal or the peripheral slot device. If the enable bit is 1, accesses for that slot ROM space ($Cnxx) are directed to the ROM on the slot 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 Control Panel (accessible by pressing and releasing the Apple-Control-Esc keys in sequence) also allows the user to select the appropriate device for each logical slot. The bit representations are shown in Figure 3-7, followed by a list of the bit descriptions.

*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 the existing Apple II products.

*Warning*: Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in Chapter 1, "Introduction to the Apple IIGS".

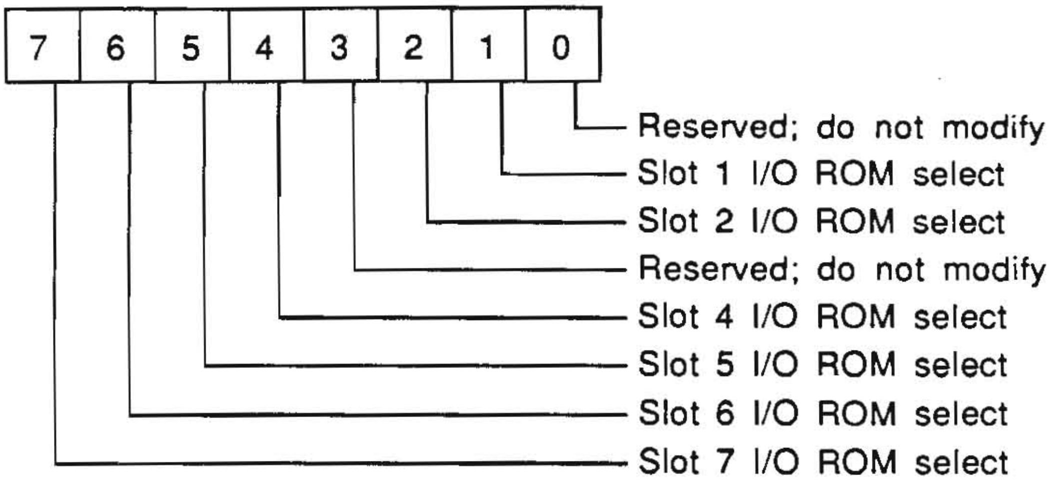


**Figure 3-7.** The Slot register at $C02D

| **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) 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) 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.

## Synchronization

Whenever data need to be transferred between the FPI and the Mega II, the FPI IC must first synchronize itself with the slower-running Mega II. This may involve a single Mega II cycle, as when a single I/O location in the Mega II must be accessed, or consecutive Mega II cycles (extended periods of low-speed operation), as when Apple II software must be run at the lower speed for compatibility. The FPI runs the processor at low speed by generating one processor cycle for each Mega II cycle, thus running the processor at precisely 1.024 MHz. This speed is necessary to support time-dependent Apple II software.

In all Apple II computers, every sixty-fifth processor cycle is elongated, or stretched by 140 nanoseconds. This practice is required to keep the video display consistent.

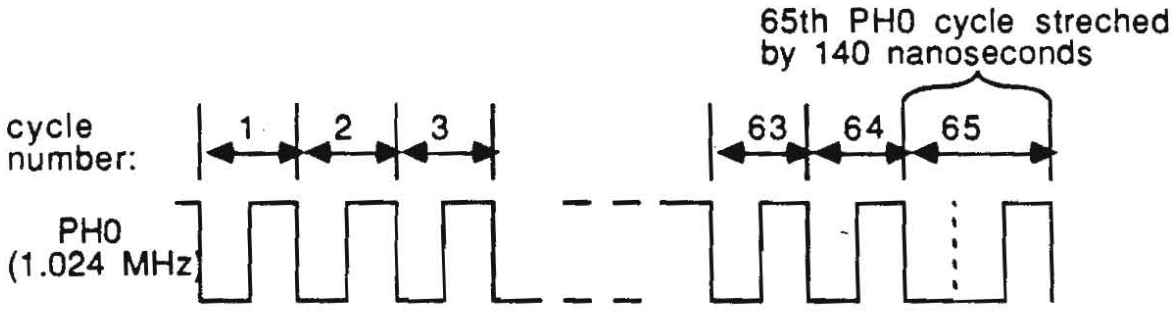


**Figure 3-8**. PH0 cycles, 14M cycles, and M-States

## The Mega II cycle

A Mega II cycle is a central processor or DMA cycle that requires access to the low-speed side of the system. These are:

- all external and most internal I/O operations
- shadowed video write operations
- *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 slow side of the system, such as listed above.

2. Approximately 90 nanoseconds after the processor PH2 clock 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 PH2 clock rises.

3. If the cycle is a Mega II cycle, the FPI holds the PH2 clock high until it synchronizes itself with the Mega II.

4. Memory or I/O access begins.

## Mega II auxiliary bank access

To allow direct access to the Mega II auxiliary 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 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 this setup to work, the programmer must first set bit 0 in the Video-Control register at $C029 to 1 (see Chapter 4, "Video," to read about the Video-Control 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.

## Real-time clock IC interface

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, nor should it be used by programs other than the system.

The control register (located at $C034), shown in Figure 3-9, serves a dual function: as the command register for the RTC and as the Border Color register. Refer to "Screen 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 compatable with future Apple II products, use the firmware calls to read and write data to the RTC. See the *Apple IIGS Firmware Reference* manual 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 Chapter 1, "Introduction to the Apple IIGS".

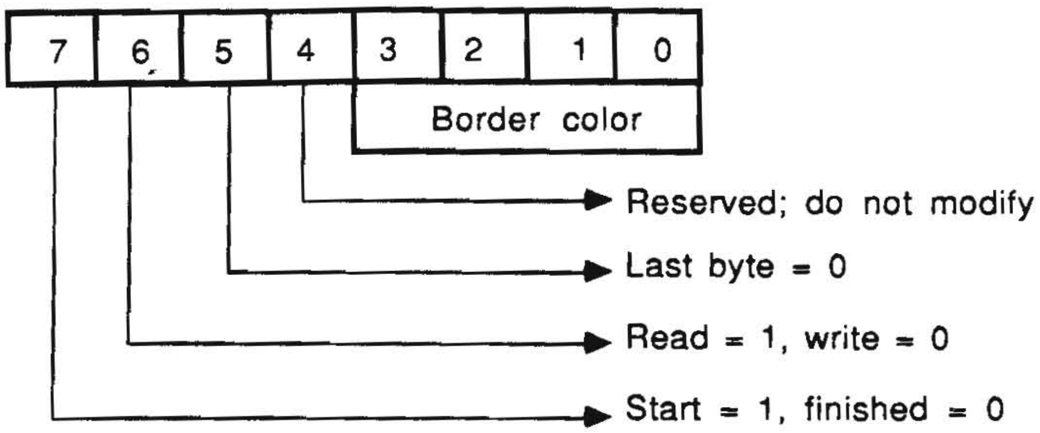


**Figure 3-9.** Control register at $C034

| 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 | - | Display border color: See "Text and Background Color" in Chapter 4, "Video," for details on selecting the video display border color. |

# Chapter 4

# New Video

The Apple IIGS can display several video modes. These include display modes that are compatible with the rest of the Apple II family (as well as some enhancements to these existing modes) and some completely new display modes. These new video modes provide higher resolution, greater color flexibility, and programming ease previously unseen in the Apple II product line. Figure 4-1 shows a block diagram of the Apple IIGS and the relationship of the video components within the system.

This chapter describes:

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

## The Video Graphics Controller

The Video Graphics Controller (VGC) custom IC is responsible for generating all video displayed by the Apple IIGS. The VGC provides these functions:

- supports and enhances 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, while the Mega II is responsible for maintaining the video RAM. All writes to the video display buffers in bank $E0 and $E1 are done via the Mega II. Figure 4-1 shows the relationships of the VGC, Mega II, main, and auxiliary RAM.

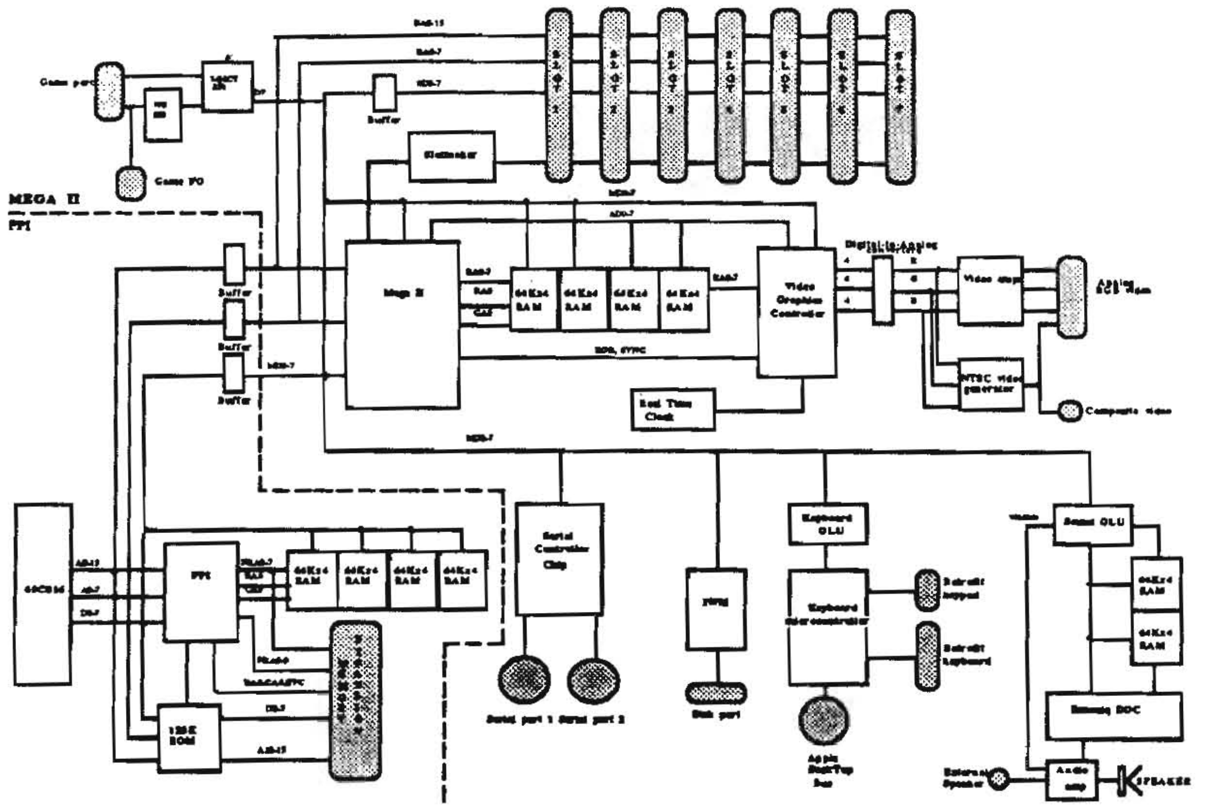


**Figure 4-1**. Diagram of the Apple IIGS and video components

## The RGB video connector

Located at the rear of the main logic board is the RGB video connector. An analog RGB video monitor may be connected to this connector. Figure 4-1 shows the pin diagram of this connector, and table 4-1 describes the signal associated with each pin.

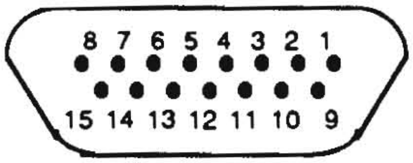


**Figure 4-2**. The RGB video connector

| Pin | Description | |
|---|---|---|
| 1 | Ground | Ground reference and supply |
| 2 | Red | Red analog video signal |
| 3 | Comp | Composite synch signal |
| 4 | N.C. | No connection |
| 5 | Green | Green analog video signal |
| 6 | Ground | 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 | Ground | Ground reference and supply |
| 14 | N.C. | No connection |
| 15 | N.C. | No connection |

**Table 4-1.** The RGB video signals

## Apple II compatibility

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

For more information on Apple II-compatible video, refer to Chapter 2, "The Mega II: Maintaining Compatibility."

Enhancements to the existing Apple II video modes include:

- the ability to select unique text and background colors from any of the 16 Apple II colors listed in Table 4-2.
- the ability to select the border color for the perimeter of the video image. You can choose this color from any of the 16 Apple II colors listed in Table 4-2.
- the ability to display gray-scale video. This means that you can display color video output on monochrome monitors in shades of gray rather than in dot patterns that represent color. This ability increases contrast between graphics colors on a monochrome monitor.

**Table 4-2.** Text and background colors

| | |
|---|---|
| $0 | Black |
| $1 | Deep red |
| $2 | Dark blue |
| $3 | Purple |
| $4 | Dark green |
| $5 | Dark gray |
| $6 | Medium blue |
| $7 | Light blue |
| $8 | Brown |
| $9 | Orange |
| $A | Light gray |
| $B | Pink |
| $C | Green |
| $D | Yellow |
| $E | Aquamarine |
| $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 4 lines of text at the bottom of the display will exhibit color fringing on composite color monitors.

*Color fringing* is the rainbow-like effect that appears around text characters when they are displayed in color on most color monitors. This fringing is unavoidable because the color detection circuitry of most composite color monitors cannot respond fast enough to the changing of the color information during the text portion of the display. Displaying text in black and white makes it more readable.

## New video 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 below. The new graphics modes—Super Hi-Res graphics and color-fill graphics—are described in the next section.

### Text and background color

The Apple IIGS provides the capability of colored text on a colored background. To select one of these new display options, 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 4-2. The user can also select these colors from the Control Panel. Figure 4-3 shows the format of the Screen Color register, followed by a description of each bit in the register.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Text | | | | Background | | | |

**Figure 4-3.** Screen Color register at $C022

| 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 Screen Border register located at $C034. You can choose this color from the 16 Apple II colors listed in Table 4-2. Alternately, the user can select the border color from the Control Panel.

The Border Color register is an 8-bit read/write register serving 2 functions. First, the least significant 4 bits determine the border color. Second, the most significant 3 bits are the control bits for the real time clock chip interface logic. Figure 4-4 shows the Border Color register format, followed by a description of each bit.

*Note*: When you change the border color by writing to the Border Color register, use values between $0 and $0F when writing to this register. This will ensure that the RTC chip contents remain unaltered.

See the section on the Real Time Clock Interface in Chapter 3, "New Features," for more information on the RTC.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RTC interface | | | | Border | | | |

**Figure 4-4.** Border Color register at $C034

| Bit | Value | Description |
|---|---|---|
| 7–4 | - | Real -time clock control bits (see "Real-Time Clock Interface" in Chapter 3) |
| 3–0 | - | Border color |

## To color or not to color...

The Apple IIGS video is displayed in either color or black and white. Figure 4-5 shows the format of the Monochrome/Color register, followed by a description of each bit. Located at $C021, this 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 this bit to 1. Displaying text in black and white results in a better-looking, more readable display. The remaining bits are reserved; do not modify them when writing to this location. You can also select color or monochrome video from the Control Panel. Figure 4-5 shows the format of the Monochrome/Color register, followed by a description of each bit in the register.

*Important:* This bit does not affect the RGB outputs.

*Warning*: Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in Chapter 1, "Introduction to the Apple IIGS".

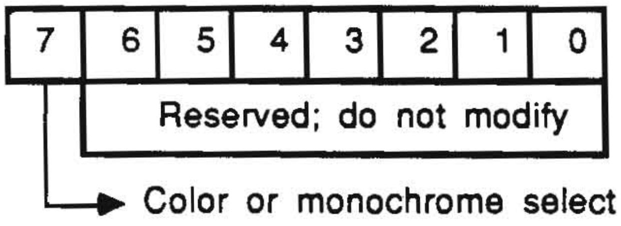


**Figure 4-5.** Monochrome/Color register at $C021

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

## New graphics display modes

The Apple IIGS has 2 video 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.

A **pixel** is the smallest individually addressable video or picture element (hence the word *pixel*). The Apple IIGS video screen displays different quantities of pixels, depending on the video mode. In Super Hi-Res graphics 640-mode, the screen contains 128,000 pixels (640 pixels on each of 200 lines). In 320-mode, half that number of pixels are displayed.

Another new 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 mode lets you draw consecutive pixels on a scan line in the same color faster and much more conveniently than previously possible. Color-Fill is available only in 320-Super Hi-Res graphics.

## Super Hi-Res graphics

The Apple IIGS uses Super Hi-Res graphics to implement new video graphics features previously unavailable in the Apple II family of computers. The VGC is primarily responsible for supporting the Super Hi-Res video graphics, which provide these new video capabilities:

- 320- or 640-horizontal resolution selectable
- 200-line vertical resolution
- 12-bit color resolution that allows 4096 available colors to choose from
- 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 Super Hi-Res graphics buffer

The Super Hi-Res graphics display buffer contains 3 types of data: pixel data, scan-line control bytes, and color palettes. Figure 4-6 shows a memory map of the display buffer. This buffer resides in contiguous bytes of the auxiliary 64K bank of the slow RAM 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 paragraphs describe the color palettes, scan-line control bytes, and pixel data bytes used in Super Hi-Res graphics mode.

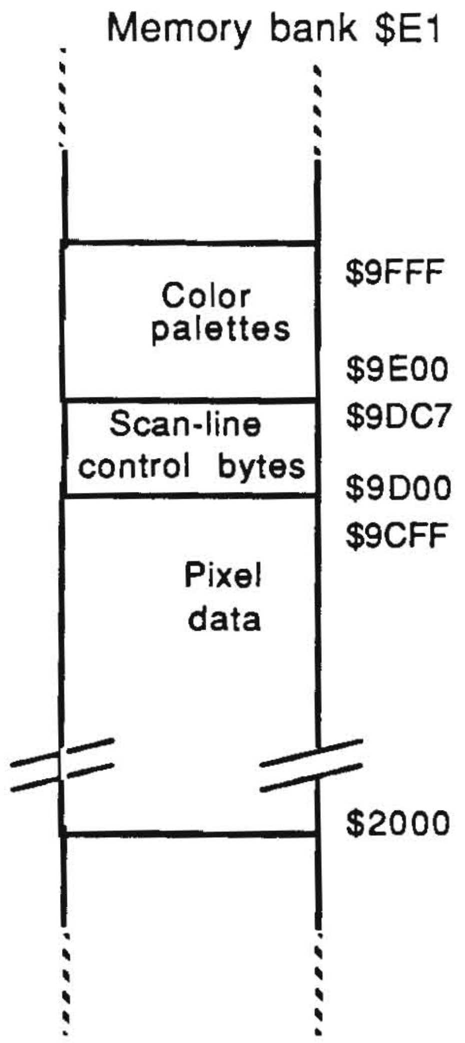


**Figure 4-6.** Super Hi-Res graphics display buffer

### 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 4-7) 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 4-7, and a description of each follows.

*Warning*: Be careful when changing bits within this byte. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in Chapter 1, "Introduction to the Apple IIGS".

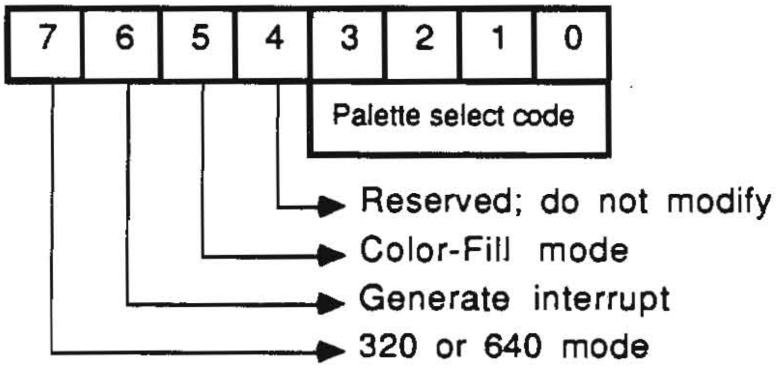


**Figure 4-7**. Scan-line control byte format

| Bit | Value | Description |
|---|---|---|
| 7 | 1 | Horizontal resolution = 640 pixels |
| | 0 | Horizontal resolution = 320 pixels |
| 6 | 1 | Interrupt generated 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 320-pixel resolution mode only. In 640-pixel mode, Color-Fill mode is disabled.) |
| | 0 | Color-Fill mode disabled |
| 4 | - | Reserved; do not modify |
| 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 1 of 16 color palettes assigned to it. The 16 colors in each palette can be chosen from any of the 4096 colors available. You can draw each pixel on the scan line in any of the 16 colors that make up the palette.

These colors are determined by a 12-bit value made up of 3 separate 4-bit values. Each 4-bit quantity represents the intensity of each red, green, and blue. The combination of the magnitudes of each of the 3 primary colors determines the resulting color. Figure 4-8 shows the format of each of these 4-bit values that make up a palette color.

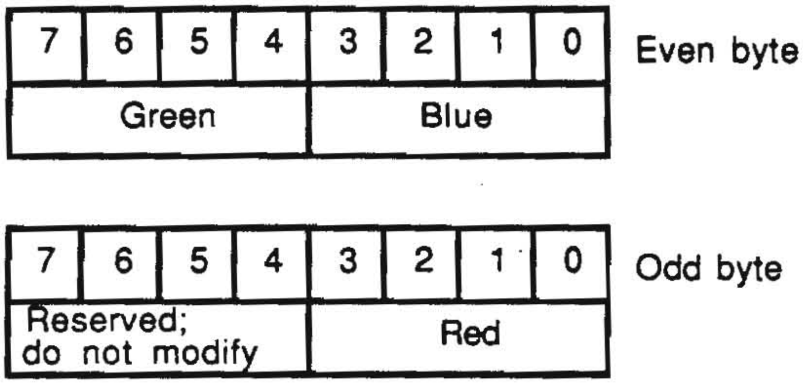


**Figure 4-8.** Color palette format

The color palettes are located in video buffer locations $9E00 through $9FFF. 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 4-3. 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 that you are going to display for each pixel.

**Table 4-3.** 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 2 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 1 of 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 4-9 shows the format in which the pixel color data are stored in both the 320-pixel and 640-pixel modes.

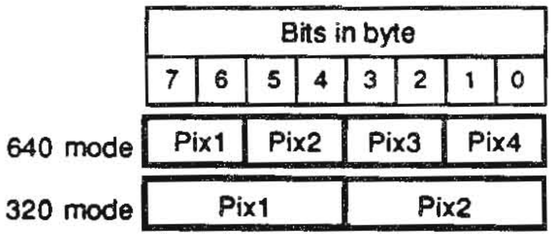


**Figure 4-9.** Pixel data byte format

In 320-pixel mode, 4 bits determine each pixel color, and data are stored 2 pixels to a byte of the display buffer. Since 4 bits determine the pixel color, in 320 mode each pixel can be any of the 16 colors from that palette.

In 640 mode, color selection is more complicated. The 640 pixels in each horizontal line occupy 160 adjacent bytes of memory, and each byte holds 4 pixels that appear side-by-side on the screen. The 16 colors in the palette are divided into 4 groups of 4 colors each. The first pixel in each horizontal line can select 1 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 Figure 4-10. The process repeats for each successive group of 4 pixels in a horizontal line. Thus, even though a given pixel can be 1 of 4 colors, different pixels in a line can take on any of the 16 colors in a palette. Using a technique called *dithering*, software for 640 mode can take advantage of this color selection scheme to display 16-color graphics on the same screen with 80-column text.

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

| Pixel | Value | Palette |
|---|---|---|
| Px3 | 0 | Color1 |
| | 1 | Color2 |
| | 2 | Color3 |
| | 3 | Color4 |
| Px4 | 0 | Color5 |
| | 1 | Color6 |
| | 2 | Color7 |
| | 3 | Color8 |
| Px1 | 0 | Color9 |
| | 1 | Color10 |
| | 2 | Color11 |
| | 3 | Color12 |
| Px2 | 0 | Color13 |
| | 1 | Color14 |
| | 2 | Color15 |
| | 3 | Color16 |

**Figure 4-10.** Color selection in 640 mode

## 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 4-11, followed by 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 Chapter 1, "Introduction to the Apple IIGS".

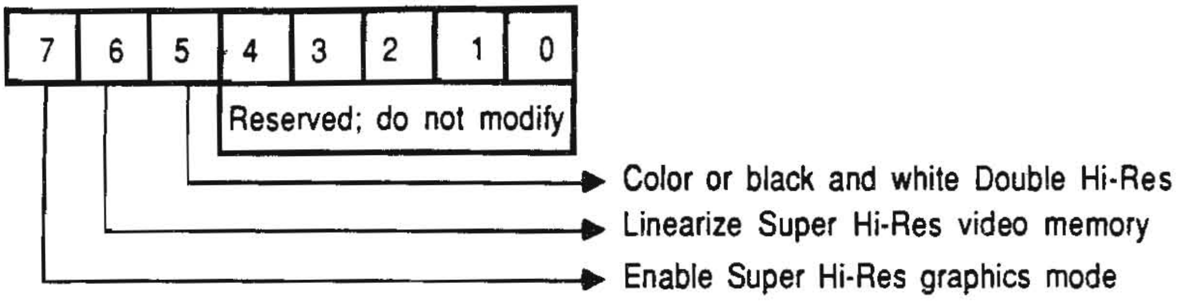

7
6
5
4
3
2
1
0
Reserved; do not modify
Color or black and white Double Hi-Res
Linearize Super Hi-Res video memory
Enable Super Hi-Res graphics mode

**Figure 4-11.** 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 video modes. If this bit is 1, all standard Apple II video modes are disabled; either 320- (and Color-Fill mode) or 640-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–$9D00 (Figure 4-6 shows the Super Hi-Res graphics buffer). |

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

*Note:* See Chapter 2: "The Mega II: Maintaining Apple II compatibility," for a description of the Apple IIe memory map.

| | | |
|---|---|---|
| 5 | 0 | If this bit is 0, Double Hi-Res graphics is displayed in color (280 x 192, 16 color). |
| | 1 | If this bit is 1, Double Hi-Res graphics is displayed in black and white (560 x 192). |
| 4–0 | - | Reserved; do not modify. |

### Color-Fill mode

Color-Fill mode, which is available in 320-pixel mode only, is used to rapidly fill 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 ($1–$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 a follows:

```
A00000000000B00000000000C00000000000
```

Method 2 would save time: The program only needs to fill the pixel area of the scan line once with 0, and then write a color value into those locations where a color should begin or change. In the example just given, only 3 bytes need be written to implement the 3 color areas on the scan line using the Color-Fill method, as opposed to 12 pixels 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.

## 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 one Hz input signal from the 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 mode only, and are not available in other video modes.

Figure 4-12 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.

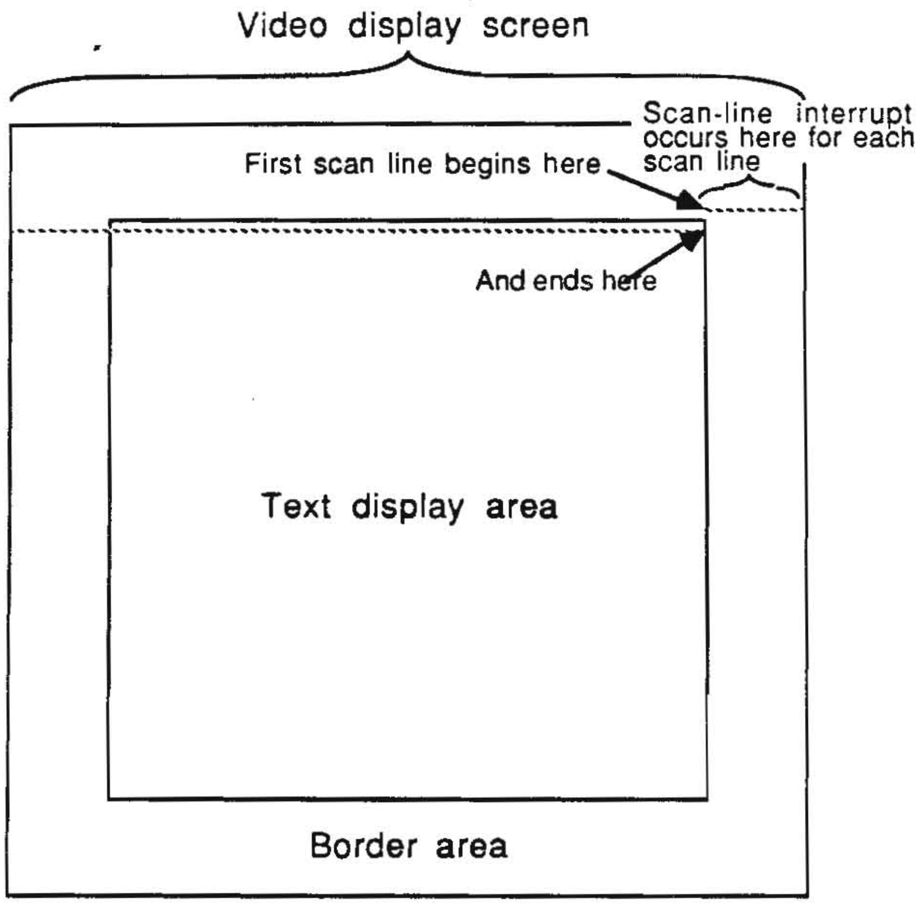

Video display screen
Scan-line interrupt occurs here for each scan line
First scan line begins here
And ends here
Text display area
Border area

**Figure 4-12.** Scan-line interrupt

## The VGC Interrupt register

The VGC Interrupt register ($C023) contains a status bit and an enable bit for each of the three 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 4-13 shows the format of the VGC Interrupt register and is followed by 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 Chapter 1, "Introduction to the Apple IIGS".

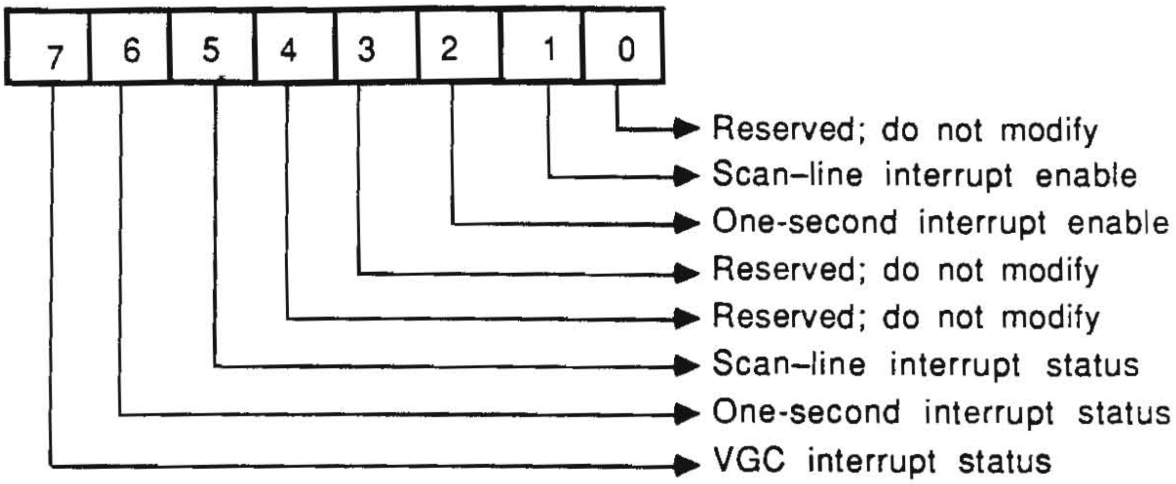

7
6
5
4
3
2
1
0
Reserved; do not modify
Scan–line interrupt enable
One-second interrupt enable
Reserved; do not modify
Reserved; do not modify
Scan–line interrupt status
One-second interrupt status
VGC interrupt status

**Figure 4-13.** VGC Interrupt register at $C023

| Bit | Value | Description |
|---|---|---|
| 7 | 1 | VGC interrupt status. This bit is set when the interrupt bit and the status bit are set for 1 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 shown in Figure 4-14. Writing a 1 into these positions or writing into the other bit positions has no effect. Figure 4-14 shows the format of the VGC Interrupt-Clear register, and is followed by 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 Chapter 1, "Introduction to the Apple IIGS".

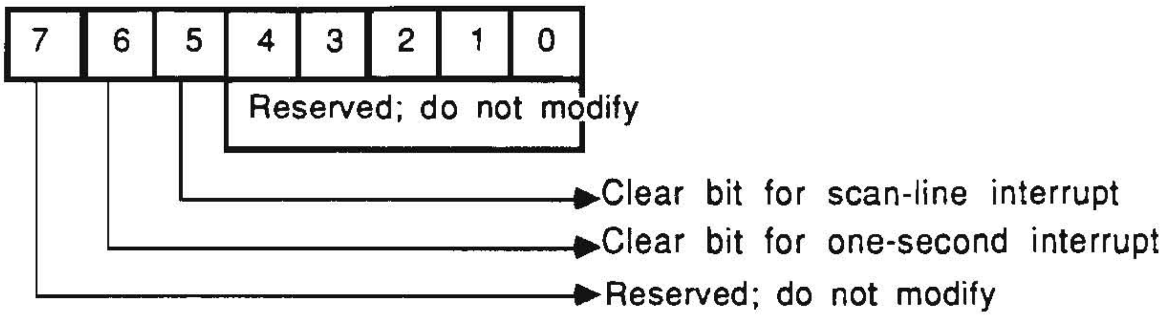

7
6
5
4
3
2
1
0
Reserved; do not modify
Clear bit for scan-line interrupt
Clear bit for one-second interrupt
Reserved; do not modify

**Figure 4-14.** VGC Interrupt-Clear register at $C032

| 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. |

## Graphics summary

The Apple IIGS supports all previous Apple II graphics modes, and provides enhancements to these modes. These are

- the ability to select unique text and background colors
- the ability to select the border color for the perimeter of the video image
- the ability to display gray-scale video

New graphics modes include:

- Super Hi-Res graphics mode in 320-pixel resolution
- Super Hi-Res graphics mode in 640-pixel resolution
- Color-Fill mode

There are 16 palettes, each palette containing 16 preselected colors, located in the palette area of the display buffer. Use these palettes to select the display colors for each pixel.

There are 200 scan-line control bytes (1 control byte per scan line), which determine:

- either 320-pixel mode or 640-pixel mode
- Color-Fill mode or regular display mode
- which of the 16 palettes in memory are to be used in this scan line
- whether the current scan-line interrupt is enabled or disabled

The pixel data bytes are then loaded with the color information for each pixel: There will be 4 bits per pixel in 320 mode, and there will be 2 bits per pixel in 640 mode.

Figure 4-15 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, 1 for each scan line, are shown at the right.

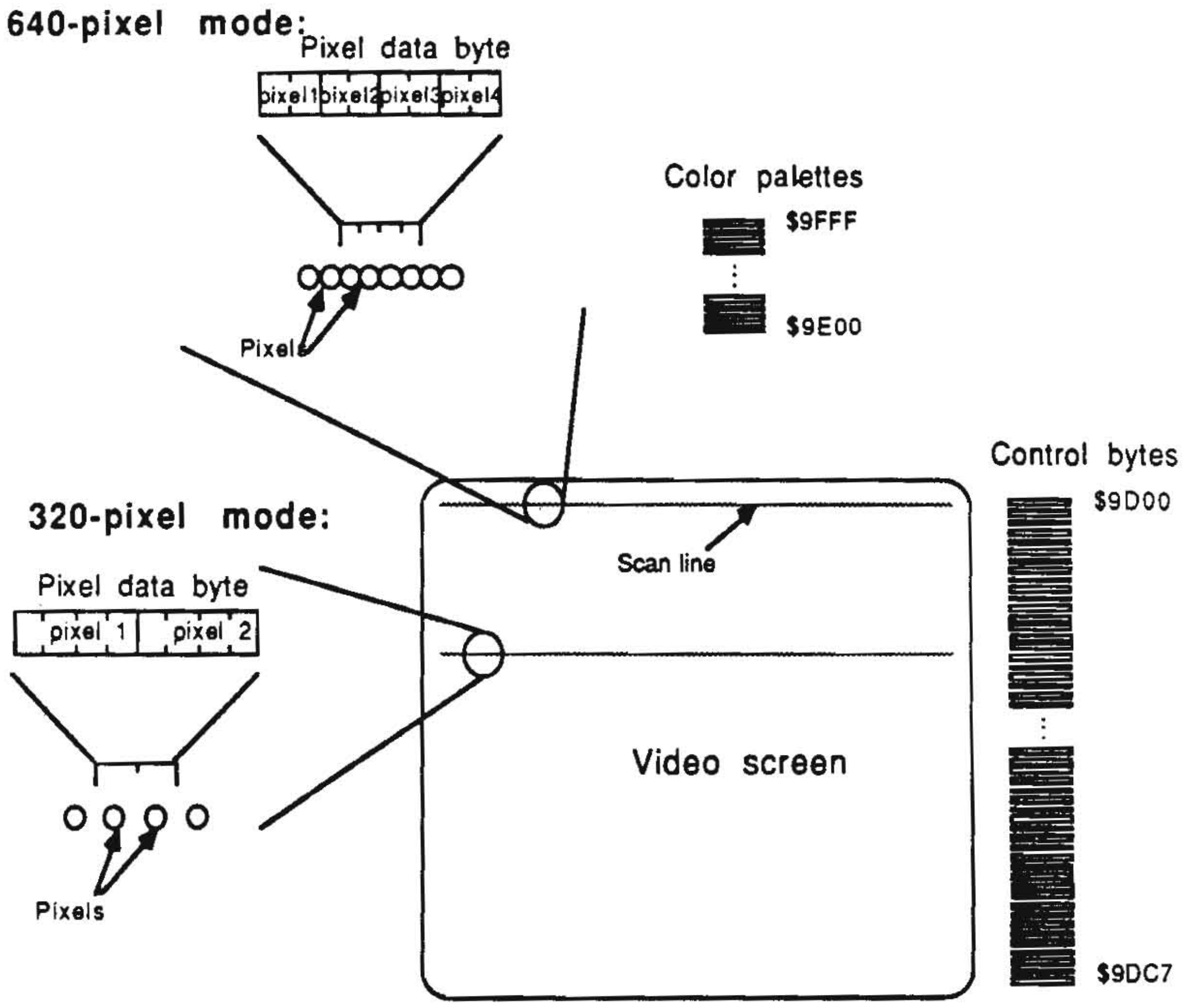


**Figure 4-15.** Drawing pixels on the screen

# Chapter 5

# Peripheral Expansion Slots

The main logic board of the Apple IIGS has seven empty 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 5-1 shows a block diagram of the Apple IIGS and the relationship of the slots in the computer.

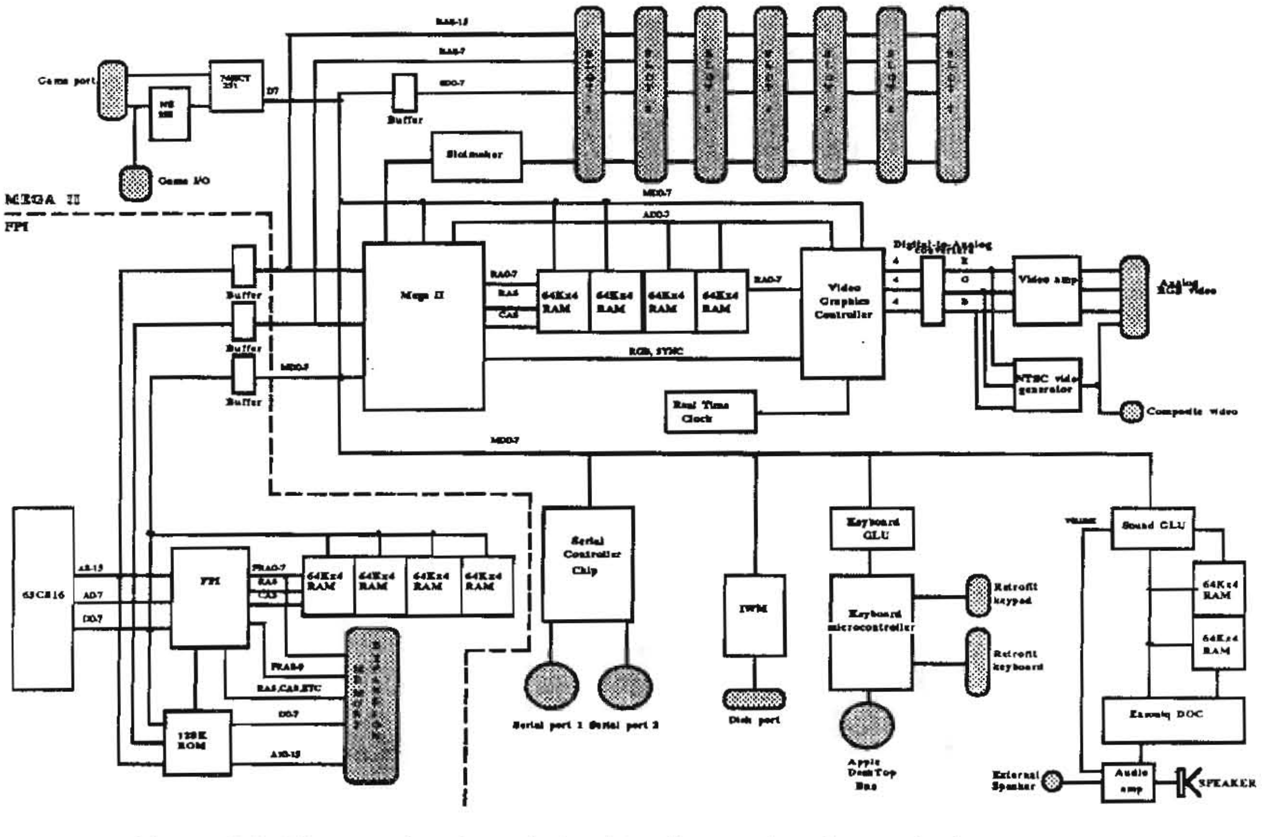


Figure 5-1. Diagram showing relationship of expansion slots and other components

*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 Chapter 9, "Memory Expansion Slot," for a description of this slot.

## The expansion slots

The 7 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 shown in Figure 5-2 and are described briefly in Table 5-1.

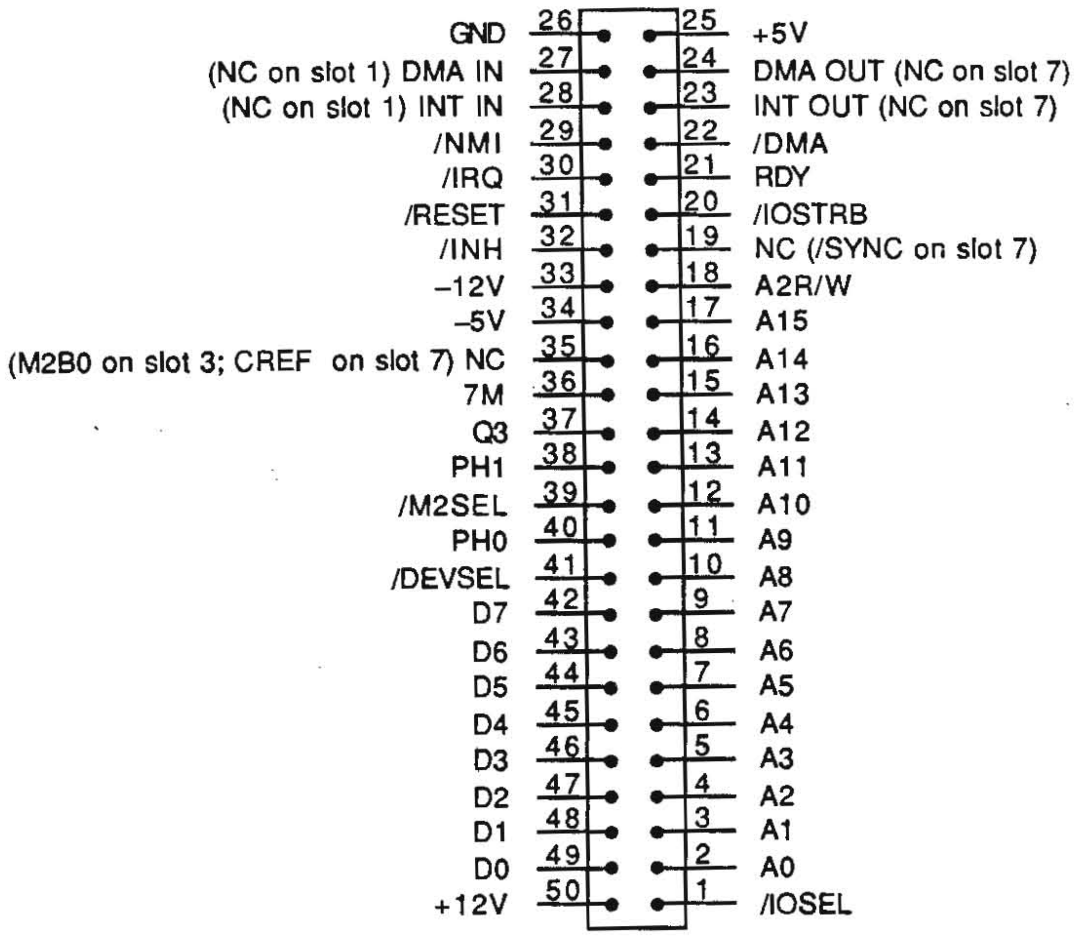


**Figure 5-2.** Peripheral expansion slot pin diagram

**Table 5-1**
Expansion slot signals

| Pin | Signal | Description |
|---|---|---|
| 1 | /IOSEL | Normally high; goes low during Φ0 when the 65C816 addresses location $Cnxx, where *n* is the connector number. This line can drive 10 LS (low-power Schottky) TTL (transistor-transistor logic) loads.* |
| 2–17 | A0–A15 | Three-state address bus. The address becomes valid during Φ1 and remains valid during Φ0. 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 line can drive 2 LS TTL loads.* |
| 20 | /IOSTRB | Normally high; goes low during Φ0 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 Φ1 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 +5V. |
| 22 | /DMA | Input to the address bus buffers. Pulling this line low during Φ1 disconnects the 65C816 from the address bus. This line has a 3300 ohm pullup resistor to +5V. |
| 23 | INT OUT | Interrupt priority daisy-chain output. Usually connected to pin 28 (INT IN). |
| 24 | DMA OUT | DMA priority daisy-chain output. Usually connected to pin 27 (DMA IN). |
| 25 | +5V | +5-volt power supply. A total of 500mA 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). |
| 28 | INT IN | Interrupt priority daisy-chain input. Usually connected to pin 23 (INT OUT). |

| | | |
|---|---|---|
| 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 +5V. |
| 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 +5V. |
| 31 | /RES | Pulling this line low initiates a reset routine. |
| 32 | /INH | Pulling this line low during Φ1 inhibits (disables) the memory on the main circuit board. This line has a 3300 ohm pullup resistor to +5V. |
| 33 | –12V | –12 volt power supply. A total of 200mA is available for all peripheral cards. |
| 34 | –5V | –5 volt power supply. A total of 200mA is available for all peripheral cards. |
| 35 | CREF | 3.58 MHz color reference signal, on slot 7 *only*. This line can drive 2 LS TTL loads.* |
| 35 | /M2B0 | Mega II bank-0 signal, on slot 3 *only*. This signal goes low whenever the Mega II is addressing the main bank of Mega II RAM. |
| 36 | 7M | System 7 MHz clock. This line can drive 2 LS TTL loads.* |
| 37 | Q3 | System 2 MHz asymmetrical clock. This line can drive 2 LS TTL loads.* |
| 38 | Φ1 | Phase-1 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 | Φ0 | Phase-0 clock. This line can drive 2 LS TTL loads.* |
| 41 | /DEVSEL | Normally high; goes low during Φ0 when the 65C816 addresses location $C0nx, where *n* is the connector number plus 8. This line can drive 10 LS TTL loads.* |
| 42–49 | D0–D7 | Three-state buffered bidirectional data bus. Data become valid during Φ0 high and remain valid until Φ0 goes low. Each data line can drive 1 LS TTL load.* |

| | | |
|---|---|---|
| 50 | +12V | +12 volt power supply. A total of 250mA is available for all peripheral cards. |

*Loading limits are for each card.

## Apple II compatibility

The 7 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, 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 an I/O enable line. Since this memory space contains video buffers and I/O addresses, peripheral video cards can make extensive use of these 2 signals.

## Direct memory access

Direct memory access (DMA) supports the full 24-bit address range. This means that any peripheral card using DMA may have direct address control of all 8Mb of memory (main and expansion memory). This is accomplished by loading the DMA bank register with the upper 8 bits of the required 24-bit address.

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. At this time, 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 R/W is high. If the access is to the high-speed RAM, the data buffers are enabled while Φ0 is high.

> *Note*: To increase read/write data timing margins to the high-speed RAMs, the FPI generates an early CAS 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.

## 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, not having 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 INHIBIT will work properly if

- the system is running at 1 MHz
- they assert inhibit within 200 nanoseconds of the Φ0 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 INHIBITs that occur 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. This improves compatibility with existing cards.

## 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 200 nanoseconds of the Φ0 rising edge will probably work properly; this allows sufficient time for the Mega II select line to be activated by the FPI when video and I/O accesses are required.

*Note*: Normally the system should be running slowly 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 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 consititute 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 diagram in Appendix D.

### 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. This disables the address and R/W lines 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 Φ0 of the instruction cycle, as shown in Figure 5-3.

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 Φ0 of the instruction cycle (refer to the timing diagram in Figure 5-4).

The peripheral devices should use /IOSEL and /DEVSEL as enables. Most peripheral 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 only asserted during Φ0 high.

### The slot data bus

The Apple IIGS has three versions of the microprocessor data bus (see Figure 5-3):

- 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), keyboard and sound general logic units (GLUs) and the Mega II RAM main bank
- the slot data bus, SDBUS, common to all expansion slots

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 7 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 argument applies 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 2 LS TTL loads per slot pin, plus additional capacitance for the Apple IIGS data bus.

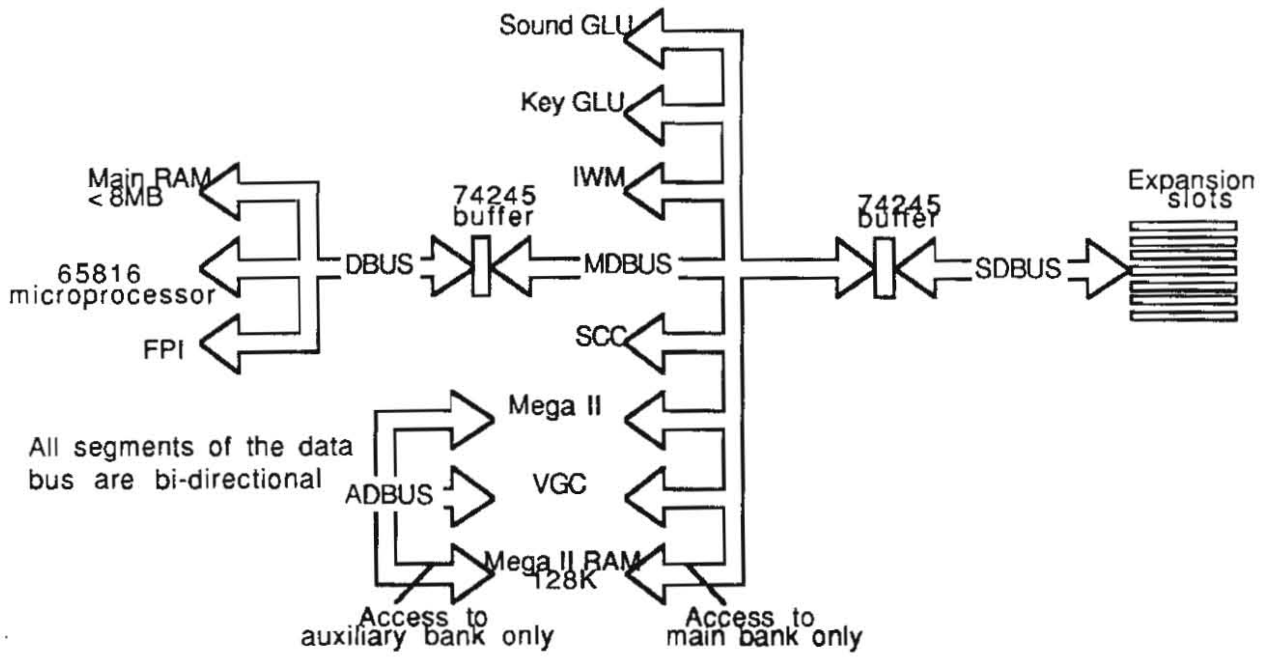


**Figure 5-3.** The data buses within the Apple IIGS

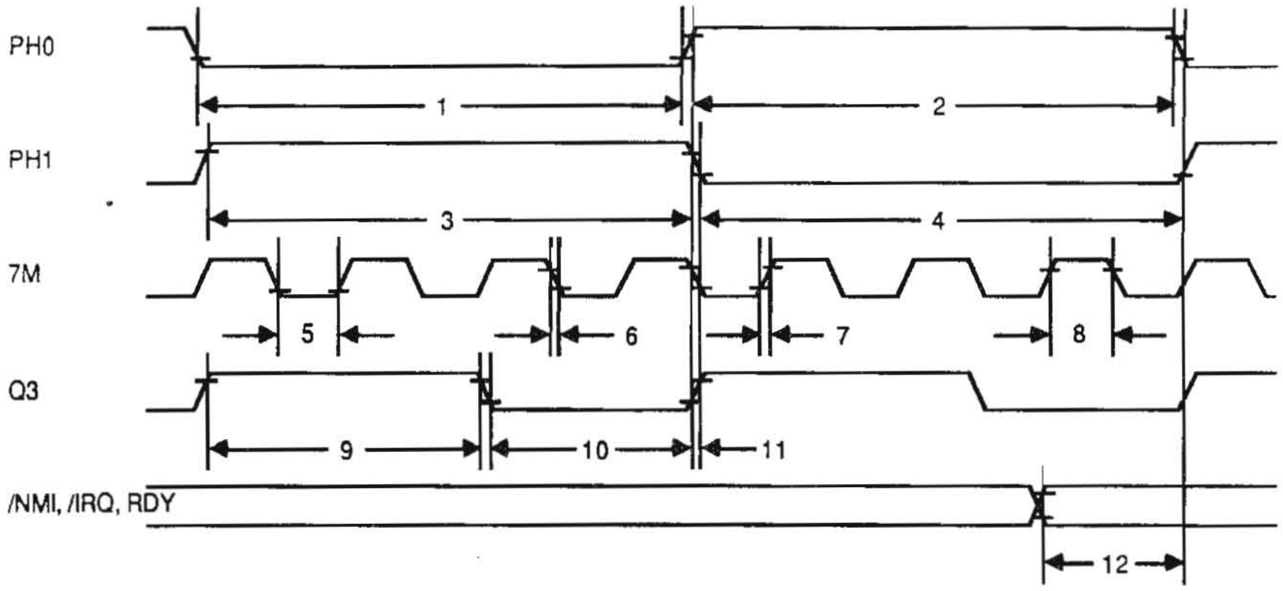


**Figure 5-4.** Input/output clock and control signal timing

| Number | Description | Min. (ns) | Max. (ns) |
|---|---|---|---|
| 1 | Φ0 low time | 480 | |
| 2 | Φ0 high time | 480 | |
| 3 | Φ1 high time | 480 | |
| 4 | Φ1 low time | 480 | |
| 5 | 7M low time | 60 | |
| 6 | Fall time, all clocks | | 10 |
| 7 | Rise time, all clocks | | 10 |
| 8 | 7M high time | 60 | |
| 9 | Q3 high time | 270 | |
| 10 | Q3 low time | 200 | |
| 11 | Skew, Φ0 to other clock signals | -10 | 10 |
| 12 | Control signal setup time | 140 | |

*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 board because of this buffering. All timing parameters in the timing diagrams in this chapter have been adjusted to account for this delay.

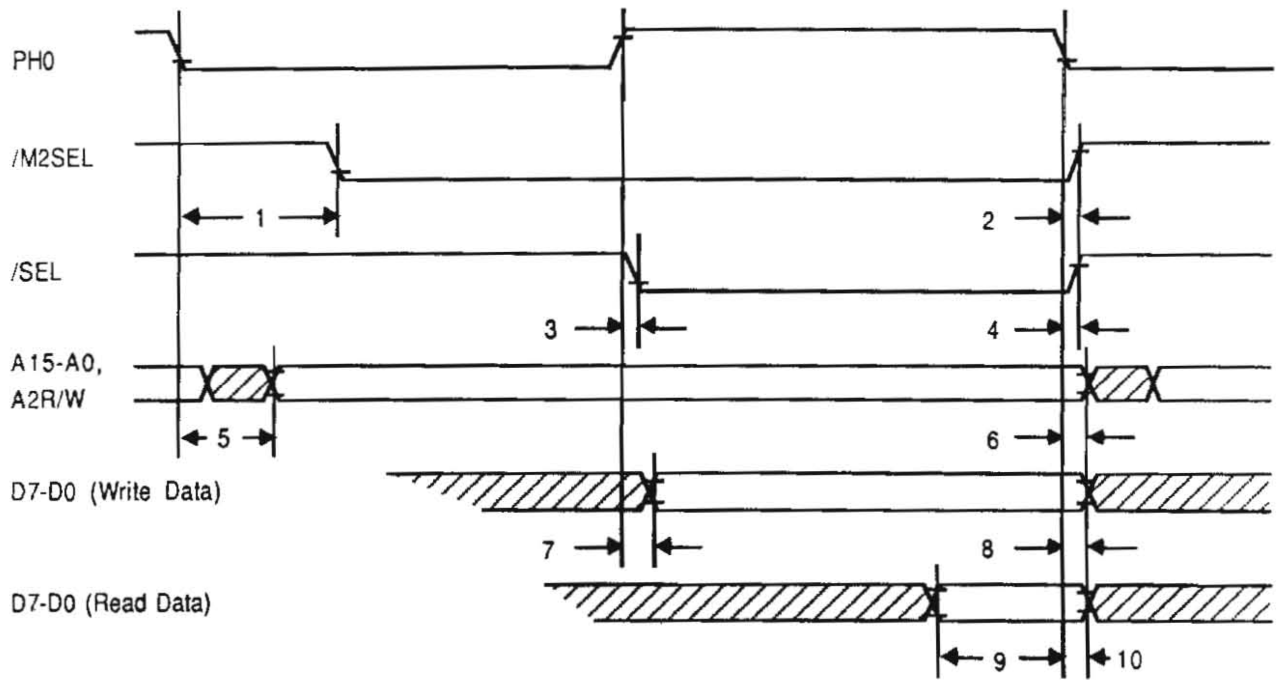


**Figure 5-5.** Slot I/O read and write timing

| Number | Description | Min. (ns) | Max. (ns) |
|---|---|---|---|
| 1 | /M2SEL low from Φ0 low | | 160 |
| 2 | /M2SEL hold time | -10 | |
| 3 | I/O enable low from Φ0 high (DEVn,/IOSELn,/IOSTRB) | | 15 |
| 4 | I/O enable high from Φ0 low (DEVn,/IOSELn,/IOSTRB) | 10 | |
| 5 | Address and A2R/W valid from Φ0 low | | 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 Φ0 | 140 | |
| 10 | Read data hold time | 10 | |

The standard Apple IIGS slot I/O timing is shown in Figure 5-5. 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 Φ0 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 slow, synchronized memory cycle is taking place and, therefore, when the value on the address bus will remain valid during the current Φ0 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.

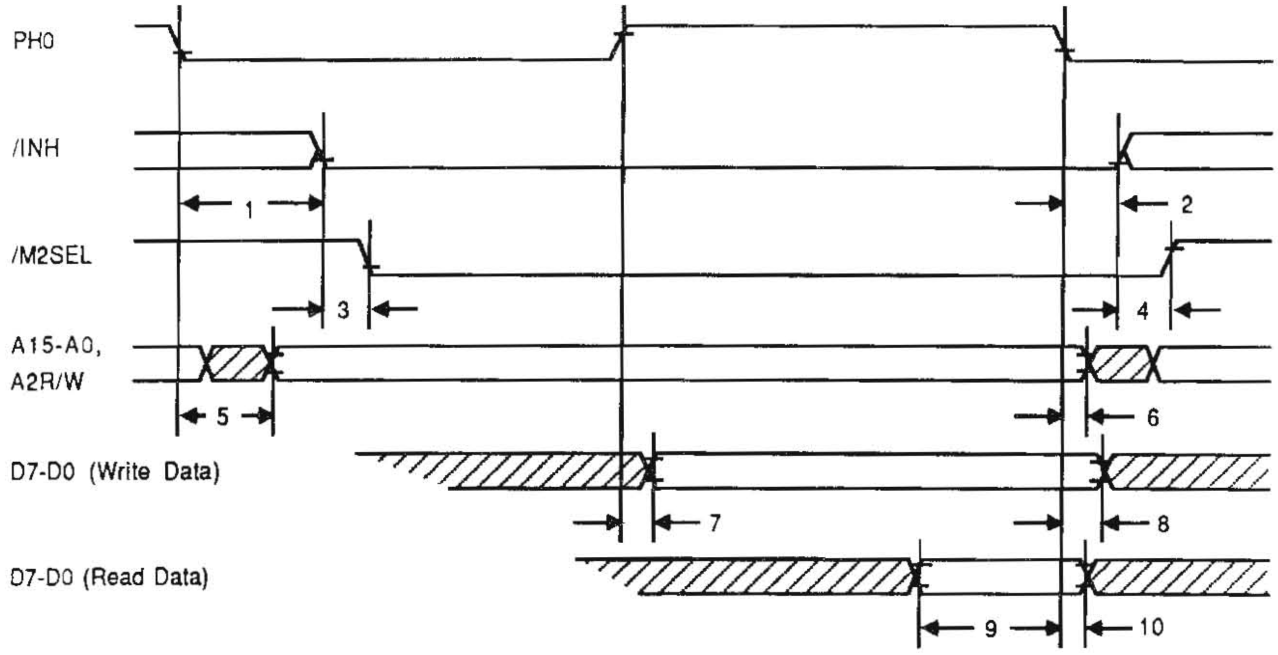


**Figure 5-6.** I/O read and write timing with /INH active

| Number | Description | Min. (ns) | Max. (ns) |
|---|---|---|---|
| 1 | /INH valid after Φ0 low | | 175 |
| 2 | /INH hold time | 15 | |
| 3 | /INH low to /M2SEL low delay | | 30 |
| 4 | /INH high to /M2SEL high delay | | 30 |
| 5 | Address and A2R/W valid from Φ0 low | | 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 Φ0 | 140 | |
| 10 | Read data hold time | 10 | |

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. When /INH is asserted, the computer responds as if a Mega II memory cycle were being performed.

Cards that use the /INH signal will function properly only if the computer is running at slow speed (1 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 Φ0 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.

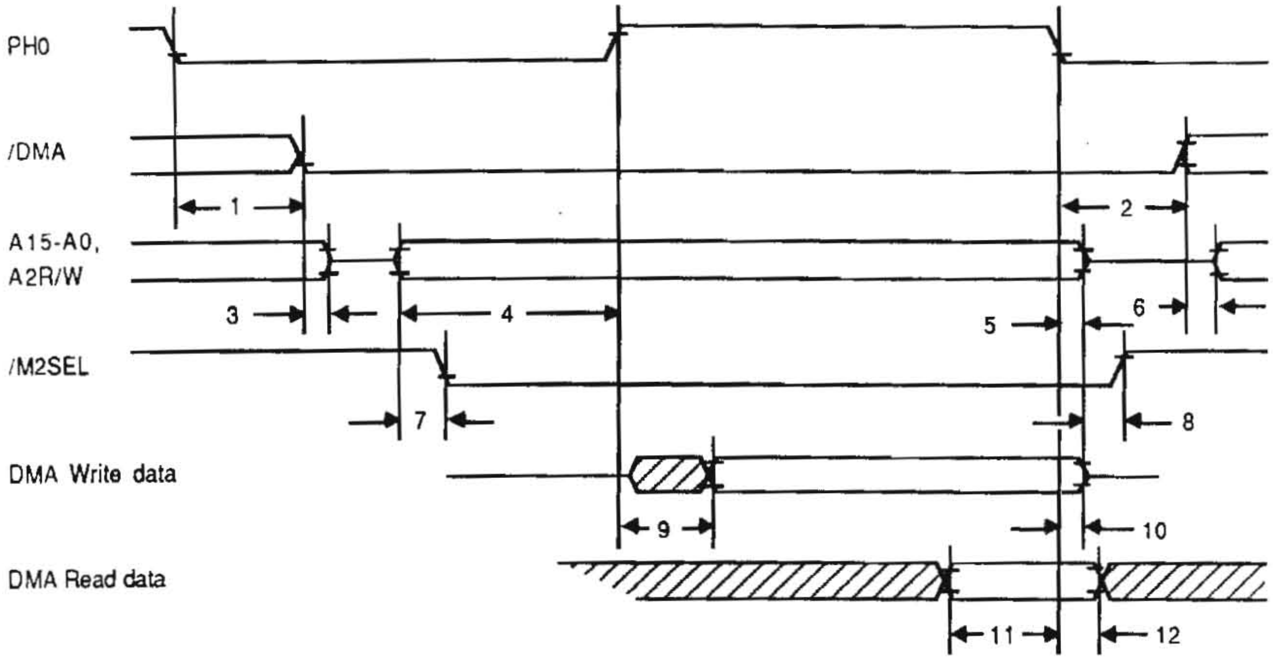


**Figure 5-7.** /DMA read and write timing

| Number | Description | Min. (ns) | Max. (ns) |
|---|---|---|---|
| 1 | /DMA low from Φ0 low | | 120 |
| 2 | /DMA high from Φ0 low | | 120 |
| 3 | A15-A0 and R/W float from /DMA | | 30 |
| 4 | DMA address and A2R/W valid before Φ0 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 | Φ0 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 | |

DMA devices will work in the Apple IIGS computer only in slow mode (1 MHz). 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 the low-speed memory (all I/O and video memory) must be done at low speed (1 MHz). To do this, 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.

### 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, 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 5-2.

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 5-1. 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

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

The support circuitry for the slots is designed to handle a DC load of 2 LS TTL loads per slot pin and an AC load of no more than 15 pF per slot pin.

## Summary

- The Apple IIGS expansion slots are almost identical to other Apple II expansion slots. The exceptions, signals /M2B0 and /M2SEL, indicate accesses to slow RAM banks $E0 and $E1, the location of I/O and video buffers.

- Expansion slot outputs are buffered to provide greater driving capability. Peripheral cards must use buffers when driving the bus on the Apple IIGS.
- The power supply provides power to the peripheral cards. This power is limited and must not be exceeded.
- The expansion slots are provided to allow an external device a means to connect to the internal data and address buses. DMA and interrupt requests are handled in a slot 7 to slot 1 priority fashion. A card in the higher numbered slot has priority when more than one device signals a request simultaneously.

# Chapter 6

# Sound

One of this 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.

## Sound synthesis

Sound is synthesized by programming digital oscillators to produce waveforms that simulate 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 outside analog input signal.

The Apple IIGS uses the Ensoniq® 5503 digital oscillator chip, a programmable sound synthesizer chip. This chip has 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. Figure 6-1 shows the relationship of the sound components to the rest of the computer.

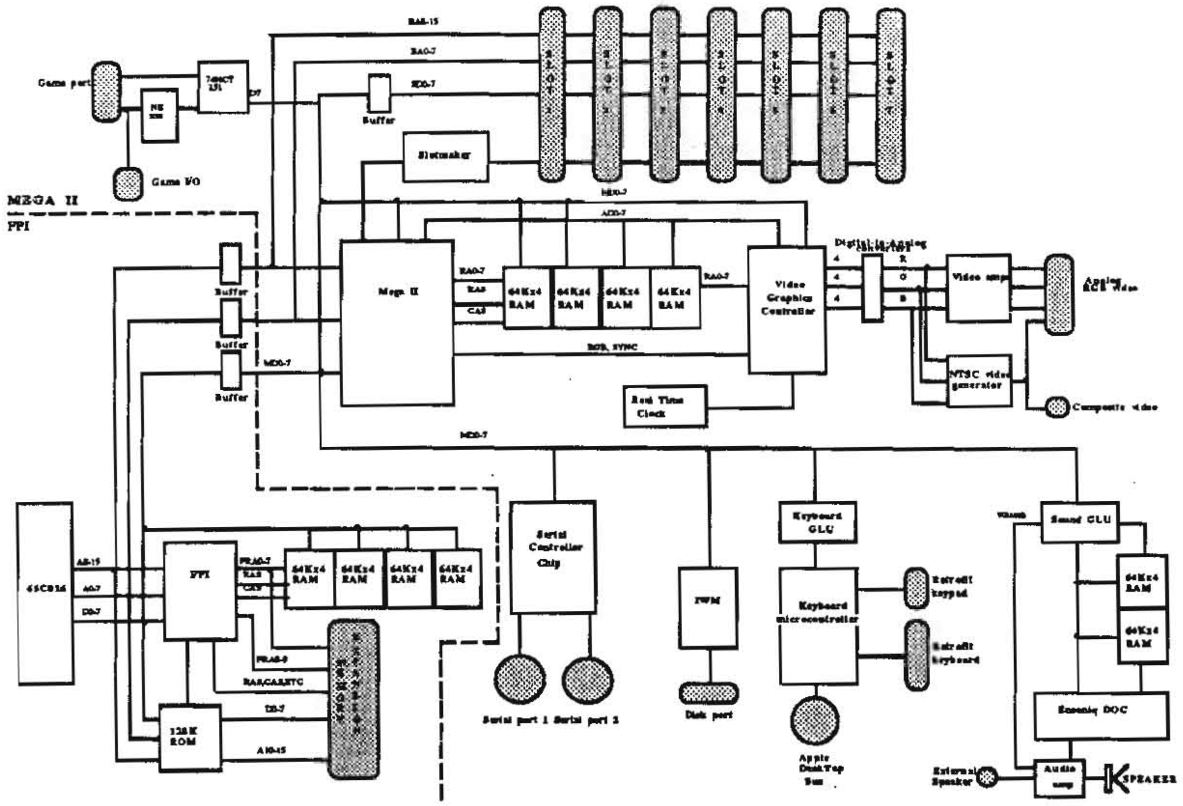


**Figure 6-1.** Diagram showing relationship of the sound components in the system

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* manual.

## Accessing the DOC

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 x 8 dynamic RAM. This interface allows the DOC chip to run independent of the rest of the system.

A *wavetable* is a series of contiguous data bytes in memory. The input signal is sampled, digitized, and placed in memory as a continuous wavetable. This wavetable will be used as data for the DOC which will result in sound output.

An alternate means of generating synthesized sound is to manually build a wavetable in memory, one byte at a time.

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 6-1 and described in detail in the sections that follow.

**Table 6-1.** 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 | $C03E | 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 6-2 shows the format of the Sound Control register, followed by a description of each bit.

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

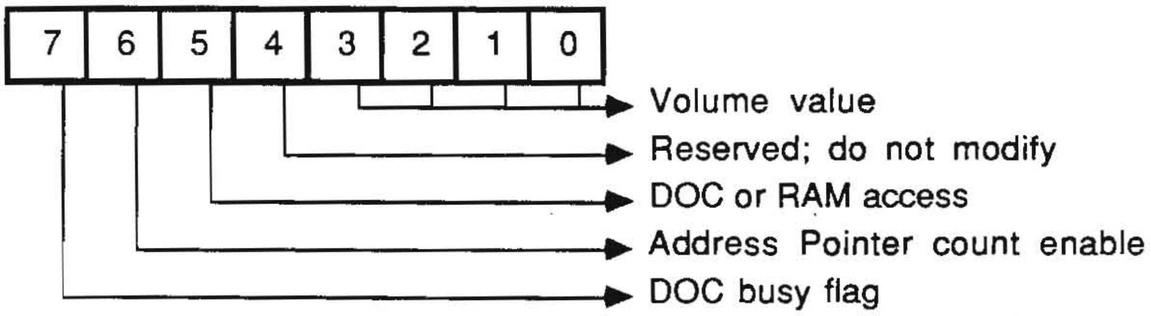


**Figure 6-2.** The Sound Control register at $C03C

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

## Address Pointer register

When accessing the sound RAM (bit 5 = 1 in the Sound Control register ), the Address Pointer register points 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 5 = 0 in the Sound Control register ), the Address Pointer register high byte is ignored by the DOC, and the low byte points to the DOC register to be written or read from. Figure 6-3 shows the format of the Address Pointer registers.

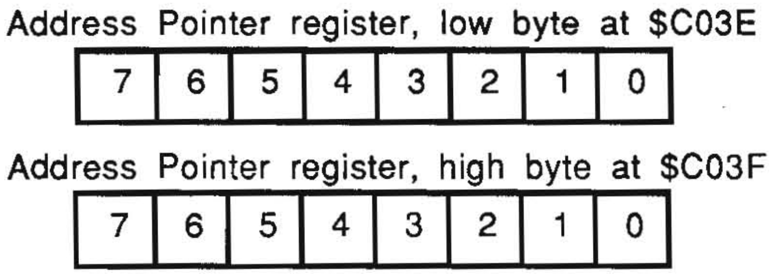

Address Pointer register, low byte at $C03E
7
6
5
4
3
2
1
0
Address Pointer register, high byte at $C03F
7
6
5
4
3
2
1
0

**Figure 6-3.** The 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 with the beginning location into which data are to be written.
3. 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 register after each write to the data register.

**Important** 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 1 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 control register bit and enabling auto-incrementing. Then set the address pointer to the starting address and read the data register 11 times, discarding the first byte read.

## The DOC registers

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

- the Oscillator Interrupt register
- the Oscillator Enable register
- the **Analog-to-Digital** (A/D) Converter register

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 register (low)
- a Frequency register (high)
- a Wavetable register
- an Address Pointer register (high)

### 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 Interrupt register is then set, and the register number is entered in bits 1 to 5 of the Oscillator Interrupt register. Figure 6-4 shows the format of the Oscillator Interrupt register, followed by a description of each of the bits.

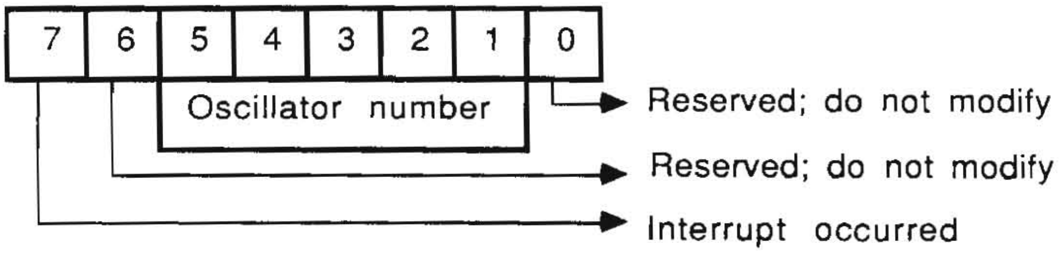


**Figure 6-4**. The Oscillator Interrupt register at $E0

| 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 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 by 2 and enter the number in this register. You may enter any number from 2 to 64 that 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.

### The A/D Converter register ($E2)

The A/D Converter register contains the output value of the 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 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.

## The Oscillator Control register ($A0–$BF)

Each Oscillator Control register controls all functions of each oscillator. This register controls:

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

Figure 6-5 shows the format of this register.

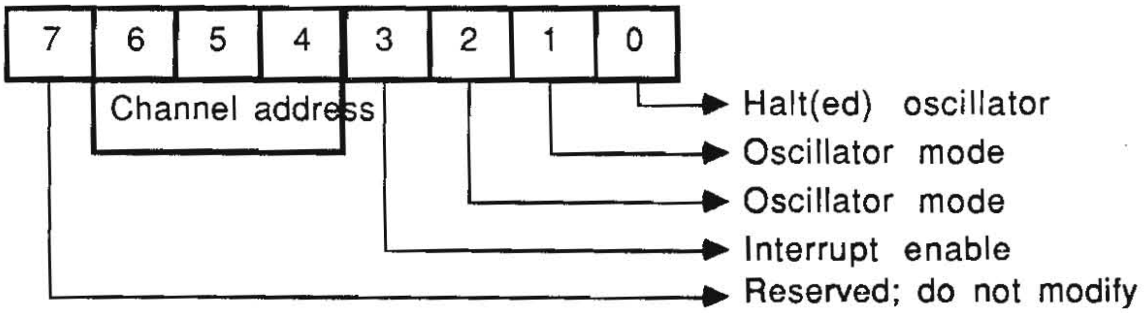


**Figure 6-5.** The Oscillator Control register

| Bit | Value | Description |
|---|---|---|
| 7 | - | Reserved; do not modify. |
| 6–4 | - | Channel address bits: These bits 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 8 separate sound channels. Figure 6-7 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 (OIR) and the DOC will assert the IRQ signal when an oscillator generates an interrupt. If the flag is already set (an oscillator interrupt is currently being handled), the flag is pushed onto a first-in, first-out buffer, and handled in that order. |
|  | 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. The oscillator may function in one of several available modes. Selected 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 |

Each of these modes is described below.

*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*: 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, also beginning its wavetable simultaneously.

*Swap mode*: Uses even/odd pairs of oscillators (a lower even-numbered oscillator paired with an adjacent higher-numbered oscillator). The enabled oscillator 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.

| | | |
|---|---|---|
| 0 | 1 | Halt(ed) 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 data register ($60–$7F)

The data register is a read-only register and contains the last byte read by the oscillator from the wavetable.

## The Volume register ($40–$5F)

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

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

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

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

where

$$\text{Scan rate (SR)} = 894.886 \text{ KHz} / (\text{OSC}+2)$$

and where RES is the resolution value in the Waveform 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 Waveform register ($C0–$DF)**

The Waveform register controls the size of the individual wavetable each oscillator will access. The size of this table may be between a minimum of 256 bytes and a maximum of 32K. Figure 6-6 shows the format of the Waveform register and is followed by a description of each bit.

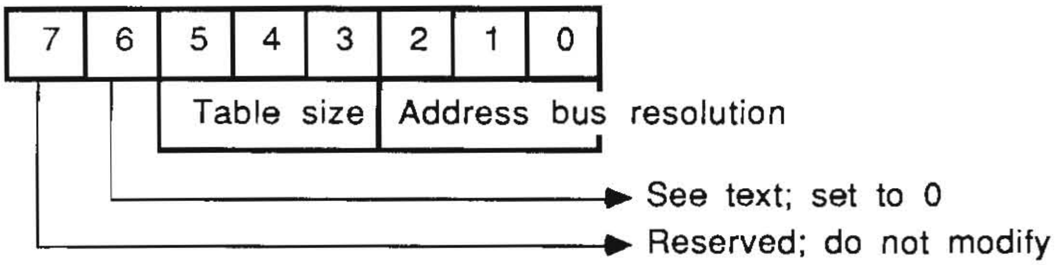


**Figure 6-6.** The Waveform register format

| 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 waveform table may extend up to 32K in size, but in discrete steps only, as listed in the table below. Wavetables must begin on a page boundary ($0C00, $0D00, and so forth). |

**Table 6-2**. Table 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 |

Unused locations within the wavetable should begin with a minimum of eight zeros. Otherwise, the oscillator will halt when it encounters these bytes and will not interpret them as data.

2–0 - Address bus resolution: The wavetable may be one of eight sizes, as just shown. The table 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. If the resolution bits are 000, all wavetable data bytes are read; if the resolution bits are 001, every other byte is used; if they are 010, every fourth byte is used, and so on. Table 6-3 shows how the different address bits to be used are selected.

**Table 6-3**. Table size determination

| Resolution bits | | | Address bits |
|---|---|---|---|
| 2 | 1 | 0 | used |
| 0 | 0 | 0 | 1–16 |
| 0 | 0 | 1 | 2–17 |
| 0 | 1 | 0 | 3–18 |
| 0 | 1 | 1 | 4–19 |
| 1 | 0 | 0 | 5–20 |
| 1 | 0 | 1 | 6–21 |
| 1 | 1 | 0 | 7–22 |
| 1 | 1 | 1 | 8–23 |

## Making sound

To generate sound, the DOC reads data bytes from a wavetable built in dedicated RAM. Each oscillator acts as an address generator, pointing to successive data bytes which make up a signal that is to be reproduced. Each data byte pointed to by an oscillator, is then converted to an analog value by the A-to-D converter within the DOC. The resulting series of values make up the output signal, which is then filtered and amplified and output to the 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 fill 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 Waveform register of each oscillator, and is arrived at using several factors. Figure 6-7 shows the process which results in the scan rate.

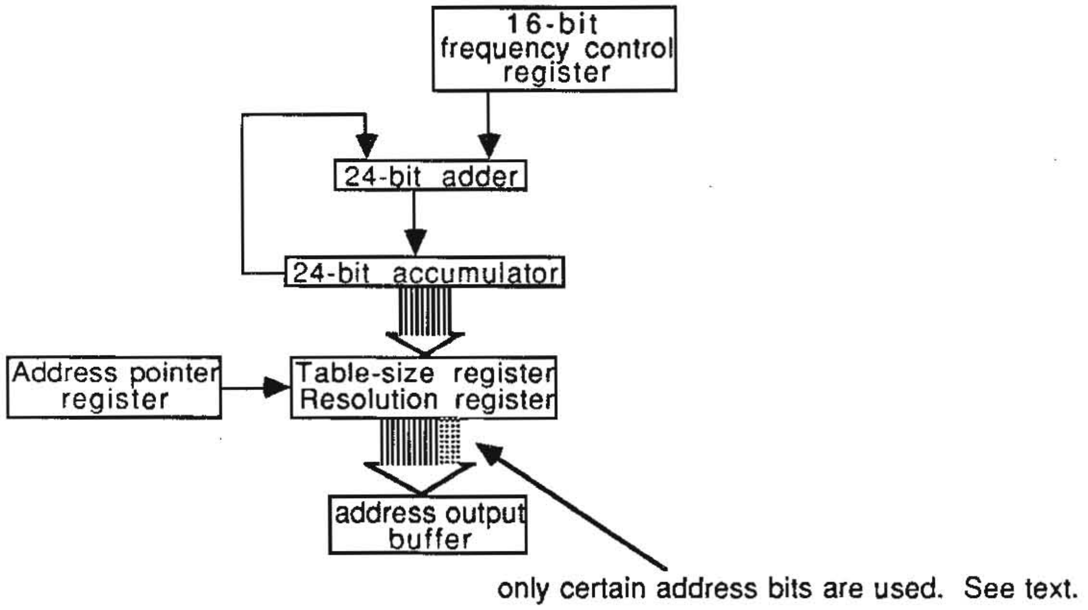


**Figure 6-7.** Generating the sound addresses

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.

Digitized soundwaves are built using consecutive data bytes (known collectively as a *wavetable*) in dedicated sound RAM. Each of the 32 oscillator 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.

## 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 below in Table 6-4.

**Table 6-4.** Sound input and output electrical specifications

| **Signal Name** | **Pin** | **Max.** | **Units** |
|---|---|---|---|
| A/D input | 1 | 2.5 | V p-p, full-scale conversion |
| Analog ground | 2 | - | - |
| Analog output | 3 | –5 to +5 | V p-p |
| Channel addr 0 | 4 | 1 | LS TTL load |
| Channel addr 1 | 5 | 1 | LS TTL load |
| Channel strobe* | 6 | 1 | LS TTL load |
| Channel addr 2 | 7 | 1 | LS TTL load |
| Input impedance | 1 | 3,000 | ohms |
| Output impedance | 3 | 10,000 | ohms |

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

Figure 6-8 shows an example of a demultiplexer circuit that can be used to produce stereo (2-channel) sound using the output from the DOC available at the 7-pin molex connector J-25. A more complex circuit would result in 15 unique sound channels.

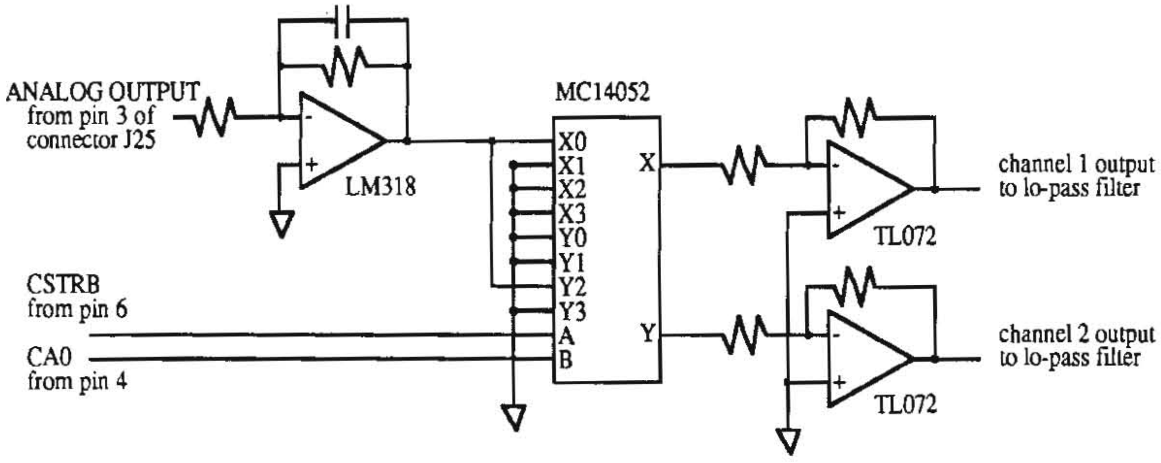


**Figure 6-8.** An example of a two-channel demultiplexer circuit

## Summary

The Apple IIGS provides sound synthesis capabilities to the Apple II family. The sound tool sets provide routines that manipulate the sound synthesizer. If you decide to bypass the tools, you must address the registers within the DOC and the sound GLU, as well as the sound RAM. To do this you must:

1. Set the Sound Control register bits for DOC or RAM access with the Address Pointer auto-increment option enabled.

2. Load the Address Pointers with the target address.
3. Read or write the data byte.

The addresses of the DOC registers for each oscillator are listed in Table 6-3.

**Table 6-3.** DOC register addresses

| Oscillator number | Frequency Low register | Frequency High register | Volume register | Data register | Address Pointer register | Control register | Waveform 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 may result in a system crash.

## Additional information

To learn more about synthesized music and sound, we recommend you read the following book:

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

**Chapter 7**

# 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 ADB consists of an ADB microcontroller chip and the Apple DeskTop Bus cabling. Figure 7-1 shows the relationship of the ADB components in the Apple IIGS computer.

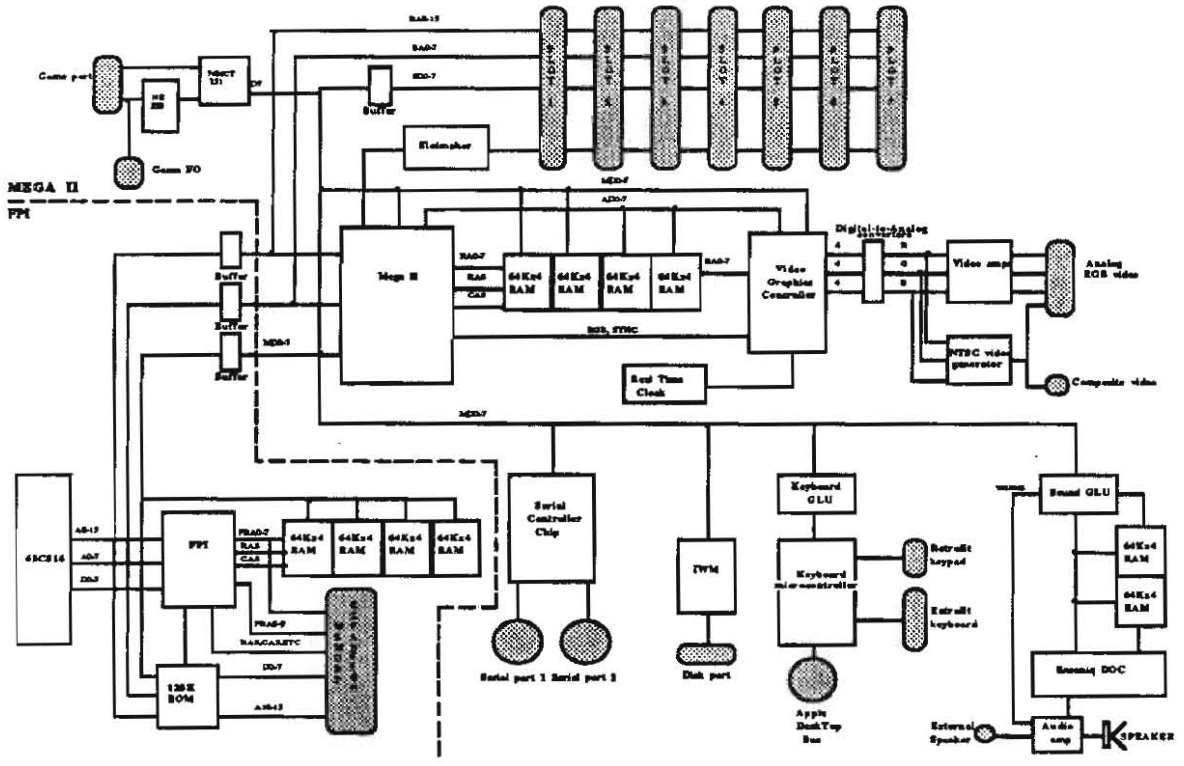


**Figure 7-1.** ADB components within the Apple IIGS

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 7-2 shows the relationship of ADB components in the host to devices on the bus.

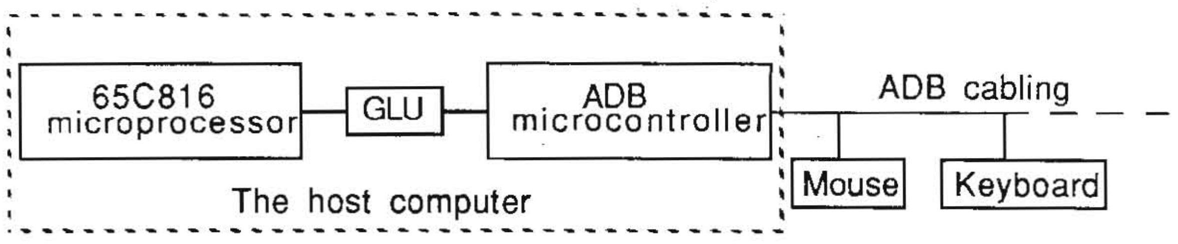


**Figure 7-2.** The ADB components

*Note*: To keep compatability with future Apple II products using ADB, use the Apple DeskTop Bus Tool Set in ROM. Directly accessing some of the ADB registers may cause the system to crash.

To find more information about the ADB Tool Set, refer to the *Apple IIGS Toolbox Reference* manuals.

This chapter describes the physical and network layers of the ADB as it is used in the Apple IIGS computer. 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*.

## 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 7-3 shows the pin assignments of the connectors. 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 cable capacitance should not exceed 100 picoFarads per meter.

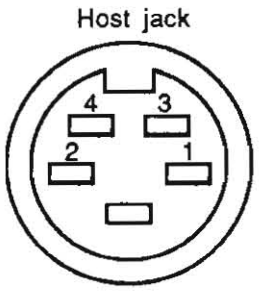


**Figure 7-3.** Mini-DIN connector pin configuration used in the ADB

| Pin | Description |
|---|---|
| 1 | Data |
| 2 | Reserved |
| 3 | +5 power supply at 500 mA for all devices |
| 4 | Signal and power ground |

## 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, issuing commands on the bus and transmitting data to and receiving data from the bus devices. 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* manual.

## The keyboard GLU

The keyboard general logic unit (GLU) works together with the ADB microcontroller to form an intelligent input-device interface. The keyboard 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.

### Keyboard GLU registers

The keyboard general logic unit contains seven data and control registers. These are used for storing keyboard data and commands, key modifiers, mouse X and Y coordinates, and status information. The registers are

- Keyboard data register ($C000)
- Mouse Coordinate register ($C024)
- Modifier key register ($C025)
- ADB Command/Data register ($C026)
- 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 keyboard data, mouse, and data registers also have an interrupt flag that generates system interrupts, if interrupts are enabled. These status and interrupt flags are located in the status register. The registers are described in the following sections.

### 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 7-4 shows the format of this register and is followed by a description of each bit.

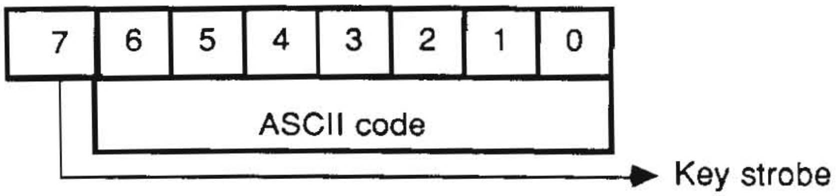


**Figure 7-4.** The keyboard data register at $C000

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

## Mouse Data register

The ADB mouse, when moved, generates movement data which is transmitted to the host. This data, along with the mouse button status, is available in the mouse data register. Figure 7-5 shows the format of this register and is followed by a description of each bit.

**Important:** Read this register only twice in succesion. The first read returns X-coordinate data, and the second read returns y-coordinate data. Reading this register an odd number of times will result in an unknown effect.

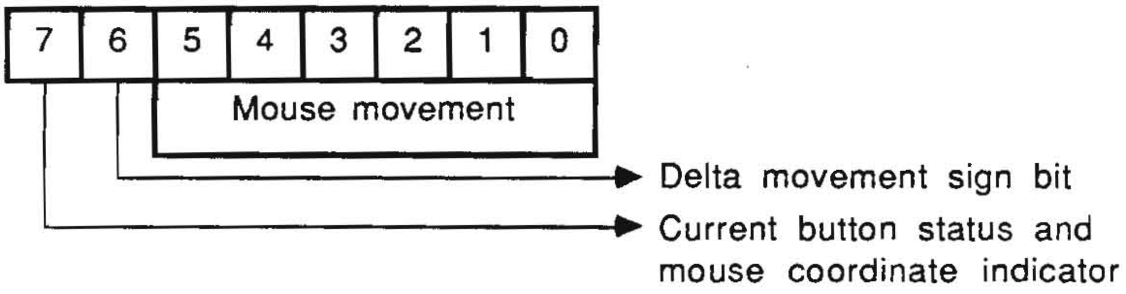


**Figure** 7-5. The Mouse Data register at $C024

| Bit | Value | Description |
|---|---|---|
| 7 | 1 | Current mouse status: Mouse button up. |
|  | 0 | Mouse button down. |
| 6 | 1 | Delta value sign bit: if this bit is 0, the delta value is +. |
|  | 0 | If this bit is 1, the delta value is –. |
| 5–0 | - | The relative mouse movement data is returned here. If you are reading this register for mouse movement data, bit 7 tells you which data (X or Y-coordinate) is represented here. If bit 7 is 1, X-coordinate data is contained in bits 5–0. If bit 7 is 0, Y-coordinate data is contained in bits 5–0. |

## Modifier Key register

The Modifier Key register contains bits that reflect the status of the modifier keys. These keys include the standard shift, control, repeat, command, Apple, and repeat keys, as well as keys on the numeric keypad. Figure 7-6 shows the format of this register and is followed by a description of each bit.

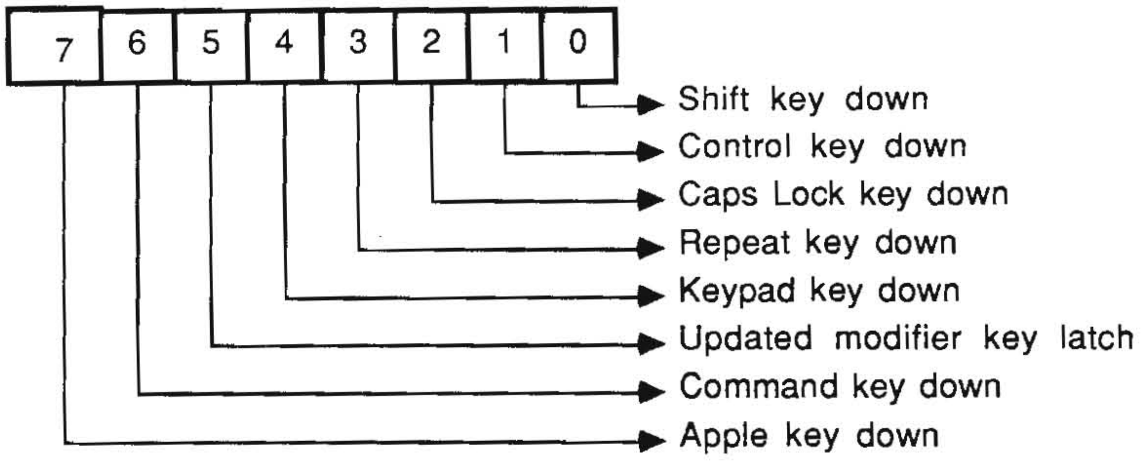


**Figure** 7-6. The Modifier Key register at $C025

| Bit | Value | Description |
|---|---|---|
| 7 | 1 | When this bit is 1, the Open Apple key has been pressed. |
| | 0 | When this bit is 0, the Open Apple key has not been pressed. |
| 6 | 1 | When this bit is 1, the Closed Apple key has been pressed. |
| | 0 | When this bit is 0, the Closed Apple 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 keypad key has been pressed. |
| | 0 | When this bit is 0, a keypad key has not been pressed. |
| 3 | 1 | When this bit is 1, the Repeat function is active. |
| | 0 | When this bit is 0, the Repeat function is inactive. |
| 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. |

## 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 7-7 shows the format of the ADB command/data register when it is read.

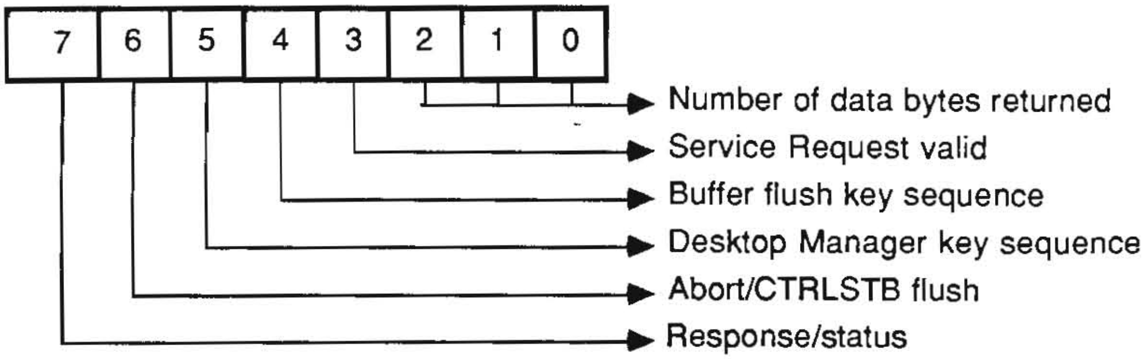

7
6
5
4
3
2
1
0
Number of data bytes returned
Service Request valid
Buffer flush key sequence
Desktop Manager key sequence
Abort/CTRLSTB flush
Response/status

**Figure** 7-7 The ADB command/data register at $C026.

| 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 available. |
| 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, this indicates that the ADB microcontroller should clear the key strobe (bit 7 in the data register at $C000). |
| | 0 | - |
| 5 | 1 | When this bit is 1, the Apple, Control, 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 Apple, Control, and Delete keys have been pressed simultaneously. This condition will result in the ADB microcontroller 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. |

## ADB Status register

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

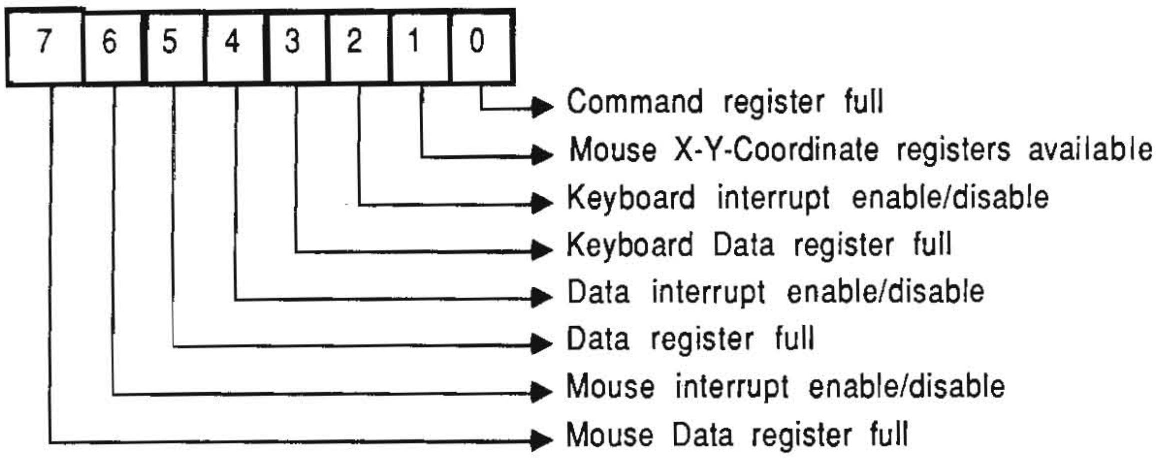

7
6
5
4
3
2
1
0
Command register full
Mouse X-Y-Coordinate registers available
Keyboard interrupt enable/disable
Keyboard Data register full
Data interrupt enable/disable
Data register full
Mouse interrupt enable/disable
Mouse Data register full

**Figure** 7-8. ADB Status register at $C027

| **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). |
| | 0 | When this bit is 0, the mouse interrupt is disabled. |
| 5 | 1 | When this bit is 1, the data register is full (read-only bit). |
| | 0 | When this bit is 0, the data register is empty. |
| 4 | 1 | When this bit is 1, the data interrupt is enabled (read/write bit). |
| | 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). |
| | 0 | When this bit is 0, the keyboard data register is empty. |
| 2 | 1 | When this bit is 1, the keyboard data interrupt is enabled (read/write bit). |
| | 0 | When this bit is 0, the keyboard data interrupt is disabled. |
| 1 | 0 | Mouse X-Coordinate register available (read-only bit). |
| | 1 | Mouse Y-Coordinate register available. |
| 0 | 1 | Command register full (read only bit) |
| | 0 | Command register empty. |

The command full flag is set to 1 when the system writes to the Command register and is cleared to 0 when the ADB microcontroller reads the Command register.

The keyboard data full flag is set when the keyboard writes data into this register; it is cleared when the system reads the System Status register and the keyboard data register.

## Bus communication

The host carries communication on the bus by sending either commands or data to a device. A device can respond to commands by sending data 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.

Figure 7-9 shows how duty-cycle modulation represents bits on the bus. A low period of less than 50 percent of the bit-cell time is interpreted as a 1. A low period of greater than 50 percent of the bit-cell time is interpreted as a 0.

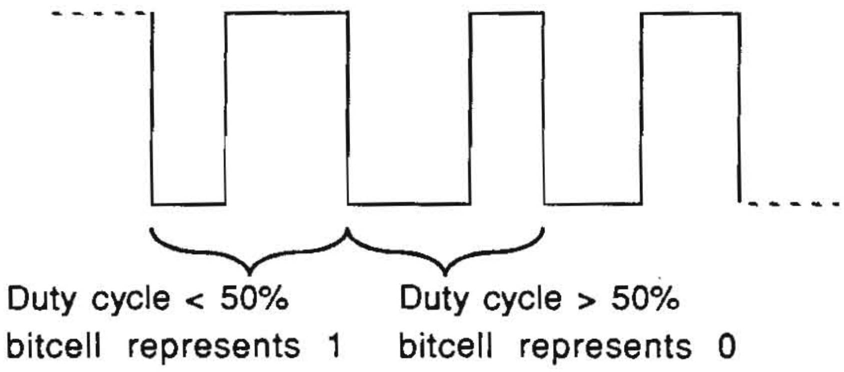


**Figure 7-9.** Bit representation via duty-cycle modulation

### Signals

Certain transactions are neither commands nor data transactions. These are special transactions that the host uses to broadcast status globally to all devices on the bus. There are four special transactions in this group: *attention*, *sync*, *reset*, and *service request.*

#### Attention and sync

The start of every command is signaled by a long low attention period that the host sends on the bus. This 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 7-10 shows the format of the attention and sync signals.

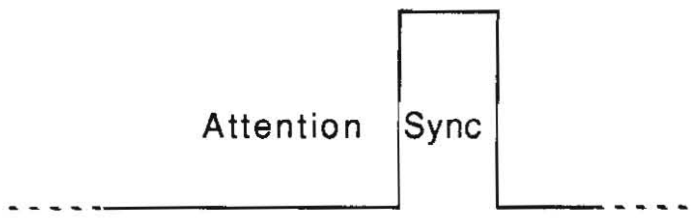


**Figure** 7-10. Attention and sync pulses

### Reset

Reset issues a break on the bus. Only the host may issue this signal, which signals all devices to reset. By holding the bus low for a minimum of 2.8 milliseconds, a reset is initiated.

### Service request

A service request is used to signal the host that a device requires service, such as 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 command transaction. This lengthens the stop by a minimum of 140 milliseconds beyond its normal bit-cell boundary. This lengthened stop bit indicates to the host that a service request is desired. Figure 7-11 shows the format of the service request signal.

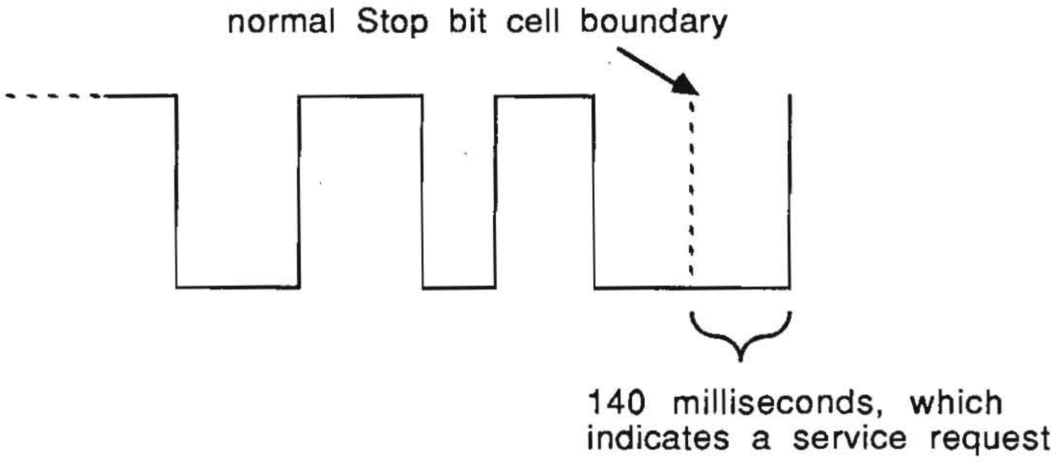

normal Stop bit cell boundary
140 milliseconds, which indicates a service request

**Figure** 7-11. Service Request

A device will signal a service request repeatedly until it is served. When a device has requested service (at this point the host does not know *which* device sent the request), the host will poll each of the devices by sending a Talk register 2 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.

## Reset

The host can reset all devices on the bus by holding the bus low for a minimum of 250 milliseconds. Upon detecting a reset on the bus, devices will reset and place themselves in a mode to receive commands.

## Transactions

A command (Talk, Listen, Flush) initiates a transaction. The sequence of the command is as follows:

1. an attention signal
2. a sync signal
3. eight command bits
4. one stop bit

To synchronize the end of the transaction, the command transmits the 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 initiate a transaction).

## ADB peripheral devices

Each device on the bus has an address. There is only one active talker on the bus at a time; this may be the host or an addressed device. A device addressed to Talk (with data to send) releases control of the bus after it sends the data. If a device has been addressed but has no data to send, it releases control of the bus immediately and allows the host to time out (waiting for data, none arrives). The host may also send data to the addressed device in a separate packet, after it sends a Listen command to the device.

Each peripheral device has a 4-bit command address that identifies its device type. A total of 16 addresses are available, which means that a maximum of 16 unique devices may be on the bus. A device always responds to its address when there is a power-on or a reset signal.

## 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 moveable addresses; the host can assign a new address to the device. However, a device will always default to its assigned address upon power-on or after it detects an ADB reset. Currently, 8 addresses are predefined and reserved. The other 8 addresses are available for moveable devices. This means that ADB can support up to 9 mouse devices at the same time, each one with a unique address.

## Commands

Only the host can send commands. There are two types of commands: the Talk command is used for data transaction from a device to the host; the Listen command is used for a data transaction from the host to a device.

A command is an 8-bit word that has a specific syntax (see Table 7-1):

- a 4-bit field that specifies the address of the desired device. The addresses range from 0–15 (A3–A0).
- a 4-bit command and register address code.

**Table 7-1.** Command byte syntax

| Bit | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** | **Command** |
| x | x | x | x | 0 | 0 | 0 | 0 | Send reset |
| A3 | through | | A0 | 0 | 0 | 0 | 1 | Flush |
| x | x | x | x | 0 | 0 | 1 | 0 | Send reset |
| x | x | x | x | 0 | 0 | 1 | 1 | Send reset |
| x | x | x | x | 0 | 1 | x | x | Send reset |
| A3 | through | | A0 | 1 | 0 | r | r | Flush |
| A3 | through | | A0 | 1 | 1 | r | r | Flush |

x = ignored
r = register number

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

## Talk

When the host addresses and requests 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.

## Listen

When the host addresses and requests a device to Listen, it is enabled to receive the data bits that the host places on the bus. The host performs its data transaction. After the stop bit that follows the data is received, the transaction is complete and the device releases the bus. If a listening device detects another command on the bus before it receives any data, the original transaction is immediately considered complete and the device releases the bus.

## Flush

The flush command is a device-independent command which requests that the device purge all pending data from its buffer.

## Send Reset

## 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: | Soft addressed devices; device specific |
| Register 3, | Talk: | Status information, including the device address handler |
| | Listen: | Status information, including the device address handler |

## 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 (either high or 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 immediatedly *untalk* itself and preserve the data that were being sent for retransmission. The 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 line turnaround time.

## Error conditions

If the bus hangs low, 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 prepare for another attention signal.

## Network layer

The network layer accommodates normal devices and extended address devices.

### Normal devices

A normal device optionally has a device, called the *activator*, on it to indicate activity. The activator can be a special key on a keyboard or a mouse button.

To aid in collision detection, the address portion of the address field of register 3 is replaced with a random number in response to a Talk R3 command. Normal devices will change their register 3 to the data received when they receive a Listen R3 command, no collision is detected, and activator inactive is true.

At the systems level, a host can change the address of normal devices by forcing the collision of devices sharing the same address. By issuing a Talk R3 command and following it with a Listen R3 command with a new address in bits 8 to 11 of the data, all devices that did not detect a collision are moved to the new address. Typically, only one device wil not detect a collision. This process can be repeated at new addresses until the response to the Talk R3 command is a time-out. This can be used to identify and relocate multiple devices of the same type after initialization of the system.

At the applications level, addresses can be changed by displaying a message requesting a user to use the activator. The host then issues a Listen register 3 command to a new address and all devices except the one with the activator being used are moved. This method can be used to identify and locate individual devices in multiuser applications. Certain handlers have been reserved to facilitate both address changing methods.

### Extended address devices

Extended address devices have the same command address and a unique 16-bit extended address that is stored in the device. Their command address cannot be changed. On power-up or RESET, they accept only the Listen register 2 command in which the data match their extended address. When enabled, they respond to all commands addressed to them. These devices become disabled after receiving a Listen register 2 command in which the data do not match their extended address.

## Register 3

The function of a device and the use of its data by the host are controlled by a handler that is stored by the device in register 3. The host changes the handler with a Listen register 3 command. If the receiving device is able to function with the new handler, it is stored and sent in response to a Talk register 3 command.

Handler $FF is reserved for the self test mode for all devices. Handler $00 in response to a Talk is reserved to indicate a failed self test. Handler $00 sent with a Listen is reserved to indicate that the device is only to change the address portion of register 3. Figure 7-12 shows the format of register 3.

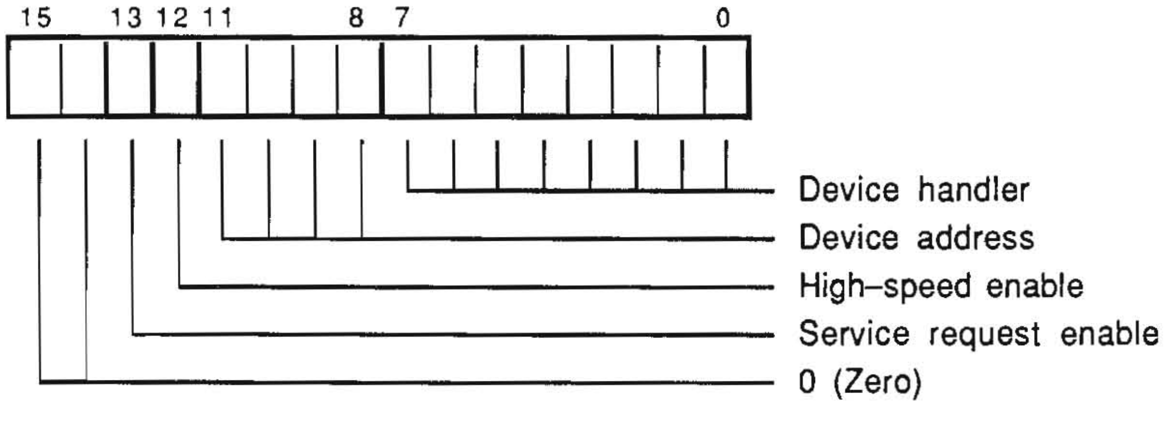


**Figure 7-12.** Register 3

## Register 0

Register 0 is used to hold device data, and is read using the Read register 0 command. Figures 7-13 and 7-14 show the format of register 0 as used in a keyboard and mouse device.

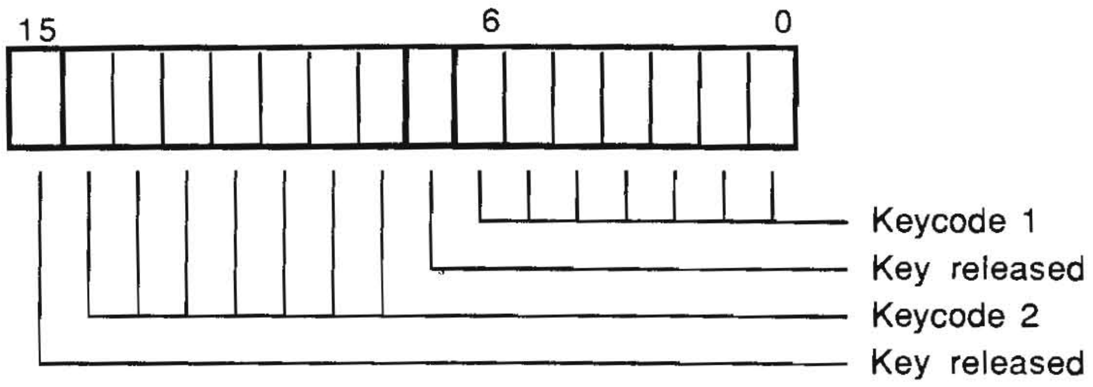


**Figure 7-13.** Keyboard Register 0

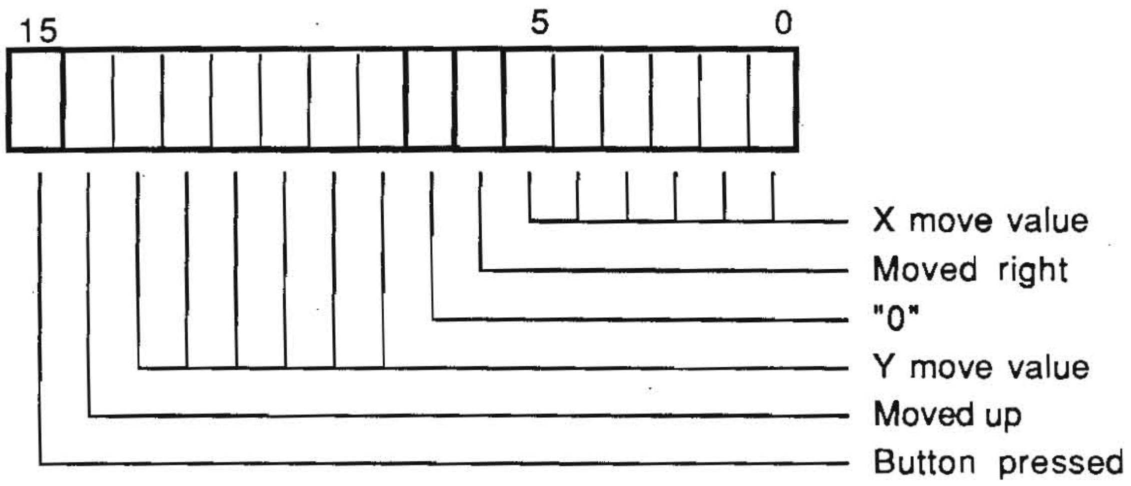


**Figure** 7-14. Mouse Register 0

## Service request

The Listen R3 command is also used to enable and disable the service request. Service request is enabled on the bus by setting the register 3 enable bit to 1; it is disabled by setting the bit to 0. This is useful in systems 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.

# Chapter 8

# The Disk Port

The Apple IIGS computer can use either 5.25-inch 140K disk drives or the 3.5-inch 800K disk drives. The disk port connector at the rear panel is compatible with both types of Apple disk drives. This chapter describes the disk port connector of the Apple IIGS. Figure 8-1 shows the Apple IIGS block diagram and position of the disk port within the system.

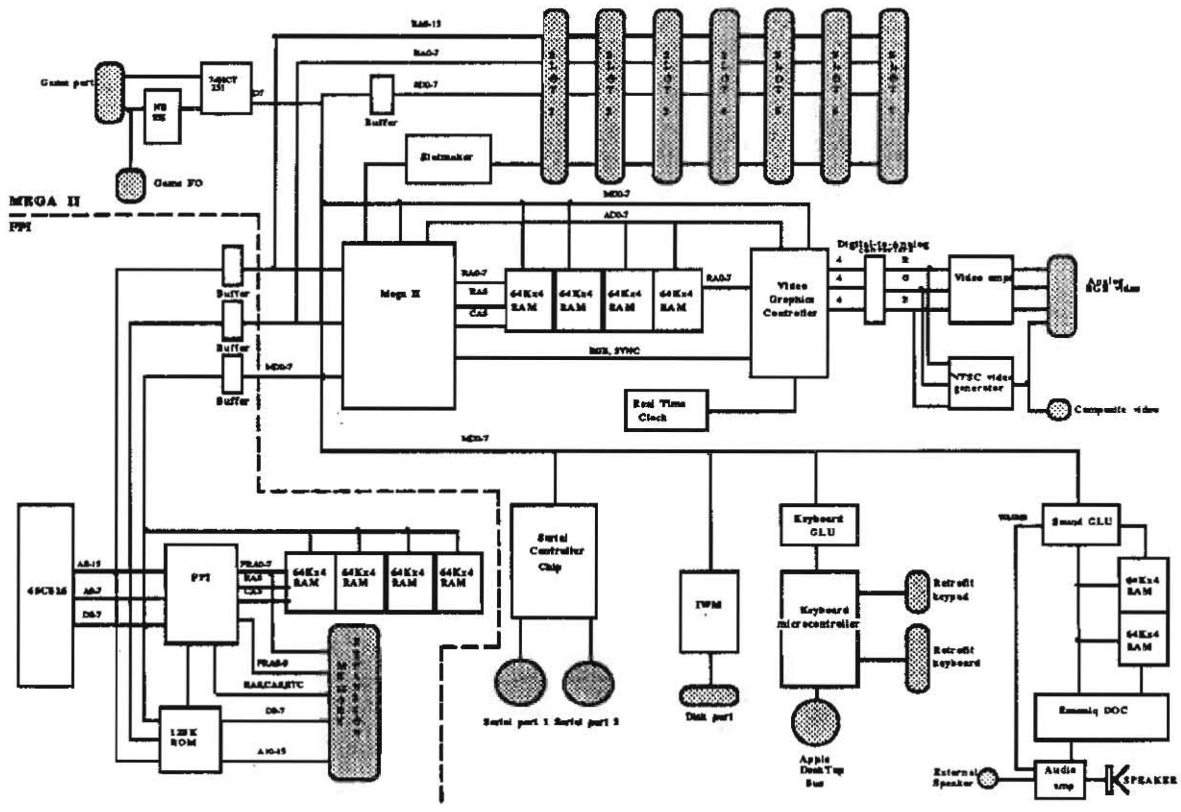


**Figure 8-1.** Relationship of the disk port and other components

## 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 and Apple DuoDisk® 5.25-inch disk drives.

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

## 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 8-2 shows the connector, and the signals and their descriptions are listed below.

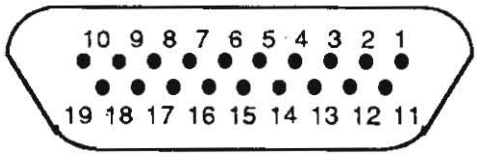


**Figure 8-2**. The disk port connector

| Pin | 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 | PH0 | Motor phase 0 output |
| 12 | PH1 | Motor phase 1 output |
| 13 | PH2 | Motor phase 2 output |
| 14 | PH3 | 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 these connections for any other purpose. Any other use of these supplies may damage the voltage regulator within the computer.

## 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 data to and from each disk drive. To perform disk operations, the microprocessor simply reads or writes control and data bytes to or from the six IWM registers.

The disk port is mapped as an internal device at addresses $C0E0 through $C0EF. These are the same addresses as in the Apple IIc and IIe. Table 8-2 shows these locations and their functions.

**Table 8-2.** 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 |

Soft switches Q6 and Q7 are select bits for accessing registers within the IWM. By setting or clearing the Q6, Q7, and spindle motor bits, you may read or write to one of the registers.

## 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 5–0). Figure 8-3 shows the format for this register. Descriptions of each bit are also listed below.

*Warning*: Be careful when changing bits within this register. Use only a read-modify-write instruction sequence when manipulating bits. See the warning in Chapter 1, "Introduction to the Apple IIGS".

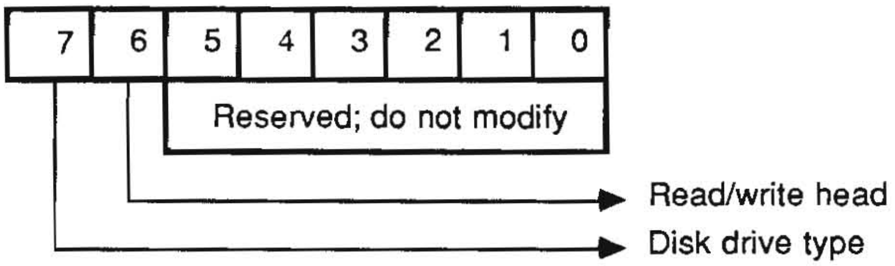


**Figure 8-3.** Disk Interface register at $C031

| 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. |

## Chapter 9

# The Memory Expansion Slot

The extended memory card slot allows you to add a memory card holding up to 8Mb of RAM and 896K of ROM memory. It supports additional memory only and is not to be used for any other purpose. RAM cards of 1Mb or 4Mb can be constructed by using 256Kilobit x 1-bit or 1Megabit x 1-bit RAM ICs. Figure 9-1 shows a block diagram of the Apple IIGS, and the relationship of the memory expansion slot with the rest of the computer system.

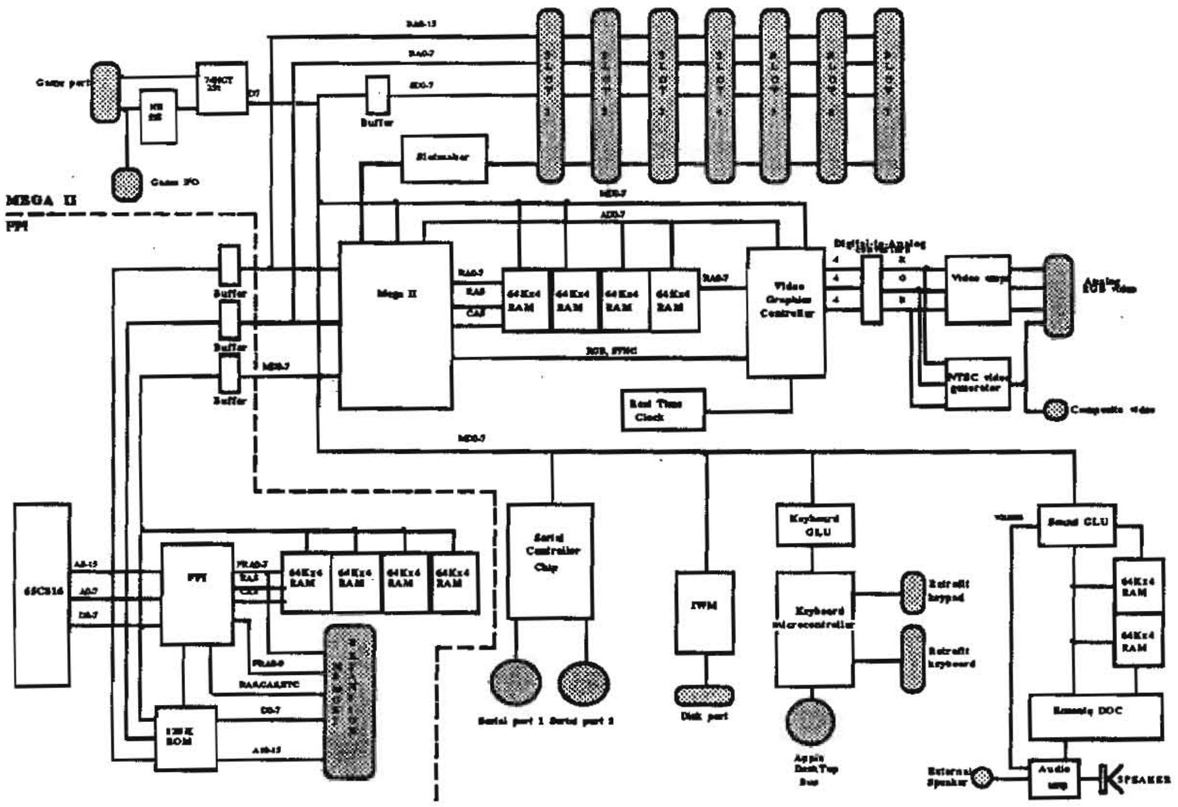


**Figure 9-1.** Diagram showing location of expansion memory in the system

## Extended RAM

Up to 4Mb (64 banks of 64K each) of RAM can be designed in the extended memory card. This would be organized as 4 rows (8 chips per row) of RAMs with each row holding either 256K or 1Mb. This requires the use of 256 kilobit x 4 bit (resulting in 1Mb total) or 1 megabit x 1 bit RAMs (yielding 4Mb total).

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 9-2 depicts a 1Mb extended RAM card using 4 rows of 256K per row totaling 1Mb. The RAM banks above bank $11 are *ghosts* (repeat 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 1Mb expansion cards.

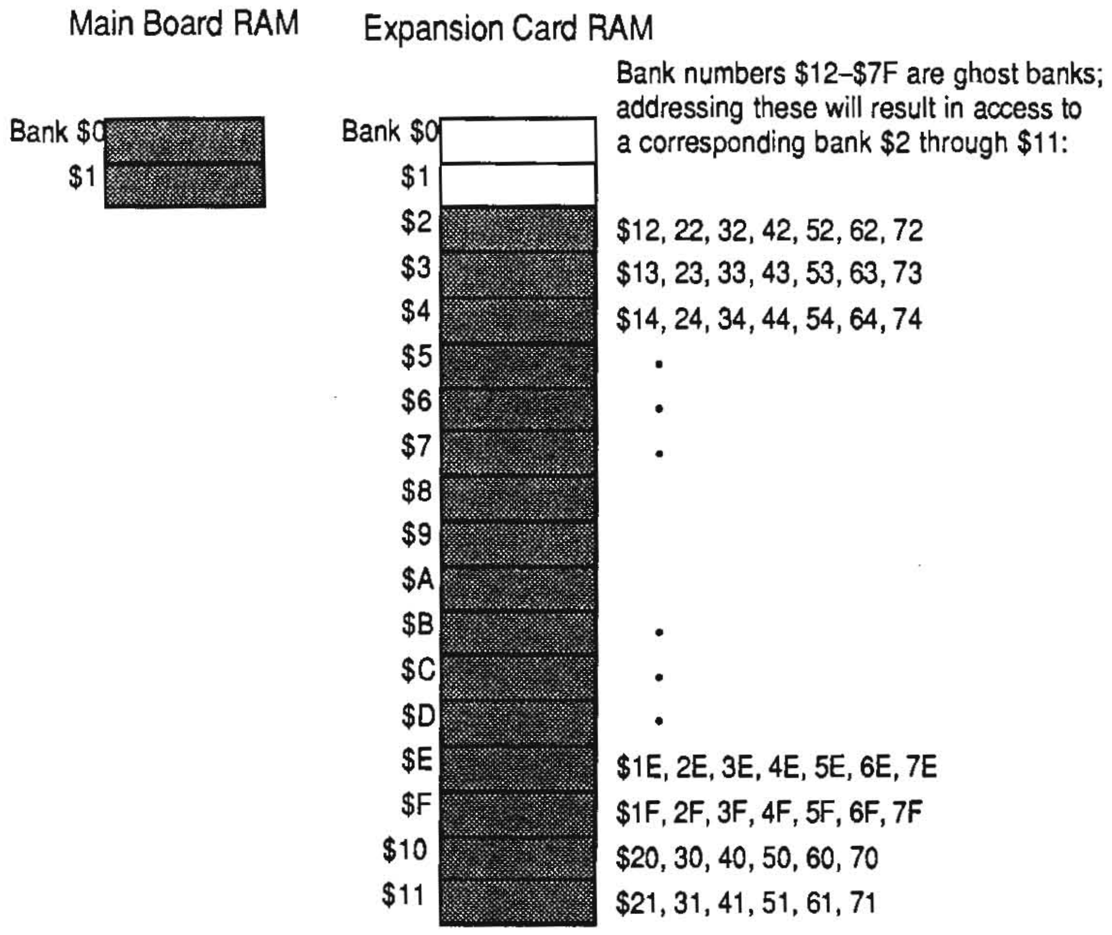


**Figure 9-2.** Extended RAM mapping

## MSIZE

A new input to the memory expansion slot, MSIZE, flags the type of memory chips being used on the memory expansion card.

If the MSIZE pin is tied to ground (when using 256 kilobit RAMs), the FPI multiplexes 18 address bits onto RA0–8 and generates the CROW0–1 row selects for rows of 256K. If the MSIZE pin is not connected (for 1 megabit RAMs), the FPI multiplexes 20 address bits onto RA0–9 and generates the CROW0–1 row selects for rows of 1Mb.

### Ghosting

The RAM expansion card is enabled for accesses in banks $2–$80, but only provides 1 megabyte of actual RAM (banks $2–$11). Four eight-IC rows of RAMs on the card are individually selected by CROW0 and CROW1. For a 1Mb card with 256 K rows (MSIZE = 0), the selected RAM row number is given by the bank number mod 4. For banks $0–$1 (main board RAM), the extended memory card is not accessed. This method of card and row selection causes multiple images or *ghosting* of the RAM areas on the card; whenever locations above $FFFFFF are addressed, locations in a corresponding low bank ($2–$11) are accessed.

## Extended ROM

Additional ROM space up to 896K is available in banks $F0 to $FD. To accomplish this, an additional bank-address latch-decoder is required on the memory card. The FPI provides a signal (CROMSEL.L) that selects one bank; however, the card must provide the additional decoding to select individual ROMs within the selected bank.

The extended memory card connector provides a group of signals to support dynamic RAM and additional general purpose signals to support ROM decoding and selection. Table 9-1 lists these signals.

**Table 9-1**. Memory card interface signals

| Signal | Description |
|---|---|
| FRA0-FRA9 | 10 bits of multiplexed RAM address for RAM cycles— the least significant 10 bits of ROM address. |
| CROW0,1 | 2 bits select 1 of 4 RAM rows. |
| CRAS.L | RAM/RAS strobe. |
| CCAS.L | RAM/RAS strobe. |
| FR/W | Write enable to RAMs. R/W from microprocessor or DMA. |
| D0-D7 | 8 bits of bidirectional data—microprocessor data bus |
| CDIR.L | Card data buffer direction control. Signal goes high when reading card data |
| CROMSEL.L | Card ROM select. Low for accesses to banks $F0–$FD. |
| PH2CLK | Microprocessor clock. Rising edge indicates valid bank address on D0–D7. |
| MSIZE | Output from card. Indicates RAM row size. |
| A10-A15 | The 6 high-order address bits. Used with ROMs. |
| 14M | 14 MHz clock signal. |
| VCC | +5v ±5%. 600mA maximum. |

To control and select individual rows of RAM located on the ROM card, the FPI provides the /CRAS, /CCAS, CROW0, and CROW1 signals. The /CRAS and /CCAS signals are for basic memory timing common to most dynamic RAMs. The CROW0 and CROW1 are row select signals that, when taken as a pair, indicate the row number to be accessed. Typically, CROW0 and CROW1 are 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 row. Figure 9-3 shows a typical circuit for RAM row selection.

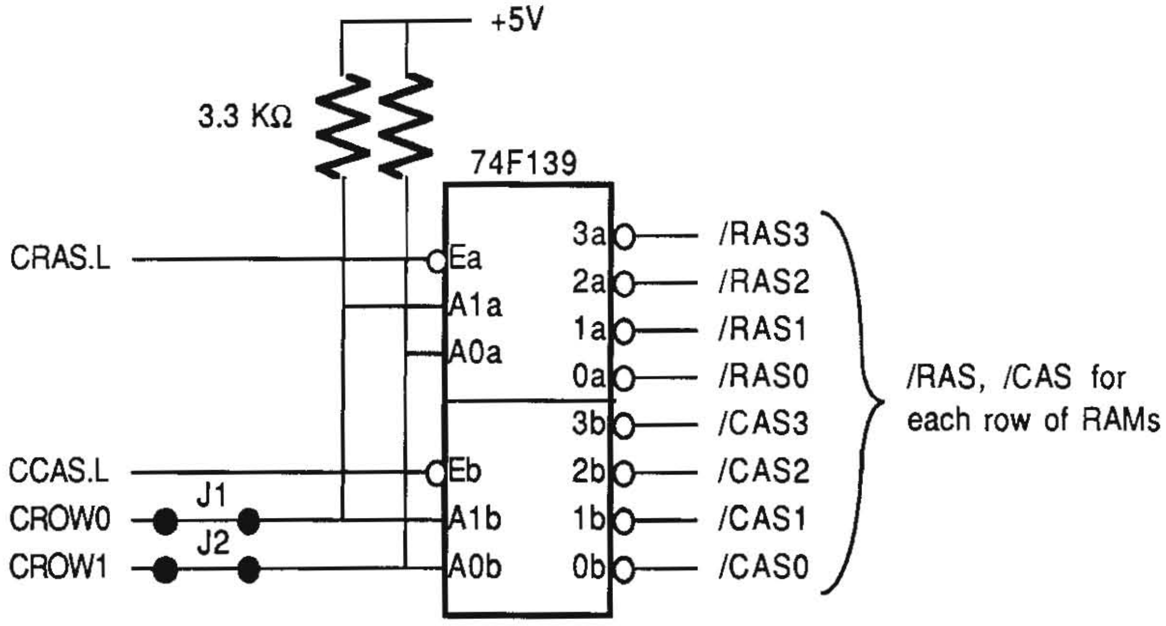


**Figure 9-3.** Example circuit for decoding the extended memory card RAS/CAS signals

## Address multiplexing

The FPI multiplexes the RAM addresses onto either 8, 9, or 10 RAM address lines to provide support for RAM with 64 kilobit, 256 kilobit, or 1 megabit RAM ICs. The main logic board RAMs (banks $0 and $1) are 64 kilobit chips requiring 8 address lines. The RAM expansion card can support 256 x 1-, 256 x 4-, 1M x 1-, or 1M x 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 MSIZE, above.)

# Chapter 10

# 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 used. The following sections describe the design of this unit.

## Function

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 a simple, maintenance-free operation.

> **Warning:** The power supply contains dangerously high voltages, and should be opened by an authorized Apple service technician only.

This power supply also contains special load-sensing circuitry; whenever it detects a short or a no-load condition, the 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 supply provides +12 volts, –12 volts, +5 volts, –5 volts, and 2 ground return lines.

The power input requirements are 107 to 132V AC. The power output specifications are as follows:

- +12 volts at 1.5A
- –12 volts at 0.25A
- +5 volts at 2.5A
- –5 volts at 0.25A

## Power connector

The connector is a 6-pin, molex-type, keyed in-line socket. Figure 10-1 shows its pin-out, followed by a description of each pin.

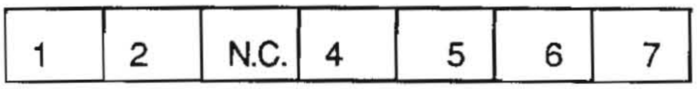


**Figure 10-1.** Power supply connector

| Pin | Description |
|---|---|
| 1 | Ground |
| 2 | Ground |
| 3 | N.C. |
| 4 | +5 volt supply |
| 5 | +12 volt supply |
| 6 | –12 volt supply |
| 7 | –5 volt supply |

# Chapter 11

# 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 11-1 shows the Apple IIGS block diagram and the relationship of the microprocessor to the rest of the computer.

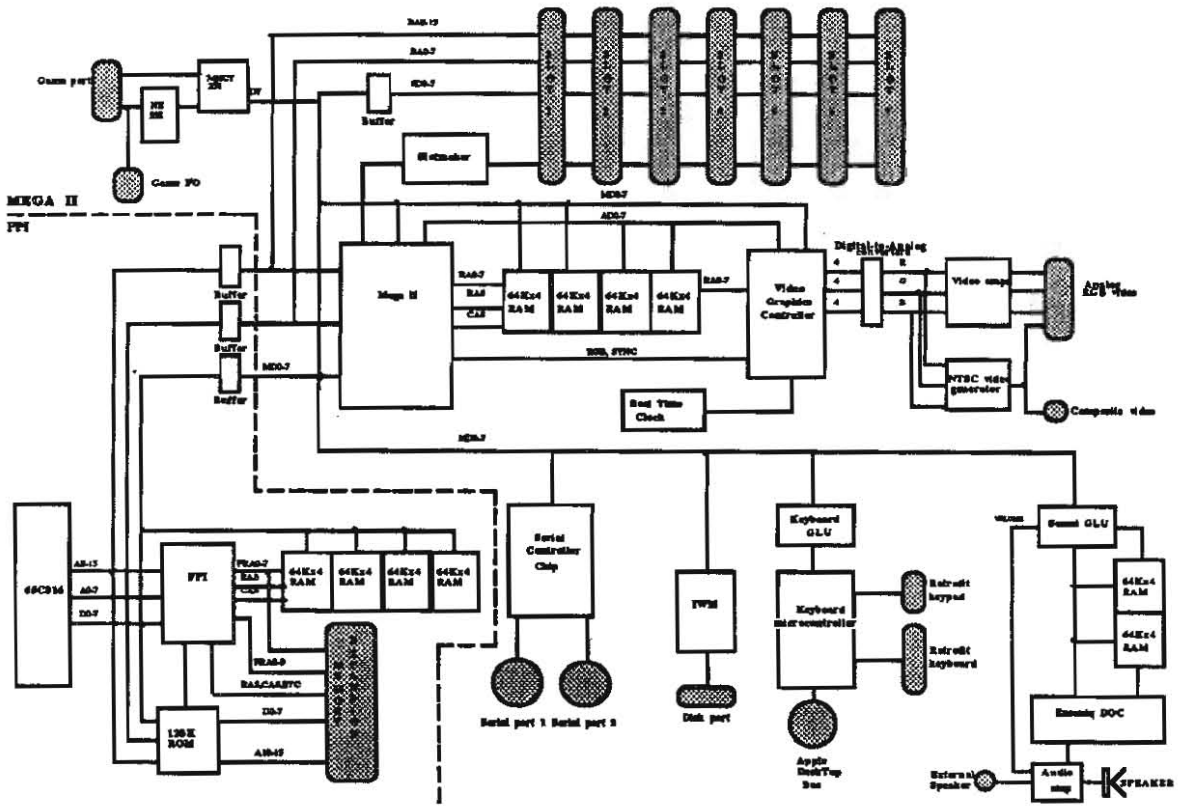


**Figure 11-1.** The 65C816 in the Apple IIGS system

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:

- 8Mb address range increases potential progam and data size
- 16-bit internal data registers increase data-handling capability
- 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.

## 65C816 features

The new 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

- 16-bit accumulator
- 16-bit X and Y Index registers
- relocatable zero page
- relocatable stack
- 24-bit internal address bus
- 8-bit data address bank register
- 8-bit program address bank register
- 11 new addressing modes
- 36 new instructions, for a total of 91 (all 256 operation codes)
- fast block-move instructions
- ability to emulate the 6502 8-bit microprocessors

For detailed descriptions of these features, refer to the manufacturer's data sheet at the end of this chapter. To learn how to implement these features, refer to the *Apple IIGS Assembler Reference* manual.

The 65C816 microprocessor shares some features with the 6502 and 65C02 microprocessors used in previous Apple II models. Table 11-1 lists some of these features.

**Table 11-1**. Some 6500 family ties

| Characteristic | Year available | 6502<br>1975* | 65C02<br>1983* | 65816<br>1985* |
|---|---|---|---|---|
| Construction | | NMOS | NMOS | 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

**NMOS** is an abbreviation for *N-doped Metal Oxide Semiconductor*, which is one of several methods of semiconductor integrated-circuit fabrication.

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.

## The 16-bit 65C816

In the Apple IIGS, the 65C816 normally operates in either of two modes: 6502 emulation mode and 65C816 native mode. Figure 11-2 shows the sizes of the registers in emulation mode and in native mode. In emulation mode, the accumulator and index registers are 8 bits wide, and existing Apple II programs run the same as they do on any other Apple II model. In native mode, the accumulator and index registers are 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 them ineffective.

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

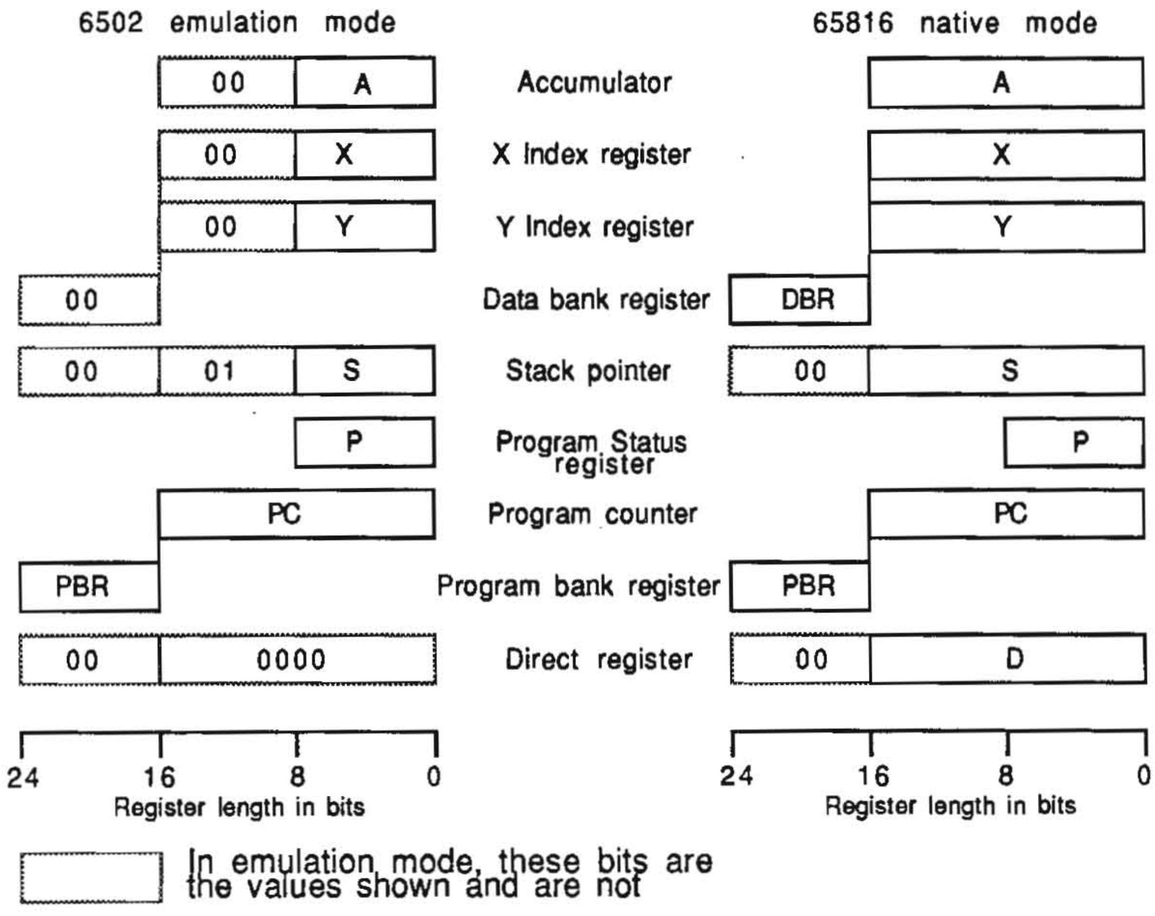


**Figure 11-2.** 65C816 registers

## 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 Assembler Reference* manual.

#### 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 result of a calculation within this register affects the status bits in the Program Status register. In emulation mode, the upper 8 bits are filled with 0's that cannot be altered.

#### 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 x bit is set, the upper 8 bits 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 x bit is set, the upper 8 bits are filled with 0's that cannot be altered.

#### Data bank register

The data bank register is an 8-bit address register that contains the most significant byte of the effective 24-bit address in all addressing modes. In emulation mode, it contains 0's that cannot be altered.

#### Stack pointer

In the previous Apple 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) and grows down toward lower addresses.

### 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 and m bits that control the emulation and native modes. In emulation mode, this register remains unchanged in size.

### 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. In emulation mode, this register remains unchanged in size. (See the program bank register address description, below.)

### 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, and remains unchanged in size. (See the program counter description, above.)

### 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).

## 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, addresses are 24 bits wide in native mode but are limited to 16 bits in emulation mode, and data registers that are 16 bits wide in native mode are reduced to 8 bits. Note in Figure 11-2 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.

### The e bit

The e bit in the Status register controls whether the 65C816 functions like a 6502 (emulation mode) or like a 65C816 (native mode). When this bit is a 1, the 65C816 executes only those instructions within the 6502 microprocessor.

In emulation mode, the microprocessor addresses 64K of memory, ignoring the program bank register. In native mode, the microprocessor addresses up to 8Mb by using the program bank register.

#### The m bit

The m bit in the Status register controls whether the accumulator and memory locations are 8 or 16 bits wide. When the m bit is set, references to the accumulator and memory locations are 8 bits wide. When the m bit is cleared, references to the accumulator and memory locations are 16 bits wide.

In emulation mode (e bit = 1) the m bit is forced to 1, and all references to the accumulator and memory are 8 bits wide.

#### The x bit

The x bit in the Status register controls whether the X and Y registers are 16 bits or 8 bits wide. When the x bit is set, references to the X and Y registers are 8 bits wide. When the x bit is cleared, references to the X and Y registers are 16 bits wide.

## 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.

## Summary

The new 65C816 16-bit microprocessor provides these improvements over the 6502:

- 16-bit accumulator
- 16-bit X and Y Index registers
- relocatable zero page
- relocatable stack
- 24-bit internal address bus
- 8-bit data address bank register
- 8-bit program address bank register
- 11 new addressing modes
- 36 new instructions, for a total of 91 (all 256 opcodes)
- fast block-move instructions
- ability to emulate the 6502 8-bit microprocessors

## 65C816 data sheets

In the following pages are the data sheets from two manufacturers of the 65C816 microprocessor.

# 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

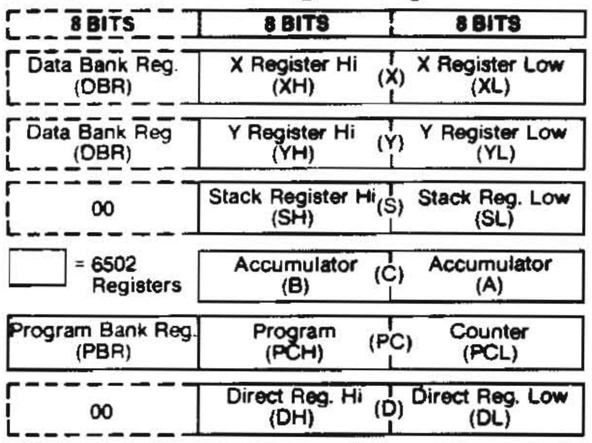
WDC's W65C802 and W65C816 are CMOS 16-bit microprocessors featuring total software compatibility with their 8-bit NMOS and CMOS 6500-series predecessors. The W65C802 is pin-to-pin compatible with 6-bit devices currently available, while the W65C816 extends addressing to a full 16 megabytes. These devices 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.

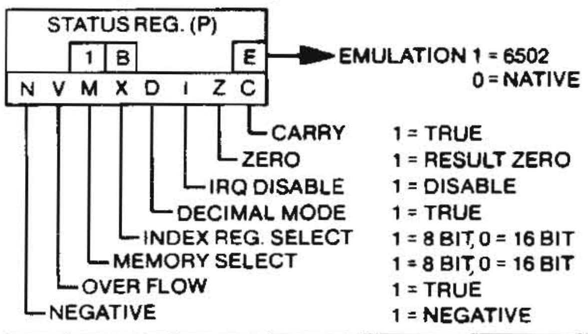
Four new signals provide the system designer with many options. The ABORT input can interrupt the currently executing instruction without modifying internal register, thus allowing virtual memory system design. Valid Data Address (VDA) and Valid Program Address (VPA) outputs facilitate dual cache memory 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.

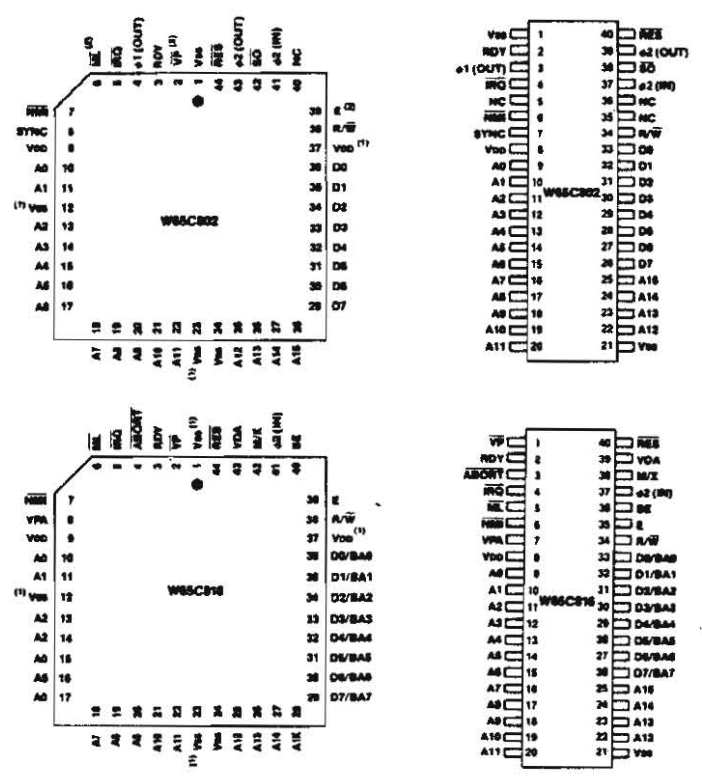
## W65C816 Processor Programming Model



## 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 | $V_{IH}$ | | | |
| $\overline{RES}$, RDY, $\overline{IRQ}$, Data, $\overline{SO}$, BE, | | 2.0 | $V_{DD}$ + 0.3 | V |
| φ2 (IN), $\overline{NMI}$, $\overline{ABORT}$ | | 0.7 $V_{DD}$ | $V_{DD}$ + 0.3 | V |
| Input Low Voltage | $V_{IL}$ | | | |
| $\overline{RES}$, RDY, $\overline{IRQ}$, Data, $\overline{SO}$, BE, | | -0.3 | 0.8 | V |
| φ2 (IN), $\overline{NMI}$, $\overline{ABORT}$ | | -0.3 | 0.2 | V |
| Input Leakage Current ($V_{IN}$ = 0 to $V_{DD}$) | $I_{IN}$ | | | |
| $\overline{RES}$, $\overline{NMI}$, $\overline{IRQ}$, $\overline{SO}$, BE, $\overline{ABORT}$ (Internal Pullup) | | -100 | 1 | μA |
| RDY (Internal Pullup, Open Drain) | | -100 | 10 | μA |
| φ2 (IN) | | -1 | 1 | μA |
| Address, Data, R/$\overline{W}$ (Off State, BE = 0) | | -10 | 10 | μA |
| Output High Voltage ($I_{OH}$ = -100μA) SYNC, Data, Address, R/$\overline{W}$, $\overline{ML}$, $\overline{VP}$, M/X, E, VDA, VPA, φ1 (OUT), φ2 (OUT) | $V_{OH}$ | 0.7 $V_{DD}$ | — | V |
| Output Low Voltage ($I_{OL}$ = 1.6mA) SYNC, Data, Address, R/$\overline{W}$, $\overline{ML}$, $\overline{VP}$, M/X, E, VDA, VPA, φ1 (OUT), φ2 (OUT) | $V_{OL}$ | — | 0.4 | V |
| Supply Current (No Load) | $I_{DD}$ | | 4 | mA/MHz |
| Standby Current (No Load, Data Bus = $V_{SS}$ or $V_{DD}$ $\overline{RES}$, $\overline{NMI}$, $\overline{IRQ}$, $\overline{SO}$, BE, $\overline{ABORT}$, φ2 = $V_{DD}$) | $I_{SB}$ | — | 10 | μA |
| Capacitance ($V_{IN}$ = 0V, $T_A$ = 25°C, f = 2 MHz) | | | | |
| Logic, φ2 (IN) | $C_{IN}$ | — | 10 | pF |
| Address, Data, R/$\overline{W}$ (Off State) | $C_{TS}$ | — | 15 | pF |

**Pin Function Table**

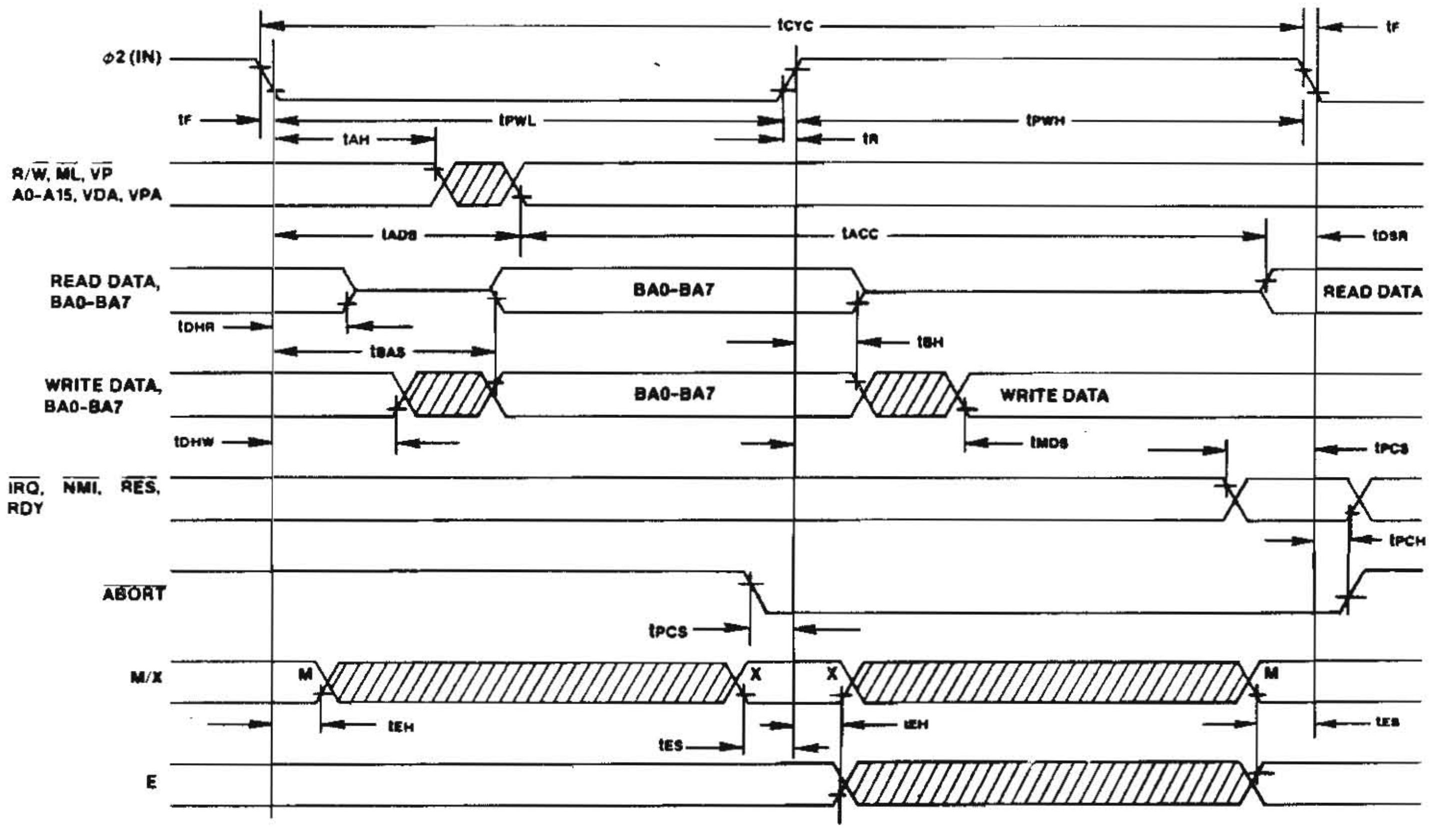
| Pin | Description |
|---|---|
| A0–A15 | Address Bus |
| $\overline{ABORT}$ | Abort Input |
| BE | Bus Enable |
| φ2 (IN) | Phase 2 In Clock |
| φ1 (OUT) | Phase 1 Out Clock |
| φ2 (OUT) | Phase 2 Out Clock |
| D0–D7 | Data Bus (G65SC802) |
| D0/BA0–D7/BA7 | Data Bus, Multiplexed (G65SC816) |
| E | Emulation Select |
| $\overline{IRQ}$ | Interrupt Request |
| $\overline{ML}$ | Memory Lock |
| M/X | Mode Select ($P_M$ or $P_X$) |

| Pin | Description |
|---|---|
| NC | No Connection |
| $\overline{NMI}$ | Non-Maskable Interrupt |
| RDY | Ready |
| $\overline{RES}$ | Reset |
| R/$\overline{W}$ | Read/Write |
| $\overline{SO}$ | Set Overflow |
| SYNC | Synchronize |
| VDA | Valid Data Address |
| $\overline{VP}$ | 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

| | | 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 |
| 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)



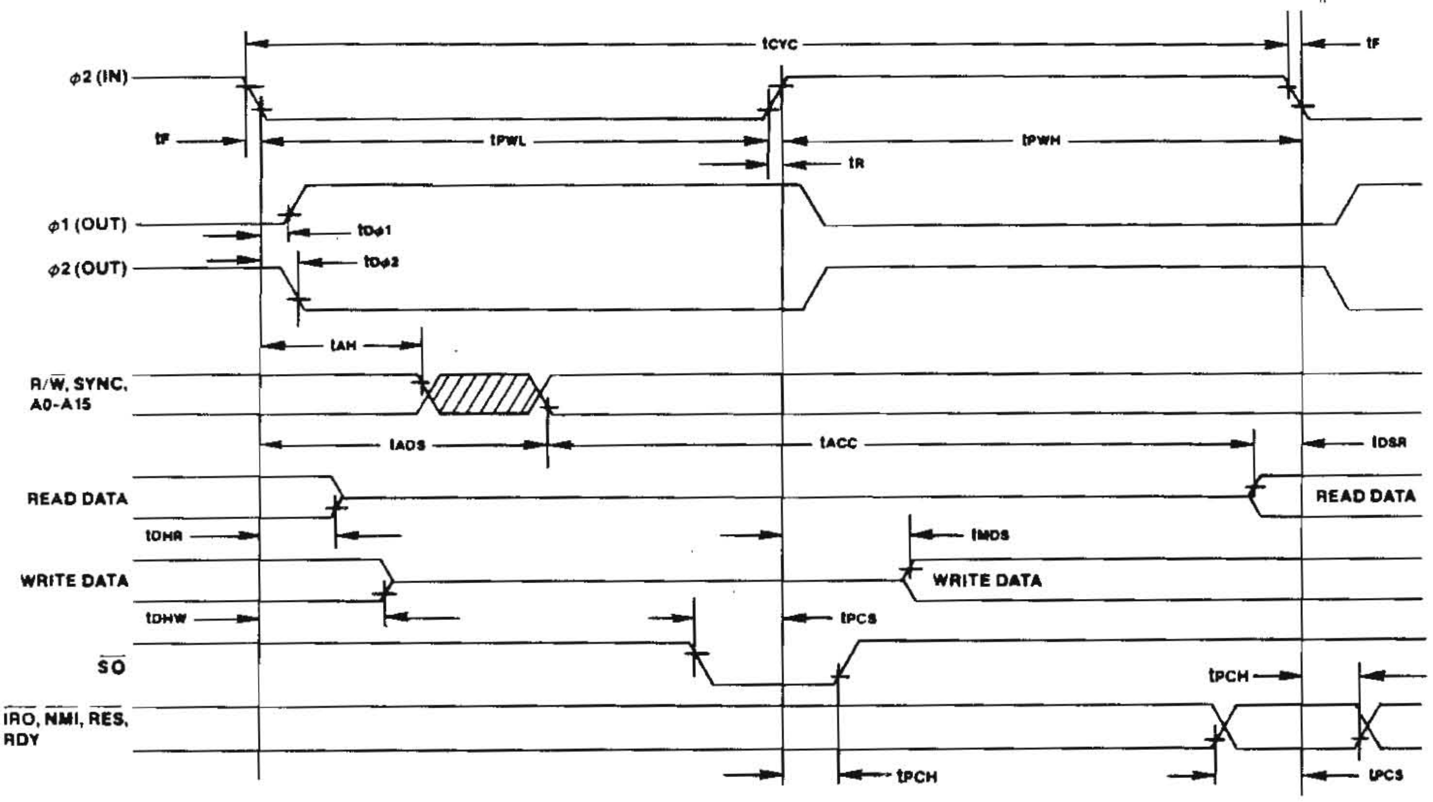
### Timing Notes:

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) | $t_{D\phi 1}$ | — | 20 | — | 20 | — | 20 | — | 20 | nS |
| Delay Time, φ2 (IN) to φ2 (OUT) | $t_{D\phi 2}$ | — | 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 chieving the added advantages of increased register lengths and faster xecution 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 equired to complete the instruction. Each data transfer between regis-ers 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.

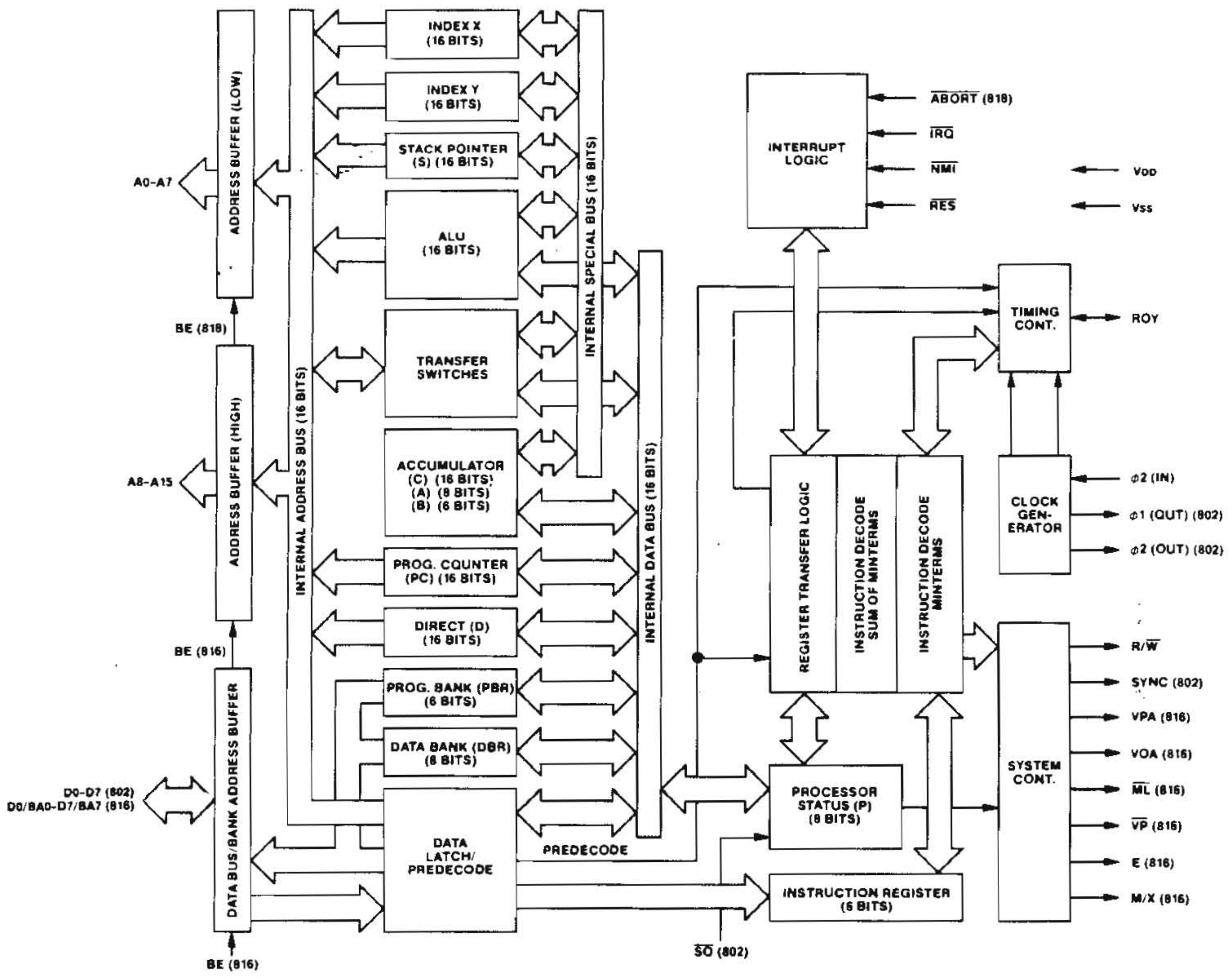


**Figure 1. Block Diagram — Internal Architecture**

## Signal Description

The following Signal Description applies to both the W65C802 and the W65C816 except as otherwise noted.

### Abort ($\overline{\text{ABORT}}$)—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 $\overline{\text{ABORT}}$ 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/$\overline{\text{W}}$ signal. With Bus Enable high, the R/$\overline{\text{W}}$ 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 ($\overline{\text{IRQ}}$)**
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 $\overline{\text{IRQ}}$ 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 ($\overline{\text{ML}}$)—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 ($\overline{\text{NMI}}$)**
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 $\overline{\text{NMI}}$ 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 $\overline{\text{NMI}}$ remains low.

**Phase 1 Out ($\phi$1 (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 ($\phi$2 (IN))**
This is the system clock input to the microprocessor internal clock generator (equivalent to $\phi$0 (IN) on the 6502). During the low power Standby Mode, $\phi$2 (IN) should be held in the high state to preserve the contents of internal registers.

**Phase 2 Out ($\phi$2 (OUT))—W65C802**
This clock output signal provides timing for external read and write operations. Addresses are valid (after the Address Setup Time ($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/$\overline{\text{W}}$)**
When the R/$\overline{\text{W}}$ 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/$\overline{\text{W}}$ 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 $\overline{\text{RES}}$, $\overline{\text{ABORT}}$, $\overline{\text{NMI}}$, or $\overline{\text{IRQ}}$ external interrupt is provided. This feature may be used to eliminate interrupt latency by placing the WAI instruction at the beginning of the $\overline{\text{IRQ}}$ servicing routine. If the $\overline{\text{IRQ}}$ Disable flag has been set, the next instruction will be executed when the $\overline{\text{IRQ}}$ 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 ($\overline{\text{RES}}$)**
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 $\overline{\text{RES}}$ signal must be held low for at least two clock cycles after $V_{DD}$ reaches operating voltage. Ready (RDY) has no effect while $\overline{\text{RES}}$ 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 | $\overline{\text{VP}}$ = 1 |
| R/$\overline{\text{W}}$ = 1 | VPA = 0 |
| SYNC = 0 | |

When Reset is brought high, an interrupt sequence is initiated:
- R/$\overline{\text{W}}$ remains in the high state during the stack address cycles.
- The Reset vector address is 00FFFC,D.

**Set Overflow ($\overline{\text{SO}}$)—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 ($\overline{\text{VP}}$)—W65C816**
The Vector Pull output indicates that a vector location is being addressed during an interrupt sequence. $\overline{\text{VP}}$ is low during the last two interrupt sequence cycles, during which time the processor reads the interrupt vector. The $\overline{\text{VP}}$ 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<br>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 $\overline{\text{IRQ}}$, $\overline{\text{NMI}}$, $\overline{\text{ABORT}}$. 00FFE6, 7 (E = 0) X = X on Stack always. | FFFE,F BRK bit = 0 on stack if $\overline{\text{IRQ}}$, $\overline{\text{NMI}}$. | FFFE,F BRK bit = 0 on stack if $\overline{\text{IRQ}}$, $\overline{\text{NMI}}$. |
| 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 ($\overline{\text{ML}}$) | $\overline{\text{ML}}$ = 0 during Read, Modify and Write cycles. | $\overline{\text{ML}}$ = 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/$\overline{\text{W}}$ During Read-Modify-Write Instructions | E = 1, R/$\overline{\text{W}}$ = 0 during Modify and Write cycles. E = 0, R/$\overline{\text{W}}$ = 0 only during Write cycle. | R/$\overline{\text{W}}$ = 0 only during Write cycle | R/$\overline{\text{W}}$ = 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 |

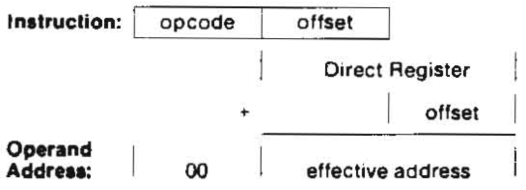
### 3. Absolute Long—al

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.



### 5. Accumulator—A

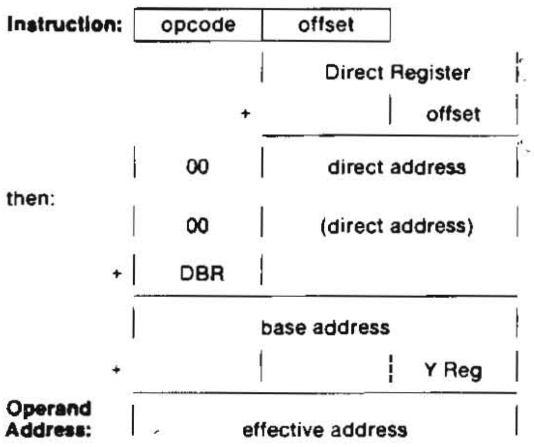
This form of addressing always uses a single byte instruction. The operand is the Accumulator.

### 6. Implied—i

Implied addressing uses a single byte instruction. The operand is implicitly defined by the instruction.

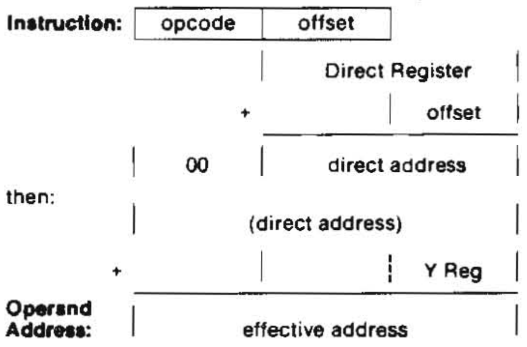
### 7. Direct Indirect Indexed—(d),y

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.



### 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.



### 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.

```
Instruction: | opcode | offset |

                      | Direct Register |
                    +          | offset |
                      ------------------
                      | direct address  |
                    +          | X Reg  |
                      ------------------
            |  00     |     address     |
then:
            |  00     |   (address)     |
          + |  DBR    |
            ----------------------------
Operand
Address:    |      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.

```
Instruction: | opcode | offset |

                      | Direct Register |
                    +          | offset |
                      ------------------
                      | direct address  |
                    +          | X Reg  |
                      ------------------
Operand
Address:    |  00     | effective address |
```

## 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.

```
Instruction: | opcode | offset |

                      | Direct Register |
                    +          | offset |
                      ------------------
                      | direct address  |
                    +          | Y Reg  |
                      ------------------
Operand
Address:    |  00     | effective address |
```

## 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.

```
Instruction: | opcode | addrl  | addrh  |

             |  DBR   | addrh  | addrl  |
                    + |        | X Reg  |
             ----------------------------
Operand
Address:     |    effective address     |
```

## 13. Absolute Long Indexed With X—al,x

The second, thı d anc' 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.

```
Instruction: | opcode | addrl  | addrh  | baddr |

             | baddr  | addrh  | addrl  |
                    + |        | X Reg  |
             ----------------------------
Operand
Address:     |    effective address     |
```

## 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.

```
Instruction: | opcode | addrl  | addrh  |

             |  DBR   | addrh  | addrl  |
                    + |        | Y Reg  |
             ----------------------------
Operand
Address:     |    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.

```
Instruction: | opcode | addrl  | addrh  |

  Indirect Address =  |  00    | addrh  | addrl  |
  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.

```
Instruction: | opcode | offset |

                      | Direct Register |
                    +          | offset |
                      ------------------
            |  00     | direct address  |
then:
            |  00     |(direct address) |
          + |  DBR    |
            ----------------------------
Operand
Address:    |      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.

```
Instruction: | opcode | offset |

                       |     Direct Register    |
                     +            |    offset   |
                                  ---------------
             |   00   |      direct address     |
then:
Operand
Address:     |          (direct 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.

```
Instruction: | opcode | addrl  | addrh  |

                      | addrh  | addrl  |
                      |        | X Reg  |
                      -------------------
             |  PBR   |     address     |
then:
             PC = (address)
```

## 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.

```
Instruction: | opcode | offset |

                      |   Stack Pointer   |
                    +          |  offset  |
                      ---------------------
Operand
Address:     |   00   | effective address |
```

## 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.

```
Instruction: | opcode | offset |

                       |    Stack Pointer    |
                     +            |  offset  |
                       -----------------------
              |  00   |      S + offset      |
then:
                       |      S + offset     |
            + |  DBR  |
              --------------------------------
              |         base address         |
            +          |          |  Y Reg   |
              --------------------------------
Operand
Address:      |      effective address       |
```

## 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.

```
Instruction: | opcode | dstbnk | srcbnk |

                        dstbnk  →  DBR
Source
Address:              | srcbnk |     X Reg     |
Destination
Address:              |  DBR   |     Y Reg     |
```

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 —$\overline{RESET}$ | 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<br>2 8 | ORA (d,x)<br>2 6 | COP s<br>2 ★ 8 | ORA d,s<br>2 ★ 4 | TSB d<br>2 ● 5 | ORA d<br>2 3 | ASL d<br>2 5 | ORA [d]<br>2 ★ 6 | PHP s<br>1 3 | ORA #<br>2 2 | ASL A<br>1 2 | PHD s<br>1 ★ 4 | TSB a<br>3 ● 6 | ORA a<br>3 4 | ASL a<br>3 6 | ORA al<br>4 ★ 5 | 0 |
| 1 | BPL r<br>2 2 | ORA (d),y<br>2 5 | ORA (d)<br>2 ● 5 | ORA (d,s),y<br>2 ★ 7 | TRB d<br>2 ● 5 | ORA d,x<br>2 4 | ASL d,x<br>2 6 | ORA [d],y<br>2 ★ 6 | CLC i<br>1 2 | ORA a,y<br>3 4 | INC A<br>1 ● 2 | TCS i<br>1 ★ 2 | TRB a<br>3 ● 6 | ORA a,x<br>3 4 | ASL a,x<br>3 7 | ORA al,x<br>4 ★ 5 | 1 |
| 2 | JSR a<br>3 6 | AND (d,x)<br>2 6 | JSL al<br>4 ★ 8 | AND d,s<br>2 ★ 4 | BIT d<br>2 3 | AND d<br>2 3 | ROL d<br>2 5 | AND [d]<br>2 ★ 6 | PLP s<br>1 4 | AND #<br>2 2 | ROL A<br>1 2 | PLD s<br>1 ★ 5 | BIT a<br>3 4 | AND a<br>3 4 | ROL a<br>3 6 | AND al<br>4 ★ 5 | 2 |
| 3 | BMI r<br>2 2 | AND (d),y<br>2 5 | AND (d)<br>2 ● 5 | AND (d,s),y<br>2 ★ 7 | BIT d,x<br>2 ● 4 | AND d,x<br>2 4 | ROL d,x<br>2 6 | AND [d],y<br>2 ★ 6 | SEC i<br>1 2 | AND a,y<br>3 4 | DEC A<br>1 ● 2 | TSC i<br>1 ★ 2 | BIT a,x<br>3 ● 4 | AND a,x<br>3 4 | ROL a,x<br>3 7 | AND al,x<br>4 ★ 5 | 3 |
| 4 | RTI s<br>1 7 | EOR (d,x)<br>2 6 | WDM<br>2 ★ 2 | EOR d,s<br>2 ★ 4 | MVP xyc<br>3 ★ 7 | EOR d<br>2 3 | LSR d<br>2 5 | EOR [d]<br>2 ★ 6 | PHA s<br>1 3 | EOR #<br>2 2 | LSR A<br>1 2 | PHK s<br>1 ★ 3 | JMP a<br>3 3 | EOR a<br>3 4 | LSR a<br>3 6 | EOR al<br>4 ★ 5 | 4 |
| 5 | BVC r<br>2 2 | EOR (d),y<br>2 5 | EOR (d)<br>2 ● 5 | EOR (d,s),y<br>2 ★ 7 | MVN xyc<br>3 ★ 7 | EOR d,x<br>2 4 | LSR d,x<br>2 6 | EOR [d],y<br>2 ★ 6 | CLI i<br>1 2 | EOR a,y<br>3 4 | PHY s<br>1 ● 3 | TCD i<br>1 ★ 2 | JMP al<br>4 ★ 4 | EOR a,x<br>3 4 | LSR a,x<br>3 7 | EOR al,x<br>4 ★ 5 | 5 |
| 6 | RTS s<br>1 6 | ADC (d,x)<br>2 6 | PER s<br>3 ★ 6 | ADC d,s<br>2 ★ 4 | STZ d<br>2 ● 3 | ADC d<br>2 3 | ROR d<br>2 5 | ADC [d]<br>2 ★ 6 | PLA s<br>1 4 | ADC #<br>2 2 | ROR A<br>1 2 | RTL s<br>1 ★ 6 | JMP (a)<br>3 5 | ADC a<br>3 4 | ROR a<br>3 6 | ADC al<br>4 ★ 5 | 6 |
| 7 | BVS r<br>2 2 | ADC (d),y<br>2 5 | ADC (d)<br>2 ● 5 | ADC (d,s),y<br>2 ★ 7 | STZ d,x<br>2 ● 4 | ADC d,x<br>2 4 | ROR d,x<br>2 6 | ADC [d],y<br>2 ★ 6 | SEI i<br>1 2 | ADC a,y<br>3 4 | PLY s<br>1 ● 4 | TDC i<br>1 ★ 2 | JMP (a,x)<br>3 ● 6 | ADC a,x<br>3 4 | ROR a,x<br>3 7 | ADC al,x<br>4 ★ 5 | 7 |
| 8 | BRA r<br>2 ● 2 | STA (d,x)<br>2 6 | BRL rl<br>3 ★ 3 | STA d,s<br>2 ★ 4 | STY d<br>2 3 | STA d<br>2 3 | STX d<br>2 3 | STA [d]<br>2 ★ 6 | DEY i<br>1 2 | BIT #<br>2 ● 2 | TXA i<br>1 2 | PHB s<br>1 ★ 3 | STY a<br>3 4 | STA a<br>3 4 | STX a<br>3 4 | STA al<br>4 ★ 5 | 8 |
| 9 | BCC r<br>2 2 | STA (d),y<br>2 6 | STA (d)<br>2 ● 5 | STA (d,s),y<br>2 ★ 7 | STY d,x<br>2 4 | STA d,x<br>2 4 | STX d,y<br>2 4 | STA [d],y<br>2 ★ 6 | TYA i<br>1 2 | STA a,y<br>3 5 | TXS i<br>1 2 | TXY i<br>1 ★ 2 | STZ a<br>3 ● 4 | STA a,x<br>3 5 | STZ a,x<br>3 ● 5 | STA al,x<br>4 ★ 5 | 9 |
| A | LDY #<br>2 2 | LDA (d,x)<br>2 6 | LDX #<br>2 2 | LDA d,s<br>2 ★ 4 | LDY d<br>2 3 | LDA d<br>2 3 | LDX d<br>2 3 | LDA [d]<br>2 ★ 6 | TAY i<br>1 2 | LDA #<br>2 2 | TAX i<br>1 2 | PLB s<br>1 ★ 4 | LDY a<br>3 4 | LDA a<br>3 4 | LDX a<br>3 4 | LDA al<br>4 ★ 5 | A |
| B | BCS r<br>2 2 | LDA (d),y<br>2 5 | LDA (d)<br>2 ● 5 | LDA (d,s),y<br>2 ★ 7 | LDY d,x<br>2 4 | LDA d,x<br>2 4 | LDX d,y<br>2 4 | LDA [d],y<br>2 ★ 6 | CLV i<br>1 2 | LDA a,y<br>3 4 | TSX i<br>1 2 | TYX i<br>1 ★ 2 | LDY a,x<br>3 4 | LDA a,x<br>3 4 | LDX a,y<br>3 4 | LDA al,x<br>4 ★ 5 | B |
| C | CPY #<br>2 2 | CMP (d,x)<br>2 6 | REP #<br>2 ★ 3 | CMP d,s<br>2 ★ 4 | CPY d<br>2 3 | CMP d<br>2 3 | DEC d<br>2 5 | CMP [d]<br>2 ★ 6 | INY i<br>1 2 | CMP #<br>2 2 | DEX i<br>1 2 | WAI i<br>1 ● 3 | CPY a<br>3 4 | CMP a<br>3 4 | DEC a<br>3 6 | CMP al<br>4 ★ 5 | C |
| D | BNE r<br>2 2 | CMP (d),y<br>2 5 | CMP (d)<br>2 ● 5 | CMP (d,s),y<br>2 ★ 7 | PEI s<br>2 ★ 6 | CMP d,x<br>2 4 | DEC d,x<br>2 6 | CMP [d],y<br>2 ★ 6 | CLD i<br>1 2 | CMP a,y<br>3 4 | PHX s<br>1 ● 3 | STP i<br>1 ● 3 | JML (a)<br>3 ★ 6 | CMP a,x<br>3 4 | DEC a,x<br>3 7 | CMP al,x<br>4 ★ 5 | D |
| E | CPX #<br>2 2 | SBC (d,x)<br>2 6 | SEP #<br>2 ★ 3 | SBC d,s<br>2 ★ 4 | CPX d<br>2 3 | SBC d<br>2 3 | INC d<br>2 5 | SBC [d]<br>2 ★ 6 | INX i<br>1 2 | SBC #<br>2 2 | NOP i<br>1 2 | XBA i<br>1 ★ 3 | CPX a<br>3 4 | SBC a<br>3 4 | INC a<br>3 6 | SBC al<br>4 ★ 5 | E |
| F | BEO r<br>2 2 | SBC (d),y<br>2 5 | SBC (d)<br>2 ● 5 | SBC (d,s),y<br>2 ★ 7 | PEA s<br>3 ★ 5 | SBC d,x<br>2 4 | INC d,x<br>2 6 | SBC [d],y<br>2 ★ 6 | SED i<br>1 2 | SBC a,y<br>3 4 | PLX s<br>1 ● 4 | XCE i<br>1 ★ 2 | JSR (a,x)<br>3 ★ 6 | SBC a,x<br>3 4 | INC a,x<br>3 7 | SBC al,x<br>4 ★ 5 | F |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |  |

| 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 | ★ = New W65C816/802 Opcodes<br>● = New W65C02 Opcodes<br>Blank = NMOS 6502 Opcodes | ADDRESSING MODE |
| BASE NO. BYTES | | 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 | |
| | OPERATION | 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 | A ∧ M (NOTE 1) | 89 | 2C | | 24 | | | | | | 34 | | 3C | | | | | | | | | | | | | $M_7$ | $M_6$ | . | . | . | . | 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 | | | | | | | | | | | | | | | | | | | . | 0 | . | . | . | . | . | . | | 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 | | | 51 | 57 | 41 | 55 | | 5D | 5F | 59 | | | | 52 | 47 | | | 43 | 53 | | N | . | . | . | . | . | Z | . | | EOR |
| INC | INCREMENTS | | EE | | E6 | 1A | | | | | 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 | A V M → 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 | M∧P → 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

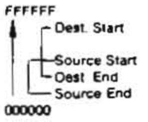
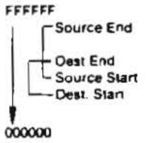
+ Add
- Subtract
∧ AND
V OR
⊻ Exclusive OR

## Table 6. Detailed Instruction Operation

| | ADDRESS MODE | | CYCLE | $\overline{VP}$, | $\overline{ML}$ | VDA | VPA | | ADDRESS BUS | DATA BUS | R/$\overline{W}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | IOL | 1 |
| | AND.EOR.ADC.BIT.LDA. | (1)(8) | 2a. | 1 | 1 | 0 | 1 | | PBR.PC+2 | IOH | 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 cyclas) | (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) | $\overline{IRQ}$.$\overline{NMI}$ | 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) | $\overline{RES}$=1 | 3 | 1 | 1 | 0 | 0 | 1 | PBR.PC+1 | IO | 1 |
| | (3 cycles) | $\overline{RES}$=0 | 1c | 1 | 1 | 0 | 0 | 1 | PBR.PC+1 | RES(BRK) | 1 |
| | | $\overline{RES}$=0 | 1b | 1 | 1 | 0 | 0 | 1 | PBR.PC+1 | RES(BRK) | 1 |
| | | $\overline{RES}$=1 | 1a | 1 | 1 | 0 | 0 | 1 | PBR.PC+1 | RES(BRK) | 1 |
| | See 21a Stack | | 1 | 1 | 1 | 1 | 1 | 1 | PBR.PC+1 | BEGIN | 1 |
| | (Hardware interrupt) | | | | | | | | | | |

| | ADDRESS MODE | | CYCLE | $\overline{VP}$, | $\overline{ML}$ | VDA | VPA | ADDRESS BUS | DATA BUS | R/$\overline{W}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 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+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 |
| 6 | 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 | Date Low | 1/0 |

## Table 6. Detailed Instruction Operation (continued)

| | ADDRESS MODE | | CYCLE | $\overline{VP}$ | $\overline{ML}$ | VDA | VPA | ADDRESS BUS | DATA BUS | R/$\overline{W}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| ★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 Codes) | (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+ | 0 |
| | (3 bytes) | | | | | | | | CARRY | |
| | (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) | | | | | | | | | |
| ★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,SBC) | | 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 |
| |  | | 1 | 1 | 1 | 1 | 1 | PBR,PC | Op Code | 1 |
| | | | 2 | 1 | 1 | 0 | 1 | PBR,PC+1 | DBA | 1 |
| | | N Byte | 3 | 1 | 1 | 0 | 1 | PBR,PC+2 | SBA | 1 |
| | | Last | 4 | 1 | 1 | 1 | 0 | SBA,X-2 | Source Data | 1 |
| | | C=0 | 5 | 1 | 1 | 1 | 0 | DBA,Y-2 | Dest Data | 0 |
| | | | 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 |
| |  | N-1 | 3 | 1 | 1 | 0 | 1 | PBR,PC+2 | SBA | 1 |
| | | Byte | 4 | 1 | 1 | 1 | 0 | SBA,X+1 | Source Data | 1 |
| | | C=1 | 5 | 1 | 1 | 1 | 0 | DBA,Y+1 | Dest Data | 0 |
| | | | 6 | 1 | 1 | 0 | 0 | DBA,Y+1 | IO | 1 |
| | | | 7 | 1 | 1 | 0 | 0 | DBA,Y+1 | IO | 1 |
| | | | 1 | 1 | 1 | 1 | 1 | PBR,PC | Op Code | 1 |
| | | | 2 | 1 | 1 | 0 | 1 | PBR,PC+1 | DBA | 1 |
| | | N Byte | 3 | 1 | 1 | 0 | 1 | PBR,PC+2 | SBA | 1 |
| | MVN is used when the | C=0 | 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 | |
| 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**

| | Instruction Times In Memory Cycles | | Memory Utilization In Number of Program Sequence Bytes | |
|---|---|---|---|---|
| **Address Mode** | **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.
- **$\overline{\text{IRQ}}$, $\overline{\text{NMI}}$, $\overline{\text{ABORT}}$ 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.

## Additional Information
For additional information on the W65C802/816, refer to the following publications:

**Programming the 65816**
William Labiak
SYBEX, Inc.
2344 Sixth St.
Berkeley, CA 94710

**The 6502, 65C02 and 65816 Handbook**
Steve Hendrix
Weber Systems, Inc.
8437 Mayfield Rd.
Chesterland, OH 44026

**65816/65802 Assembly Language Programming**
Michael Fisher
Osborne McGraw-Hill
2600 Tenth St.
Berkeley, CA 94710

**Programming the 65816 Including the 6502, 65C02, and 65802**
David Eyes and Ron Lichty
Prentice Hall Press
A Division of Simon & Schuster, Inc.
Gulf & Western Bldg.
One Gulf & Western Plaza
New York, NY 10023

# 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.

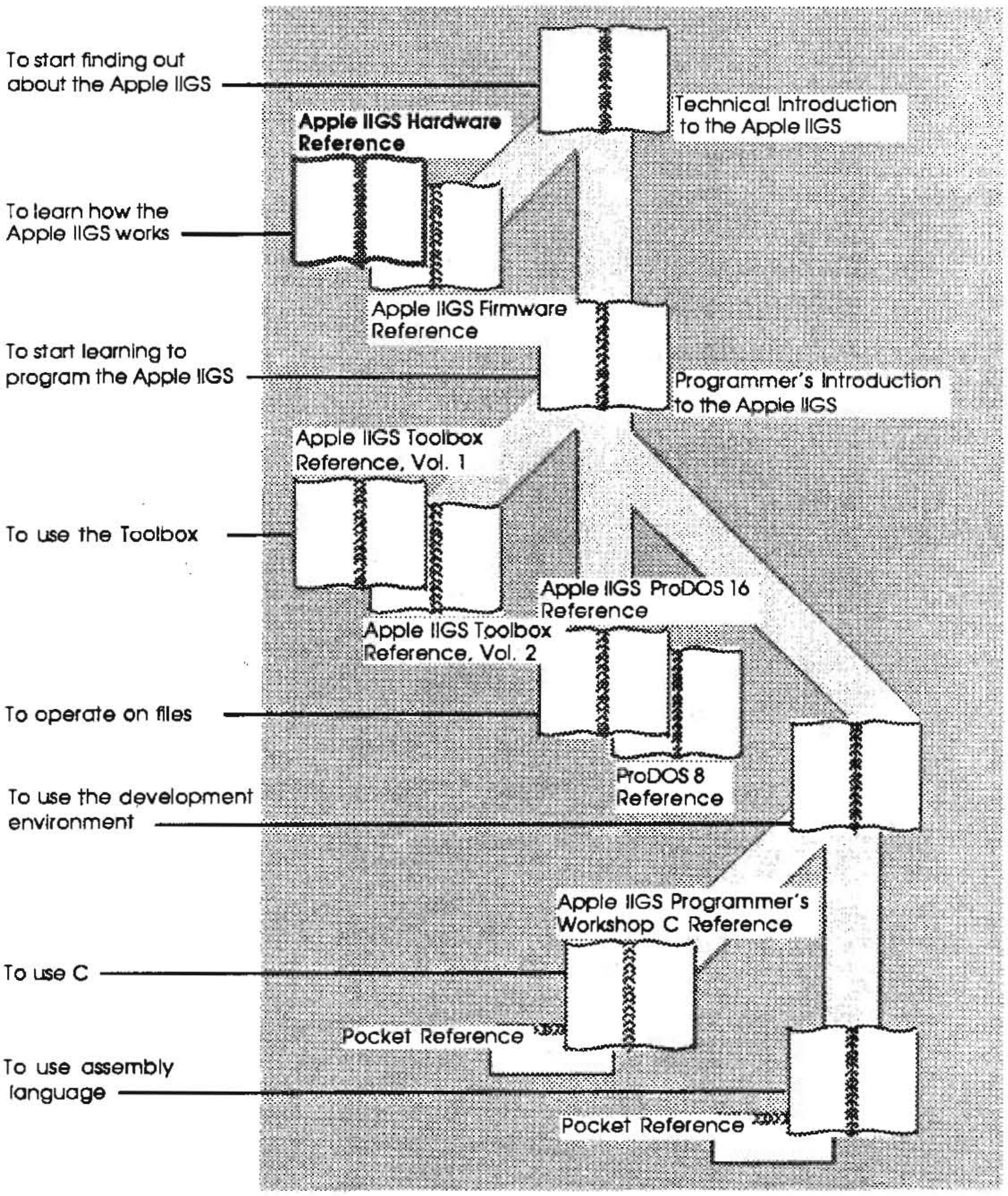
The technical manuals are listed in Table A-1. Figure A-1 is a diagram showing the relationships among the different manuals.

**Table A-1**
The Apple IIGS technical manuals

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

*There is a Pocket Reference for each of these.

**Figure A-1**
Roadmap to the technical manuals


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

## Introductory manuals

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

### The technical introduction

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

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

### The programmer's introduction

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

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

## Machine reference manuals

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

### The hardware reference manual

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

### The firmware reference manual

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

## The toolbox manuals

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

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

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

## The Programmer's Workshop manual

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

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

## Programming-language manuals

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

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

## Operating-system manuals

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

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

## All-Apple manuals

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

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

The *Apple Numerics Manual* is the reference for the Standard Apple Numeric Environment (SANE), a full implementation of the IEEE standard floating-point arithmetic. The functions of the Apple IIGS SANE tool set match those of the Macintosh SANE package and of the 6502 assembly language SANE software. If your application requires accurate arithmetic, you'll probably want to use the SANE routines in the Apple IIGS. The *Apple IIGS 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. A description of the version of the SANE routines for the 65C816 is available through the Apple Programmer's and Developer's Association, administered by the A.P.P.L.E. cooperative in Renton, Washington.

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

# Appendix B

# 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 countires. The ADB keyboard on the Apple IIGS is available in the following versions:

- U.S.A. English
- U.K. English
- Canadian
- French
- German
- Italian
- Spanish
- Swedish
- U.S.A. Dvorak

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. Figures A-1 through A-9 show the legends on the different keyboards.

**Figure A-1**
U.S.A. English keyboard

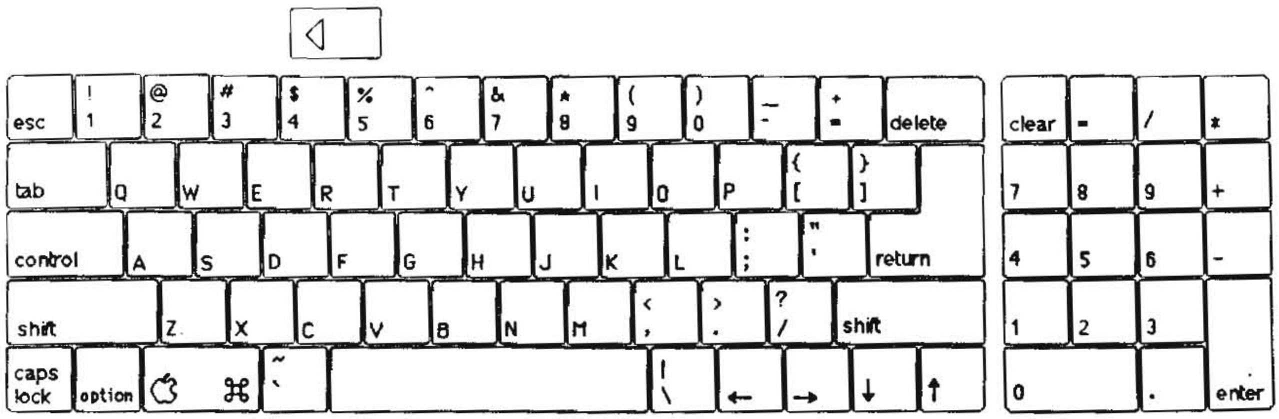


**Figure A-2**
U.K. English keyboard

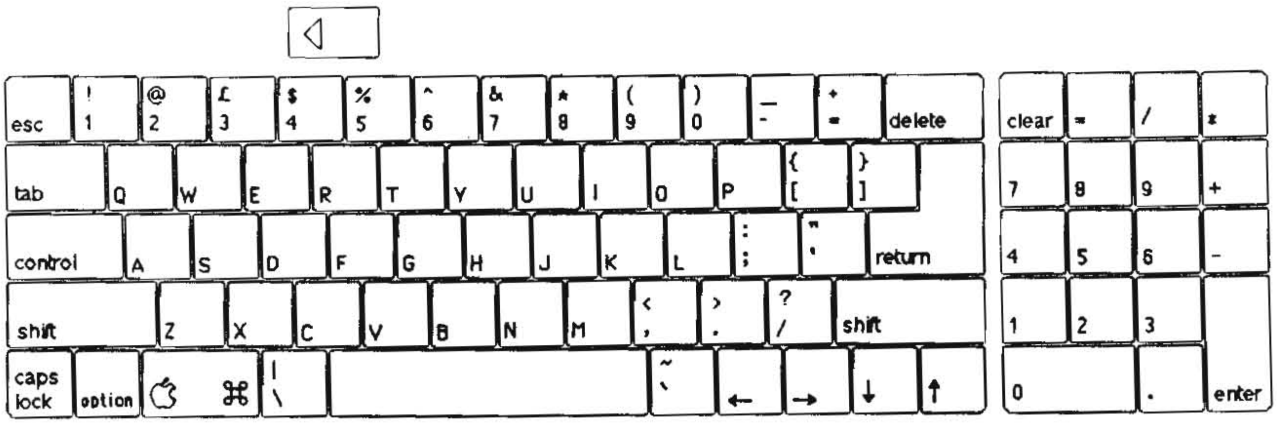


**Figure A-3**
Canadian keyboard

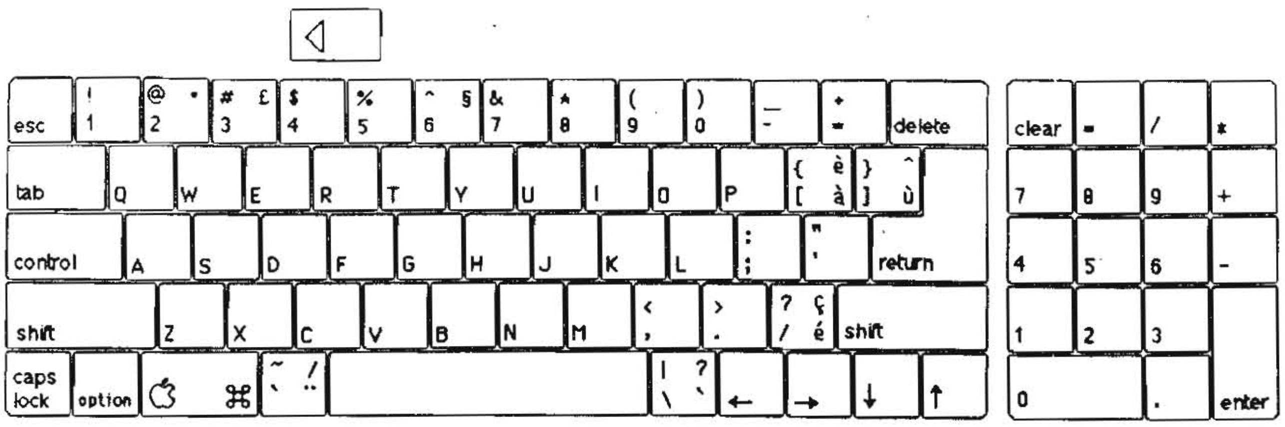


**Figure A-4**
French keyboard

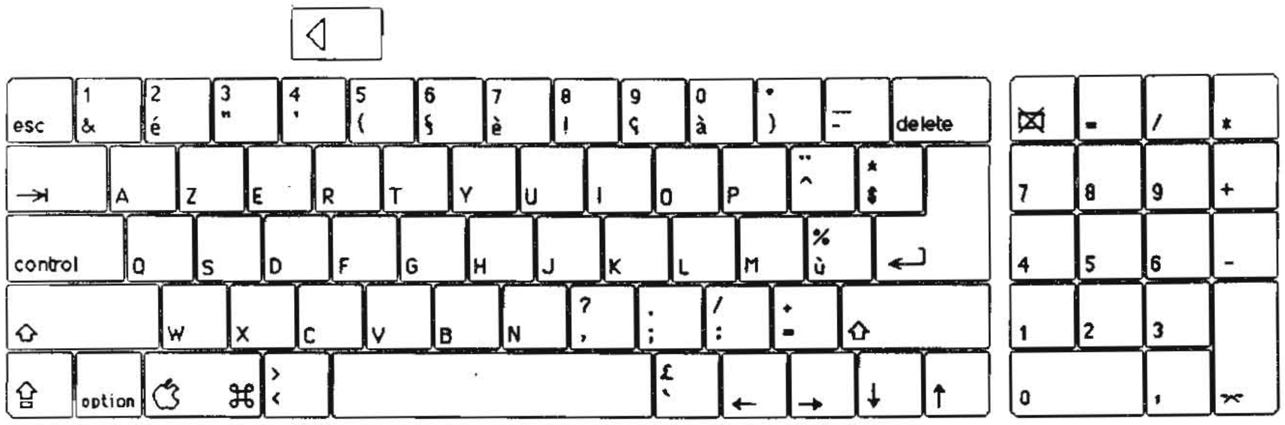


**Figure A-5**
German keyboard

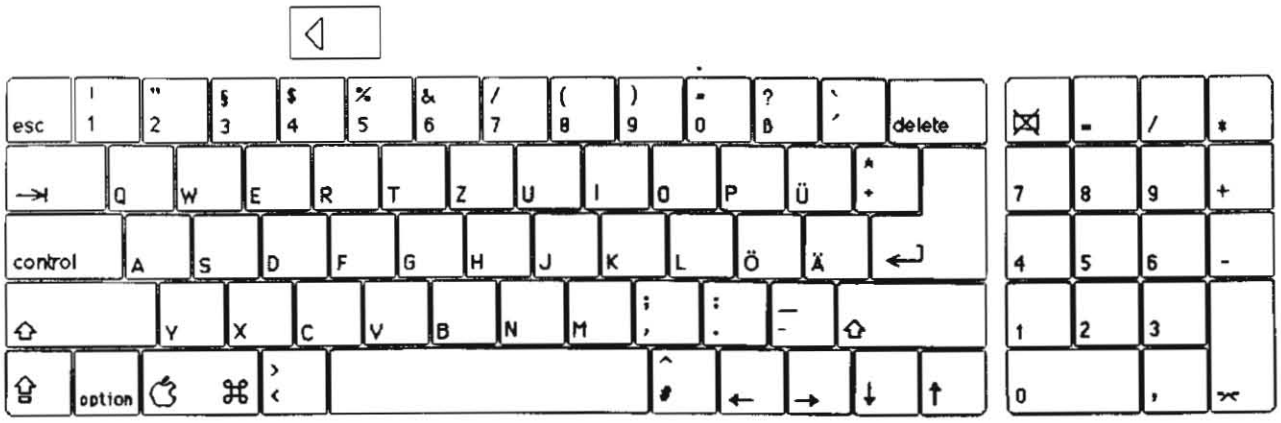


**Figure A-6**
Italian keyboard

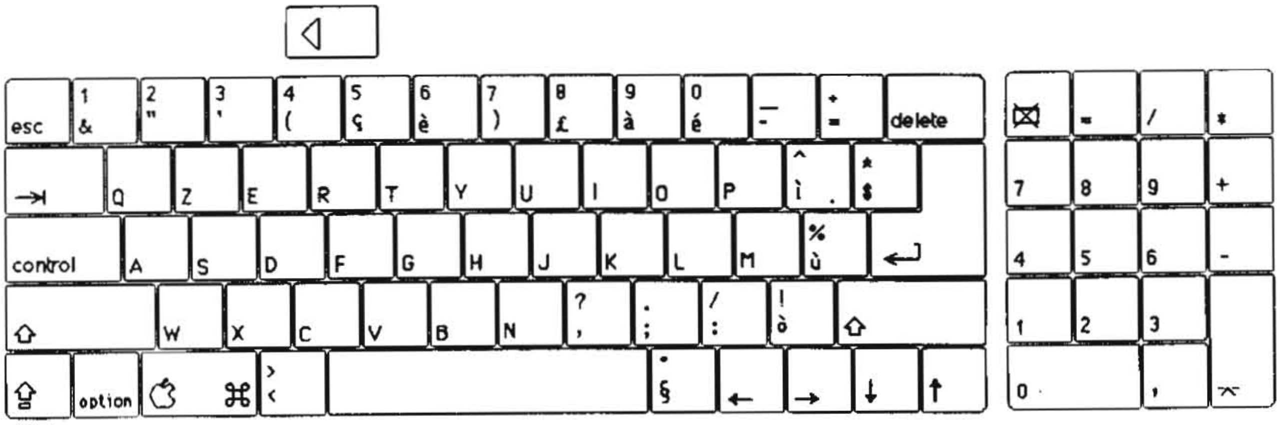


**Figure A-7**
Spanish keyboard

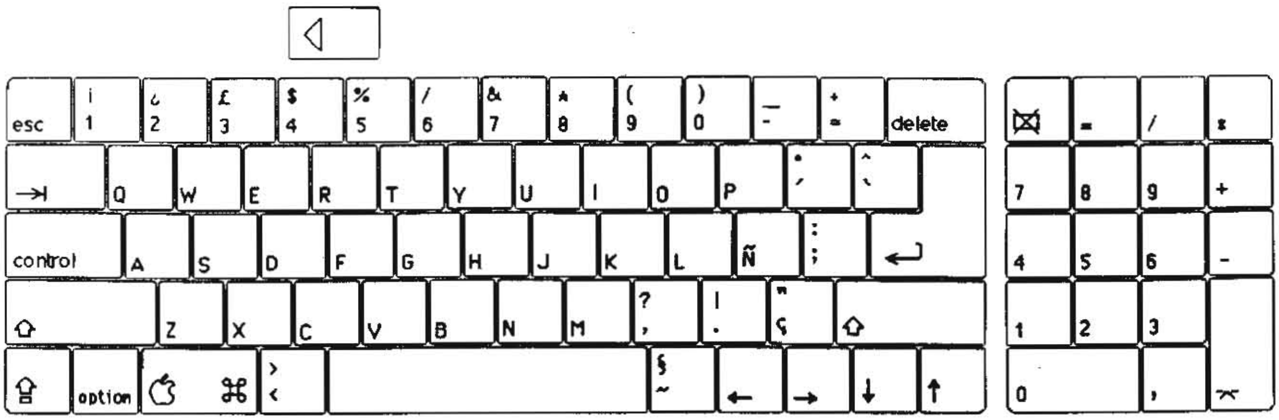


**Figure A-8**
Swedish keyboard

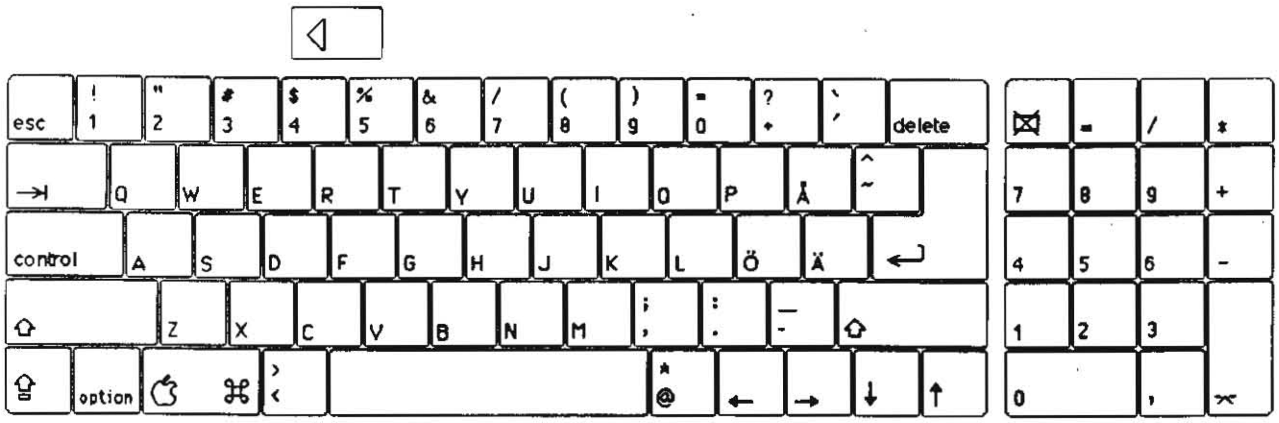
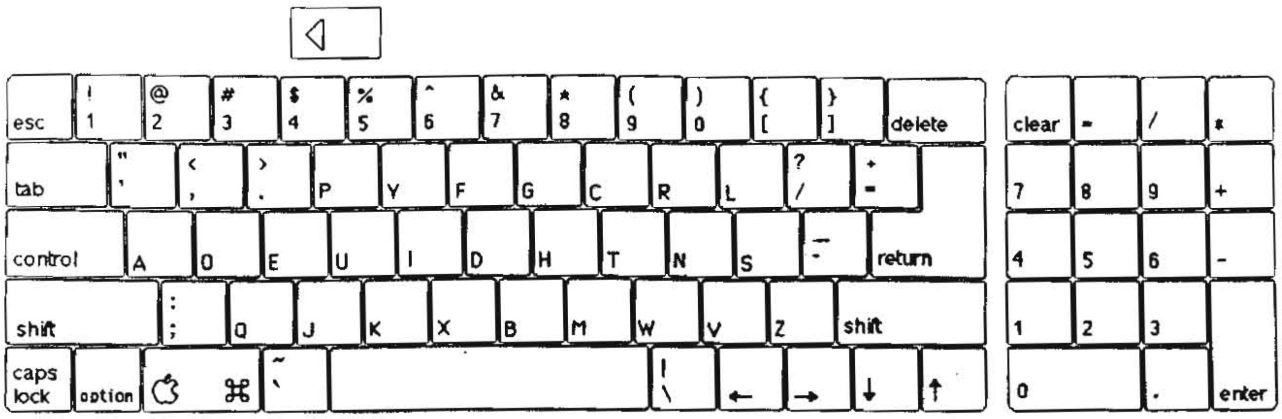


**Figure A-9**
U.S.A. Dvorak keyboard



# Appendix C

# 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 displays, refer to the QuickDraw II tool set in *Apple IIgs Toolbox Reference,* Volume 1.

## Character Generator ROM

The ROM contains the dot patterns making up the characters in the 40-column and 80-column displays.

### U.S. Characters

Figures C-1, C-2, C-3 show the characters for the U.S. versions of the computer.

**Figure C-1**
Uppercase characters

```
@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _
```

**Figure C-2**
Lowercase characters

```
` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
```

**Figure C-3**
Special characters

```
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
```

## 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. Figure C-4 shows those characters.

**Figure C-4**

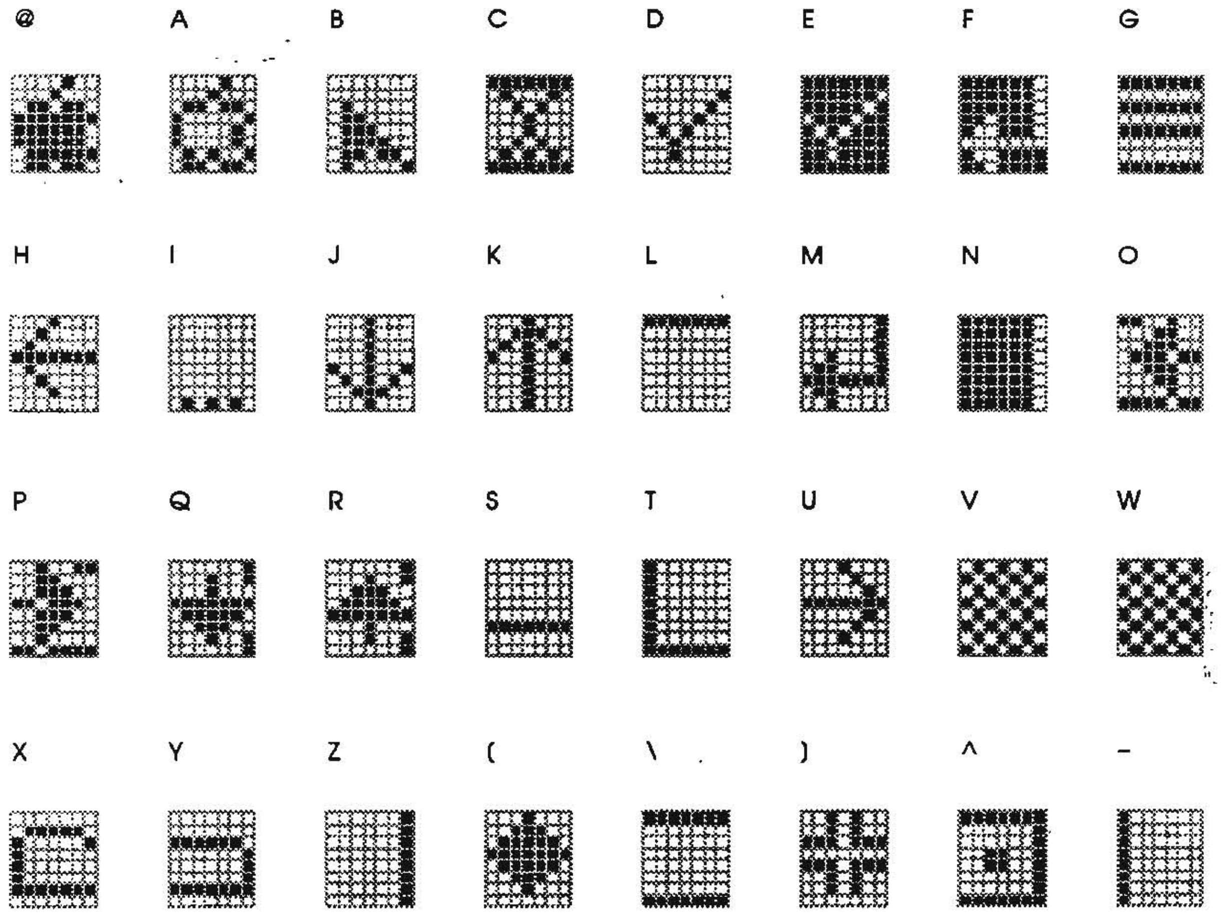
International characters

| Language | Equivalent characters | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| U.S.A. English | # | @ | [ | \ | ] | ` | { | \| | } | ~ |
| U.K. English | £ | @ | [ | \ | ] | ` | { | \| | } | ~ |
| French | £ | à | " | ç | § | ` | é | ù | è | ¨ |
| Danish | # | @ | | Ø | Å | ` | | 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 | ~ |

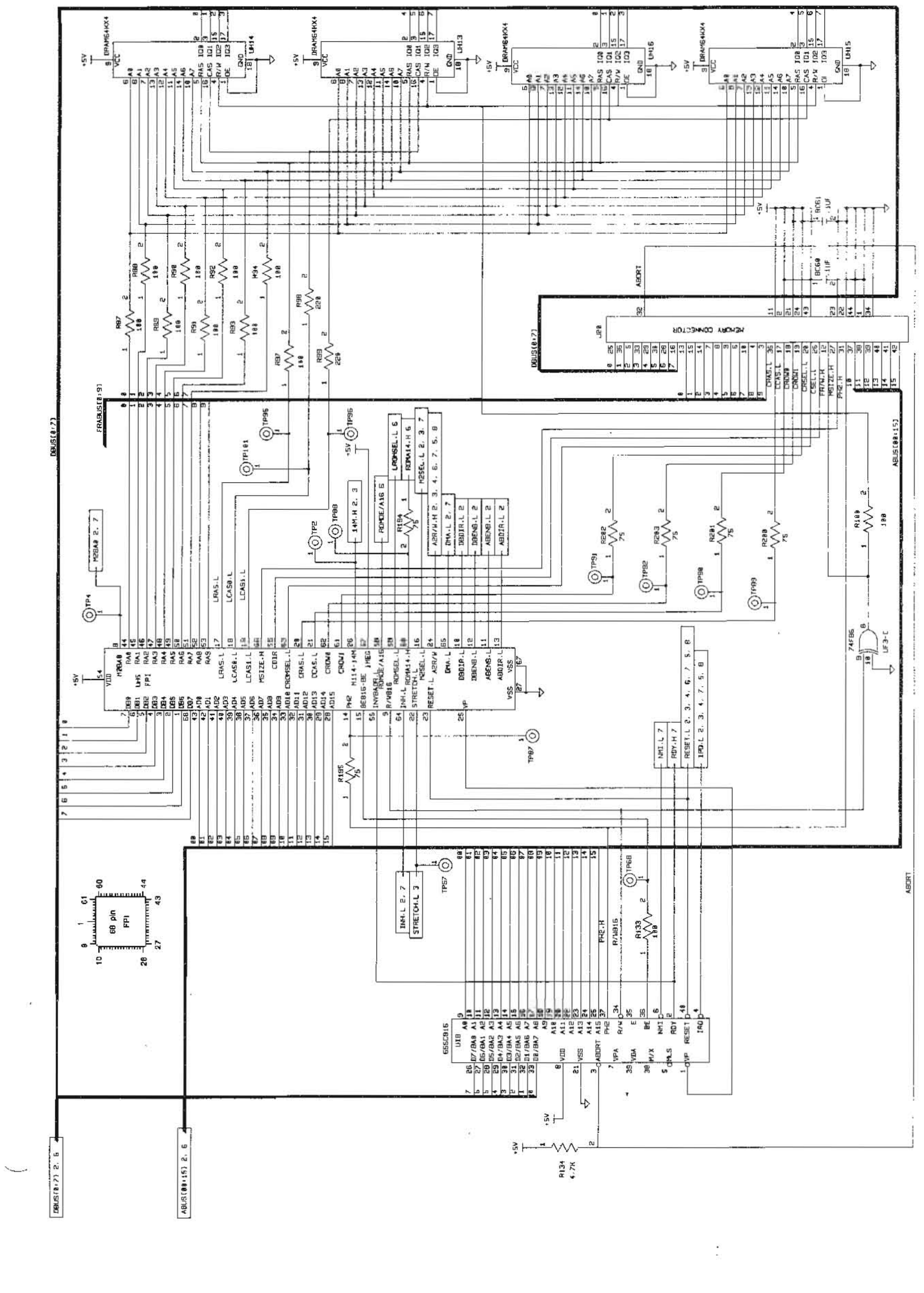
## MouseText characters

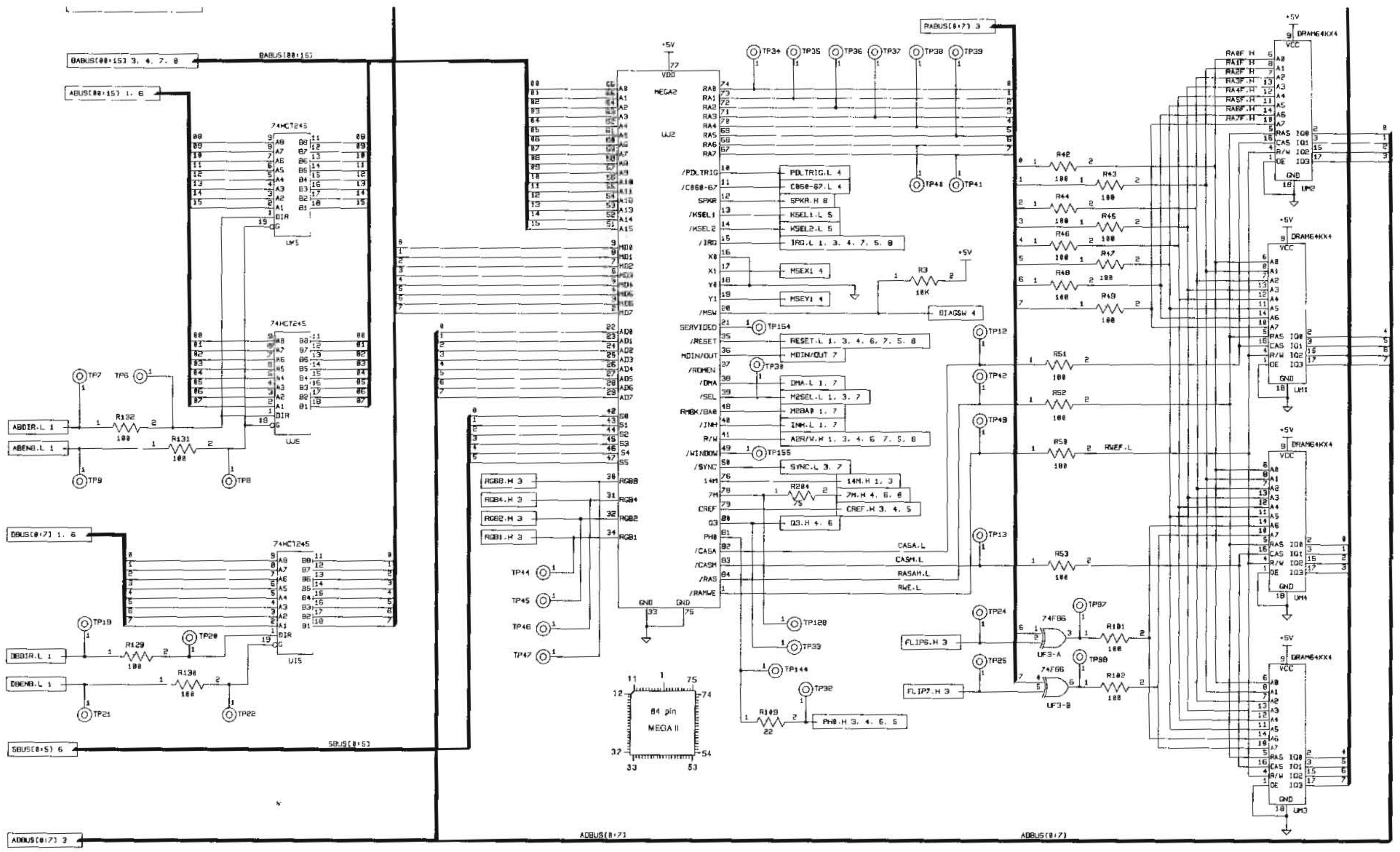
The character ROM includes several graphic characters used in displaying the desktop user interface in text mode. Figure C-5 shows those characters.

**Figure C-5**
MouseText characters

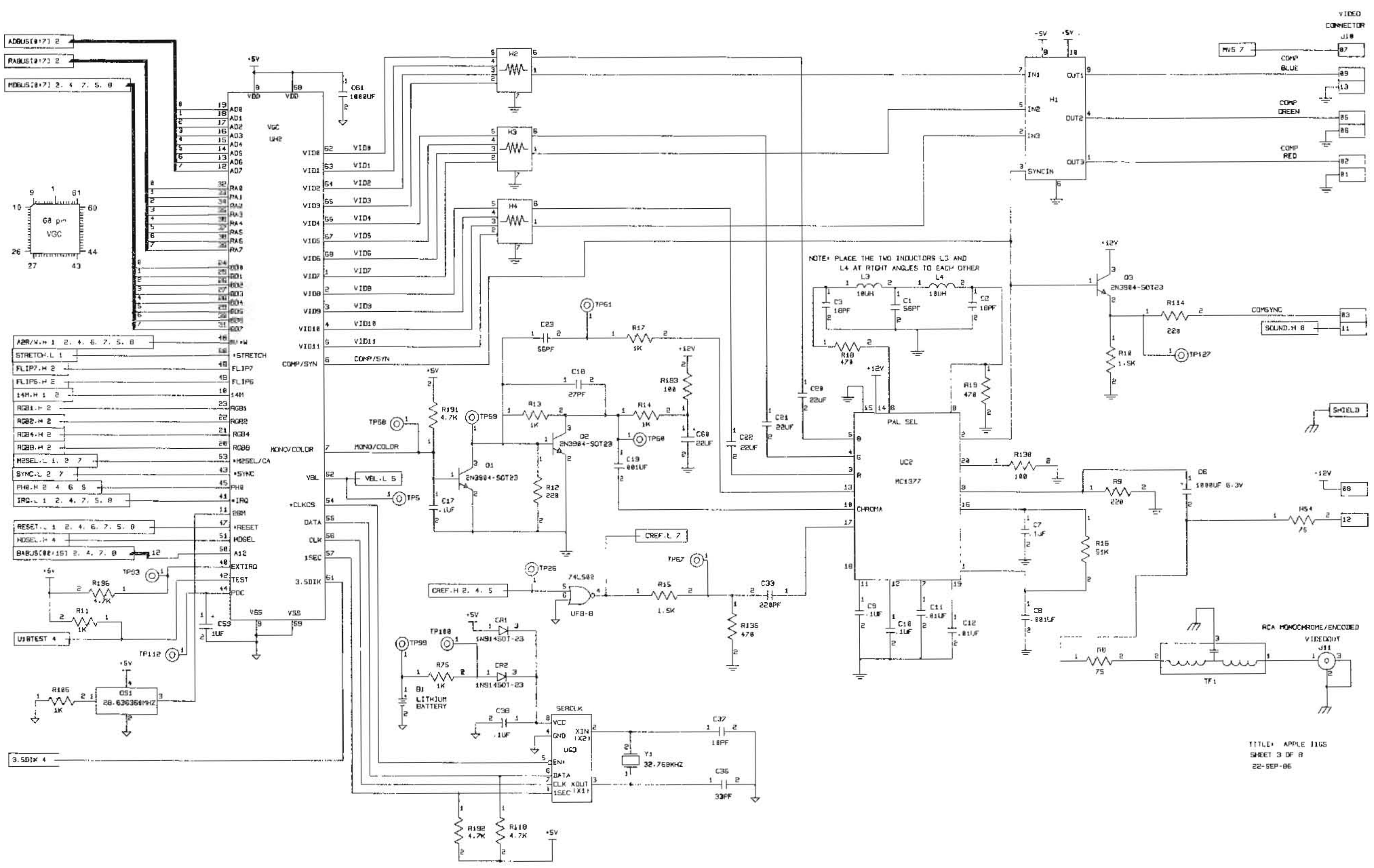


TITLE: APPLE IIGS
SHEET 1 OF 8
22-SEP-86

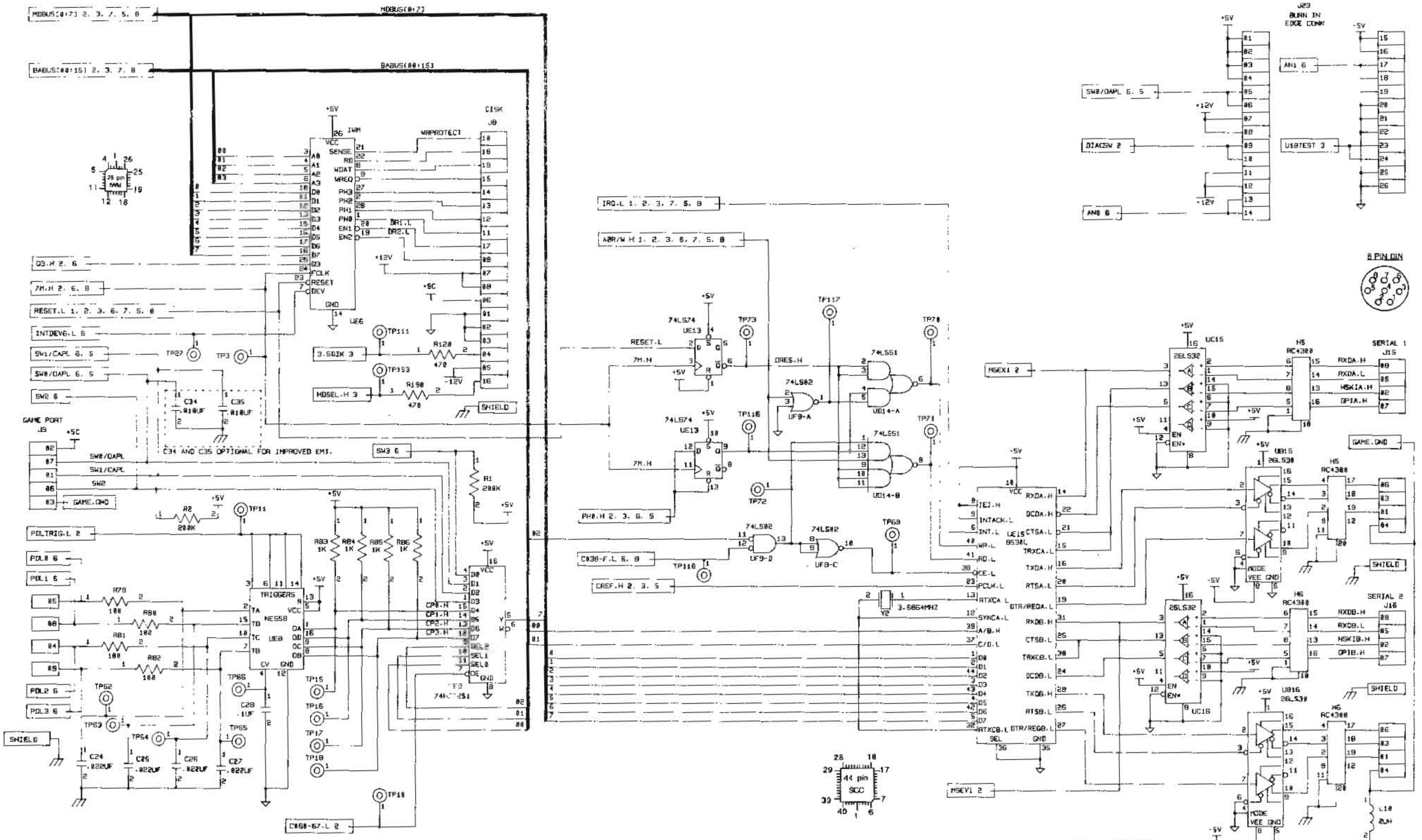




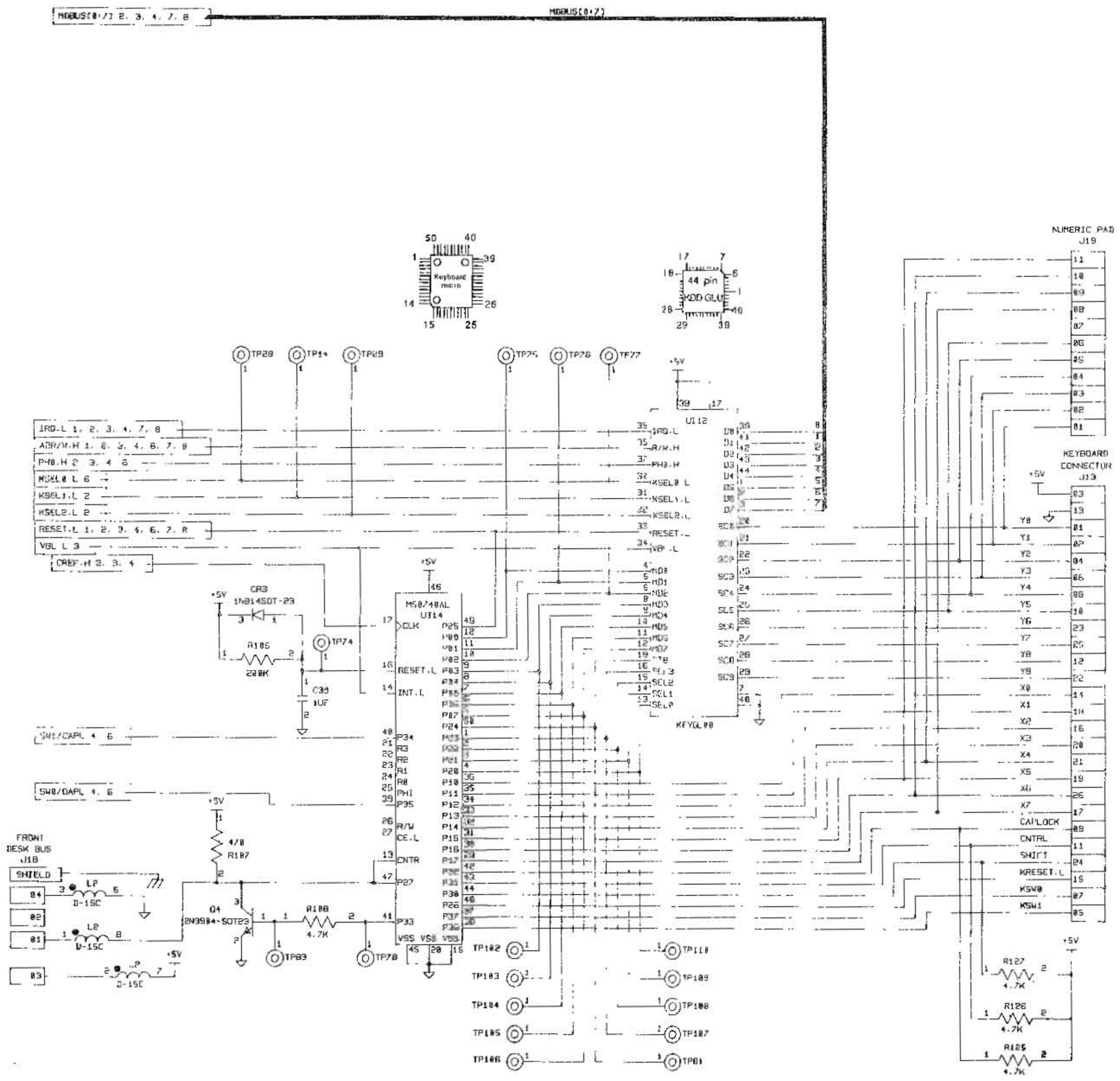
TITLE: APPLE IIGS
SHEET 2 OF 8
22-SEP-86



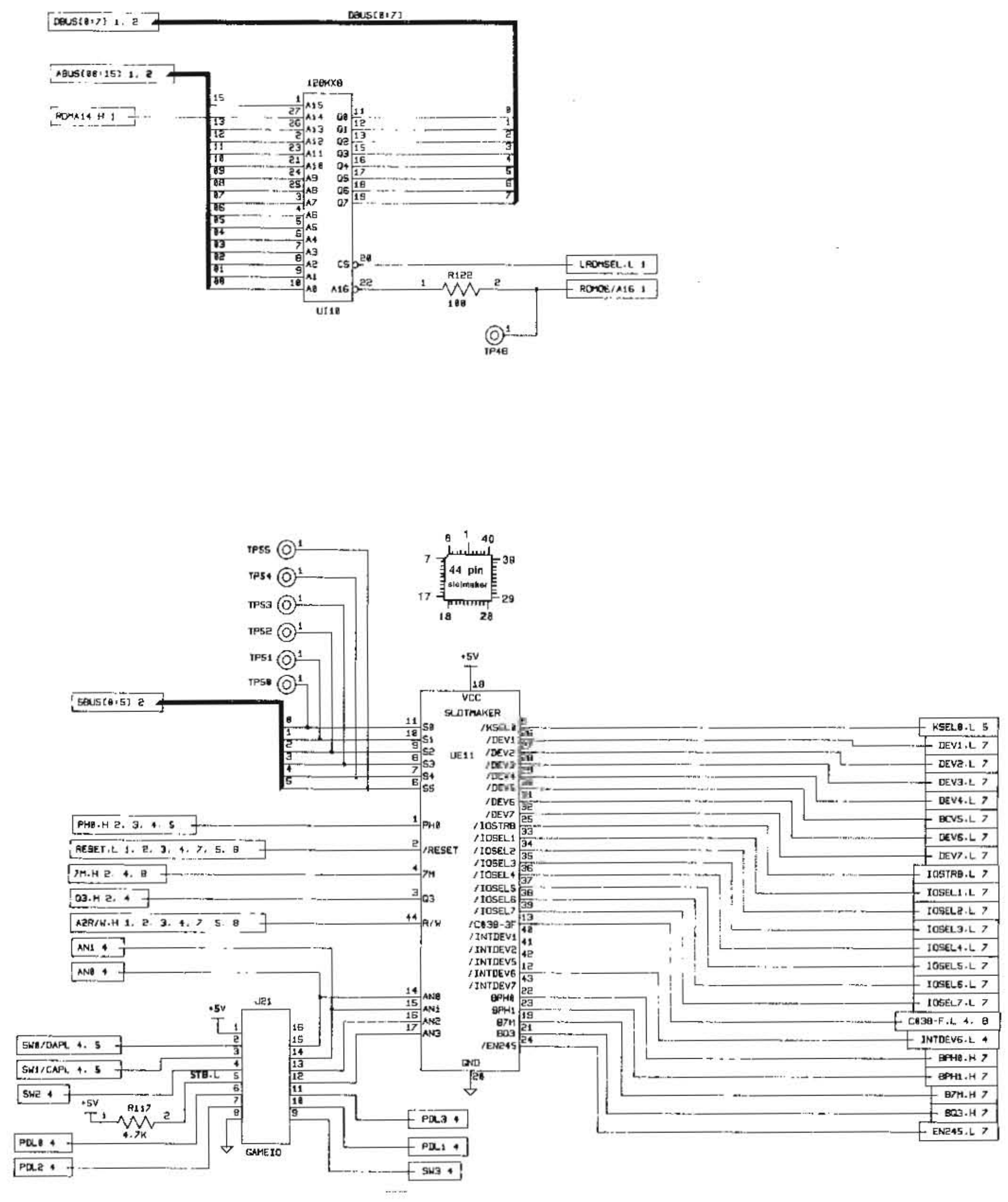
TITLE: APPLE IIGS
SHEET 3 OF 8
22-SEP-86



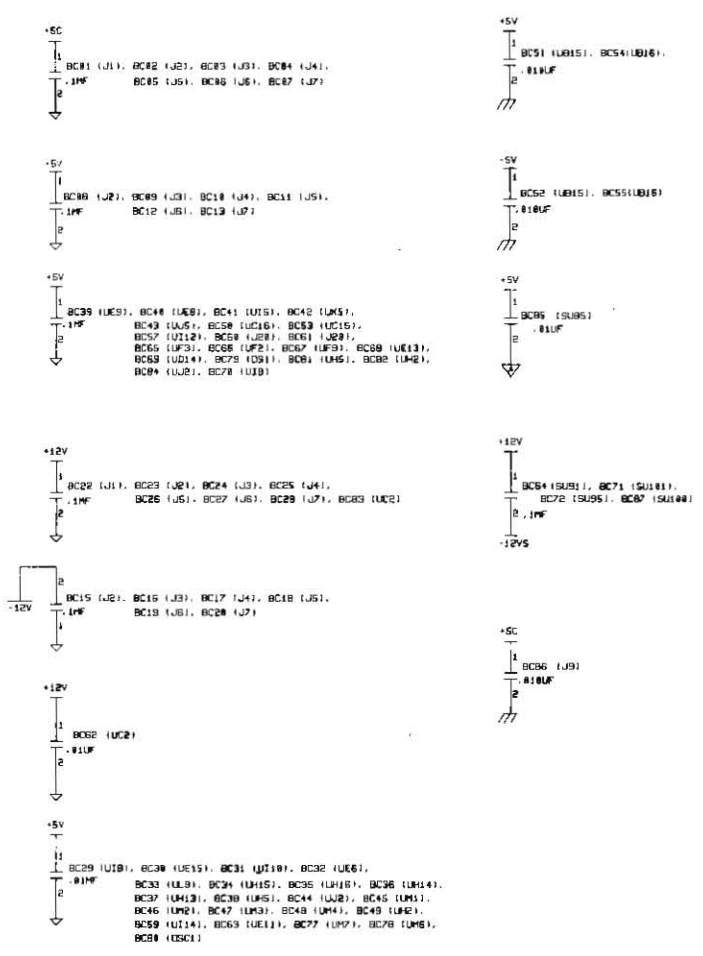
TITLE: APPLE IIGS
SHEET 4 OF 8
22-SEP-86



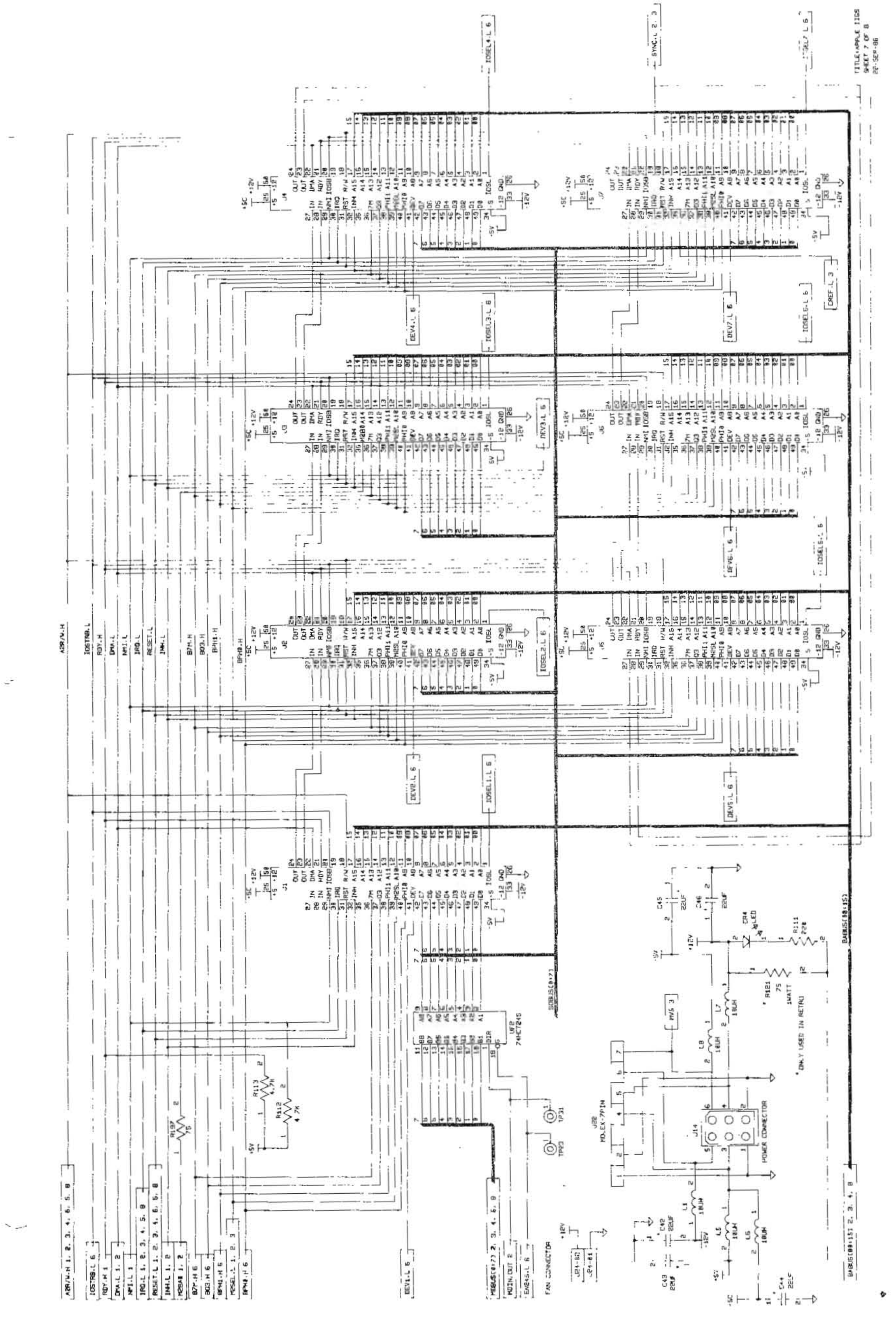
TITLE: APPLE IIGS
SHEET 5 OF 8
22-SEP-86

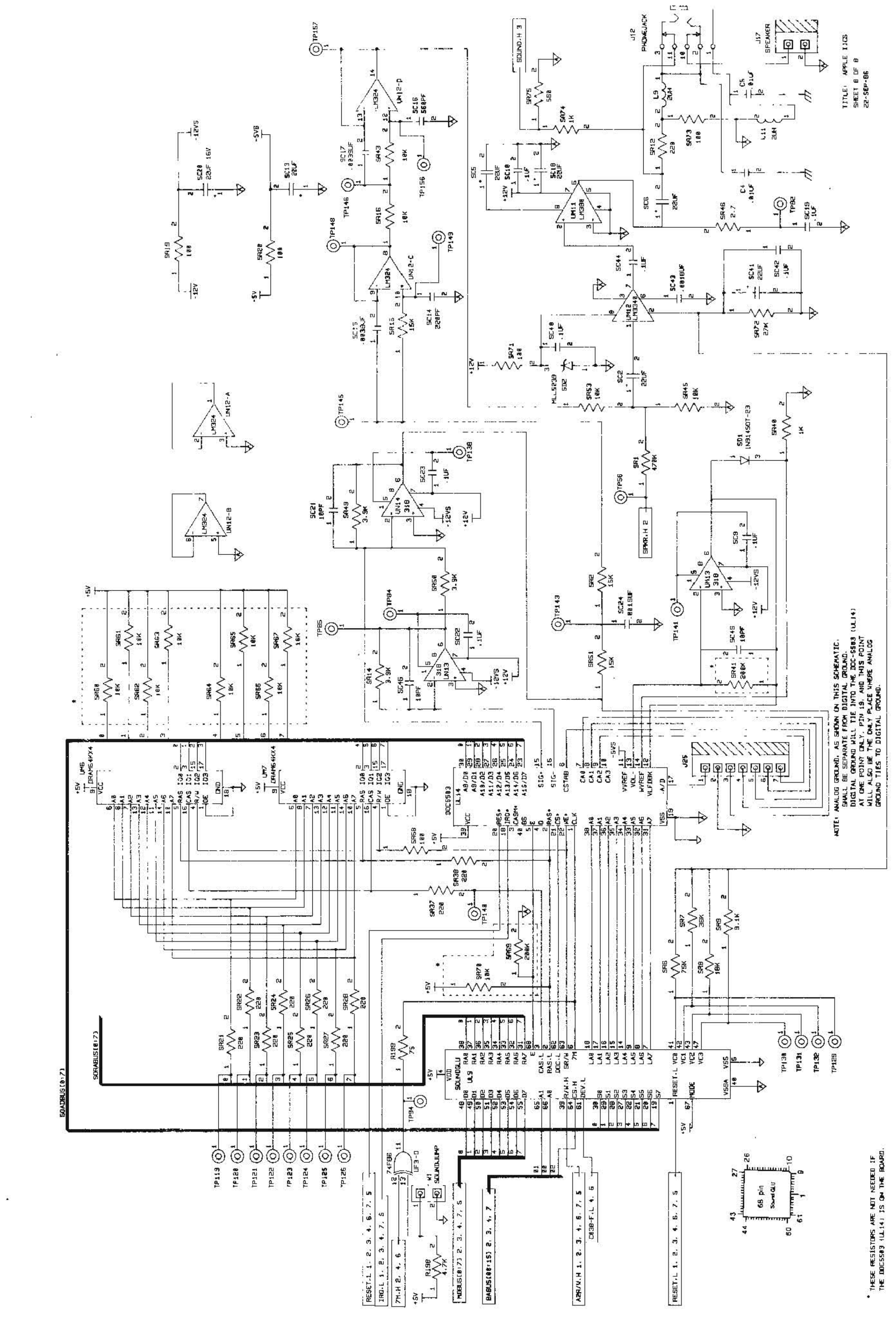


BYPASS CAPACITORS


+5C
BC01 (J1), BC02 (J2), BC03 (J3), BC04 (J4),
BC05 (J5), BC06 (J6), BC07 (J7)
.1MF
+5V
BC08 (J2), BC09 (J3), BC10 (J4), BC11 (J5),
BC12 (J6), BC13 (J7)
.1MF
+5V
BC39 (UE9), BC40 (UE8), BC41 (UI5), BC42 (UK5),
BC43 (UJ5), BC50 (UC16), BC53 (UC15),
BC57 (UI12), BC60 (J20), BC61 (J20),
BC65 (UF3), BC66 (UF2), BC67 (UF9), BC68 (UE13),
BC69 (UD14), BC79 (DS1), BC81 (UH5), BC82 (UH2),
BC84 (UJ2), BC70 (UI8)
.1MF
+12V
BC22 (J1), BC23 (J2), BC24 (J3), BC25 (J4),
BC26 (J5), BC27 (J6), BC28 (J7), BC83 (UC2)
.1MF
-12V
BC15 (J2), BC16 (J3), BC17 (J4), BC18 (J5),
BC19 (J6), BC20 (J7)
.1MF
+12V
BC62 (UC2)
.01UF
+5V
BC29 (UI8), BC30 (UE15), BC31 (UI10), BC32 (UE6),
BC33 (UL9), BC34 (UH15), BC35 (UH16), BC36 (UH14),
BC37 (UH13), BC38 (UH5), BC44 (UJ2), BC45 (UM1),
BC46 (UM2), BC47 (UM3), BC48 (UM4), BC49 (UH2),
BC59 (UI14), BC63 (UE11), BC77 (UM7), BC78 (UM6),
BC80 (OSC1)
.01MF
+5V
BC51 (UB15), BC54(UB16),
.010UF
-5V
BC52 (UB15), BC55(UB16)
.010UF
+5V
BC85 (SU85)
.01UF
+12V
BC64 (SU91), BC71 (SU101),
BC72 (SU95), BC87 (SU100)
.1MF
-12VS
+5C
BC86 (J9)
.010UF

TITLE: APPLE IIGS
SHEET 6 OF 8
22-SEP-86


TITLE: APPLE IIGS
SHEET 7 OF 8
22-SEP-86
J1
J2
J3
J4
J5
J6
J7
UF2 74HCT245
J14 POWER CONNECTOR
J32 MOLEX-7PIN
FAN CONNECTOR
TP31
TP23
R112 4.7K
R113 4.7K
R121 75 1WATT
R111 220
CR4 LED
C45 22UF
C46 22UF
C42 22UF
C43 22UF
C44 22UF
L1 10UH
L5 10UH
L6 10UH
L7 10UH
L8 10UH
* ONLY USED IN RETAIL
A20/V.H 1, 2, 3, 4, 6, 5, 8
IOSTRB.L 6
RDY.H 1
DMA.L 1, 2
NMI.L 1
IRQ.L 1, 3, 4, 5, 8
RESET.L 1, 2, 3, 4, 6, 5, 8
INH.L 1, 2
M2B0 1, 2
B7M.H 6
BQ3.H 6
BPH1.H 6
M2SEL.L 1, 2, 3
BPH0.H 6
DEV1.L 6
DEV2.L 6
DEV3.L 6
DEV4.L 6
DEV5.L 6
DEV6.L 6
DEV7.L 6
IOSEL1.L 6
IOSEL2.L 6
IOSEL3.L 6
IOSEL4.L 6
IOSEL5.L 6
IOSEL6.L 6
IOSEL7.L 6
SYNC.L 2, 3
CREF.L 3
MDBUS(0:7) 2, 3, 4, 5, 8
MDIN.OUT 2
EN245.L 6
BABUS(00:15) 2, 3, 4, 8
SDBUS(0:7)
MVS 3



TITLE: APPLE IIGS
SHEET 8 OF 8
22-SEP-86

NOTE: ANALOG GROUND, AS SHOWN ON THIS SCHEMATIC, SHALL BE SEPARATE FROM DIGITAL GROUND. DIGITAL GROUND WILL TIE INTO THE DOC-5503 (UL14) AT ONE POINT ONLY, PIN 19. AND THIS POINT WILL ALSO BE THE ONLY PLACE WHERE ANALOG GROUND TIES TO DIGITAL GROUND.

* THESE RESISTORS ARE NOT NEEDED IF THE DOC5503 (UL14) IS ON THE BOARD.

# Appendix E

# 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.

## E.1 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 E-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 comprise 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 and A through F.
- Eight bits (two nibbles) make a byte (figure E-1).
- One byte can represent any of 16 x 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 E-5 through E-8 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 x 256, or 65536, different values.
- Three bytes make an address. The 24 bits of an address can represent any one of 256 x 65536, or 16,777,216, different values.
- The 65C816 uses a 24-bit adress to identify a memory location. It can therefore distinguish among 16,777,216 (16M) locations at any given time.
- A memory location is one 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 E-1**
What a bit can represent

| Context | Representing | 0 = | 1 = |
|---|---|---|---|
| Binary number | Place value | 0 | 1 x that power of 2 |
| Logic | Condition | False | True |
| Any switch | Position | Off | On |
| Any switch | Position | Clear* | Set |
| Serial transfer | Beginning | Start | Carrier (no information yet) |
| Serial transfer | Data | 0 value | 1 value |
| Serial transfer | Parity | SPACE | MARK |
| Serial transfer | End | | Stop bit(s) |
| Serial transfer | Communication state | BREAK | Carrier |
| P reg. bit N | Neg. result? | No | Yes |
| P reg. bit V | Overflow? | No | Yes |
| P reg. bit B | BRK command? | No | Yes |
| P reg. bit D | Decimal mode? | No | Yes |
| P reg. bit I | IRQ interrupts | Enabled | Disabled (masked out) |
| P reg. bit Z | Zero result? | No | Yes |
| P reg. bit C | Carry required? | No | Yes |

* Sometimes ambiguously termed *reset*.

**Figure E-1**
Bits, nibbles, and bytes

| 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 |

## E.2 Hexadecimal and decimal

Use Table E-2 for conversion of hexadecimal and decimal numbers.

**Table E-2**
Hexadecimal/Decimal conversion

| Digit | $x000 | $0x00 | $00x0 | $000x |
|---|---|---|---|---|
| F | 61440 | 3840 | 240 | 15 |
| E | 57344 | 3584 | 224 | 14 |
| D | 53248 | 3328 | 208 | 13 |
| C | 49152 | 3072 | 192 | 12 |
| B | 45056 | 2816 | 176 | 11 |
| A | 40960 | 2560 | 160 | 10 |
| 9 | 36864 | 2304 | 144 | 9 |
| 8 | 32768 | 2048 | 128 | 8 |
| 7 | 28672 | 1792 | 112 | 7 |
| 6 | 24576 | 1536 | 96 | 6 |
| 5 | 20480 | 1280 | 80 | 5 |
| 4 | 16384 | 1024 | 64 | 4 |
| 3 | 12288 | 768 | 48 | 3 |
| 2 | 8192 | 512 | 32 | 2 |
| 1 | 4096 | 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.

**Examples:**

```
$3C = ?                  $FD47 = ?

$30 = 48                 $F000 = 61440
$0C = 12                 $ D00 =  3328
________                 $  40 =    64
                         $   7 =     7
$3C = 60                 _____________

                         $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.

**Example:**

```
16215 = $ ?

16215 - 12288 = 3927     12288 = $7000    3927 -
  3840 = 87          3840 = $ F00      87 -
    80 = 7             80 = $  50
          7                  7 = $    7
                         ________________

                         16215  =  $7F57
```

## E.3 Hexadecimal and negative decimal

If a number is larger than decimal 32767, Applesoft BASIC allows you to use the negative-decimal equivalent of the number. Table E-3 is set up to make it easy for you to convert a hexadecimal number directly to a negative-decimal number.

**Table E-3**
Hexadecimal to negative decimal conversion

| Digit | $x000 | $$0x00 | $$00x0 | $$000x |
|---|---|---|---|---|
| | 0 | 0 | 0 | -1 |
| E | -4096 | -256 | -16 | -2 |
| D | -8192 | -512 | -32 | -3 |
| C | -12288 | -768 | -48 | -4 |
| B | -16384 | -1024 | -64 | -5 |
| A | -20480 | -1280 | -80 | -6 |
| 9 | -24576 | -1536 | -96 | -7 |
| 8 | -28672 | -1792 | -112 | -8 |

| | | | |
|---|---|---|---|
| 7 | -2048 | -128 | -9 |
| 6 | -2304 | -144 | -10 |
| 5 | -2560 | -160 | -11 |
| 4 | -2816 | -176 | -12 |
| 3 | -3072 | -192 | -13 |
| 2 | -3328 | -208 | -14 |
| 1 | -3584 | -224 | -15 |
| 0 | -3840 | -240 | -16 |

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (0s 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.

**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 65536. (This addition ends up looking like subtraction.)

**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 E-2.

## E.4 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.

Table E-4 is a definition of the PIN number digits. When communication mode is selected, the seventh digit is ignored.

**Example:** 252/1111 means:

| | |
|---|---|
| Communication mode | No parity |
| 8 data bits, 1 stop bit | Do not echo output to display |
| 300 baud (bits per second) | No line feed after carriage return<br>Do not generate carriage returns |

**Table E-4**
PIN numbers

x x x / x x x x

1 = Printer mode
2 = Communication mode*

1 = 6 data bits, 1 stop bit
2 = 6 data bits, 2 stop bits
3 = 7 data bits, 1 stop bit
4 = 7 data bits, 2 stop bits
5 = 8 data bits, 1 stop bit
6 = 8 data bits, 2 stop bits

1 = 110 bits per second
2 = 300 bits per second
3 = 1200 bits per second
4 = 2400 bits per second
5 = 4800 bits per second
6 = 9600 bits per second
7 = 19200 bits per second

- - - - - - - - - - - - - - - - - - - - -

1 = No parity
2 = Even parity (total on = even)
3 = Odd parity (total on = odd)
4 = MARK parity (parity bit = 1)
5 = SPACE parity (parity bit = 0)

1 = Do not echo output on screen
2 = Echo output on screen

1 = Do not generate LF after CR
2 = Generate LF after CR

1 = Do not generate CR*

2 = Generate CR after 40 characters
3 = Generate CR after 72 characters
4 = Generate CR after 80 characters
5 = Generate CR after 132 characters

* If you select communication mode, then seventh digit must equal 1. This value is supplied automatically when you use the UUD.

## E.5 Eight-Bit code conversions

Tables E-5 through E-8 show the entire ASCII character set. Note that character values are shown with the high bit off. Unless otherwise noted, all ASCII character values above $7F (127 decimal) generate the same character as that value with the high bit off. Here is how to interpret these tables:

- The *Binary* column has the 8-bit code for each ASCII character.
- The first 128 ASCII entries represent 7-bit ASCII codes plus a high-order bit of 0 (SPACE parity or Pascal)—for example, 01001000 for the letter *E*.
- The last 128 ASCII entries (from 128 through 255) represent 7-bit ASCII codes plus a high-order bit of 1 (MARK parity or BASIC)—for example, 11001000 for the letter *E*.
- A transmitted or received ASCII character will take whichever form (in the communication register) is appropriate if odd or even parity is selected—for example, 11001000 for an odd-parity *H*, 01001000 for an even-parity *E*.
- The *ASCII Char* 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 *Pri* and *Alt* 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 5-1) if the firmware is set to do so (Section 5.2.2), or if the firmware is bypassed.

❖ *Note:* The primary and alternate displayed character sets in Tables E-5 through E-8 are the result of firmware mapping. The character generator ROM actually contains only one character set. The firmware mapping procedure is described in Section 3.3.6.

**Table E-5**
Control characters, high bit off

| Binary | Dec | Hex | ASCII Char | Interpretation | What to Type | Pri | Alt |
|---|---|---|---|---|---|---|---|
| 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 E-6**
Special characters, high bit off

| Binary | Dec | Hex | ASCII Char | Interpretation | What to Type | Pri | Alt |
|---|---|---|---|---|---|---|---|
| 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 E-7**
Uppercase characters, high bit off

| Binary | Dec | Hex | ASCII Char | Interpretation | What to Type | Pri | Alt* |
|---|---|---|---|---|---|---|---|
| 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 | \ | Reverse slant | *\* |
| 1011101 | 93 | $5D | ] | Closing bracket | *]* |
| 1011110 | 94 | $5E | ^ | Caret | *^* |
| 1011111 | 95 | $5F | _ | Underline | *_* |

* If the high bit is set, the MouseText characters are replaced with their equivalent in the primary character set with that value.

**Table E-8**
Lowercase characters, high bit off

| Binary | Dec | Hex | ASCII Char | Interpretation | What to Type | Pri | Alt |
|---|---|---|---|---|---|---|---|
| 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 | ? | **DE** |

**L**

# Appendix F

# Frequently Used Tables

This appendix contains frequently-used tables from throughout the manual.

# Glossary

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**bootstrap:** See **boot.**

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

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

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

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

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

**cathode–ray tube:** A display device.

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

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

**chip:** See **integrated circuit.**

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

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

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

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

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

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

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

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

**computer language:** See **programming language.**

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

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

**control 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.

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

**CPU:** See **central processing unit.**

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

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

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

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

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

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

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

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

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

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

**DCD:** See **Data Carrier Detect.**

**DCE:** See **Data Communication Equipment.**

**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.

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

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

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

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

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

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

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

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

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

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

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

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

**DOC:** See **digital oscillator chip.**

**DSR:** See **Data Set Ready.**

**DTE:** See **Data Terminal Equipment.**

**DTR:** See **Data Terminal Ready.**

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

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

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

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

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

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

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

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

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

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

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

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

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

**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.**

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

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

**graph:** A pictorial representation of data.

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

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

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

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

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

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

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

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

**Hz:** See **hertz.**

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

**IC:** See **integrated circuit.**

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

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

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

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

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

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

**GTE**

# G65SC802
# G65SC816

## Microcircuits

## CMOS 8/16-Bit Microprocessor Family

### Features

- Advanced CMOS design for low power consumption and increased noise immunity
- Emulation mode for total software compatibility with 6502 designs
- Full 16-bit ALU, Accumulator, Stack Pointer, and Index Registers
- Direct Register for "zero page" addressing
- 24 addressing modes (including 13 original 6502 modes)
- Wait for Interrupt (WAI) and Stop the Clock (STP) instructions for reduced power consumption and decreased interrupt latency
- 91 instructions with 255 opcodes
- Co-Processor (COP) instruction and associated vector
- Powerful Block Move instructions

### Features (G65SC802 Only)

- 8-Bit Mode with both software and hardware (pin-to-pin) compatibility with 6502 designs (64 KByte memory space)
- Program selectable 16-bit operation
- Choice of external or on-board clock generation

### Features (G65SC816 Only)

- Full 16-bit operation with 24 address lines for 16 MByte memory
- Program selectable 8-Bit Mode for 6502 coding compatibility.
- Valid Program Address (VPA) and Valid Data Address (VDA) outputs for dual cache and DMA cycle steal implementation
- Vector Pull ($\overline{\text{VP}}$) output indicates when interrupt vectors are being fetched. May be used for vectoring/prioritizing interrupts
- Abort interrupt and associated vector for interrupting any instruction without modifying internal registers
- Memory Lock ($\overline{\text{ML}}$) for multiprocessor system implementation

### General Description

The G65SC802 and G65SC816 are ADV-CMOS (ADVanced CMOS) 16-bit microprocessors featuring total software compatibility with 8-bit NMOS and CMOS 6500 series microprocessors. The G65SC802 is pin-to-pin compatible with 8-bit 6502 devices currently available, while also providing full 16-bit internal operation. The G65SC816 provides 24 address lines for 16 MByte addressing, while providing both 8-bit and 16-bit operation.

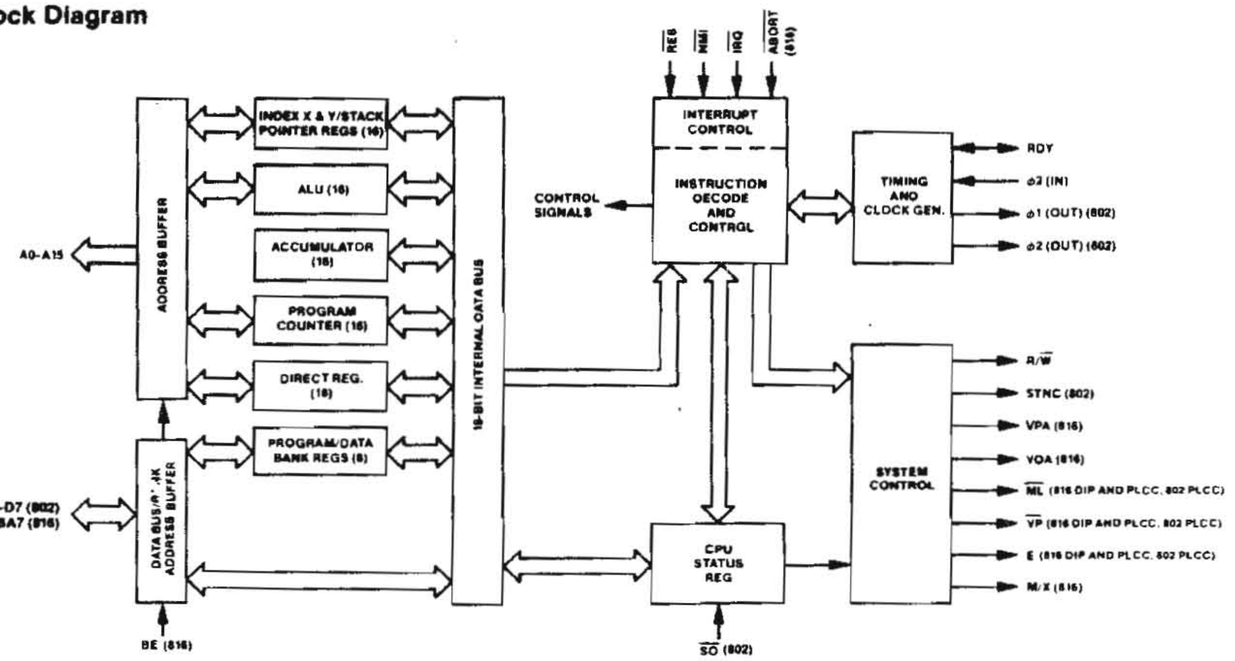
Each microprocessor contains an Emulation (E) mode for emulating 8-bit NMOS and CMOS 6500-Series microprocessors. A software switch determines whether the processor is in the 8-bit emulation mode or in the Native 16-bit mode. This allows existing 8-bit system designs to use the many powerful features of the G65SC802 and G65SC816.

The G65SC802 and G65SC816 provide the system engineer with many powerful features and options. A 16-bit Direct Page Register is provided to augment the Direct Page addressing mode, and there are separate Program Bank Registers for 24-bit memory addressing. Other valuable features include:

- An Abort input which can interrupt the current instruction without modifying internal registers.
- Valid Data Address (VDA) and Valid Program Address (VPA) outputs which facilitate dual cache memory by indicating whether a data or program segment is being accessed.
- Vector modification by simply monitoring the Vector Pull ($\overline{\text{VP}}$) output.
- Block Move instructions.

GTE Microcircuits' G65SC802 and G65SC816 microprocessors offer the design engineer a new freedom of design and application, and the many advantages of state-of-the-art ADV-CMOS technology.

### Simplified Block Diagram



**ADVANCE INFORMATION**

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 | $V_{IH}$ | | | |
| $\overline{\text{RES}}$, RDY, $\overline{\text{IRQ}}$, Data, $\overline{\text{SO}}$, BE | | 2.0 | $V_{DD}$ + 0.3 | V |
| $\overline{\text{ABORT}}$, $\overline{\text{NMI}}$, ϕ2 (IN) | | 0.7 $V_{DD}$ | $V_{DD}$ + 0.3 | V |
| Input Low Voltage | $V_{IL}$ | | | |
| $\overline{\text{RES}}$, RDY, $\overline{\text{IRQ}}$, Data, $\overline{\text{SO}}$, BE | | -0.3 | 0.8 | V |
| $\overline{\text{ABORT}}$, $\overline{\text{NMI}}$, ϕ2 (IN) | | -0.3 | 0.2 | V |
| Input Leakage Current ($V_{IN}$ = 0 to $V_{DD}$) | $I_{IN}$ | | | |
| $\overline{\text{RES}}$, $\overline{\text{NMI}}$, $\overline{\text{IRQ}}$, $\overline{\text{SO}}$, BE, $\overline{\text{ABORT}}$ (Internal Pullup) | | -100 | 1 | μA |
| RDY (Internal Pullup, Open Drain) | | -100 | 10 | μA |
| ϕ2 (IN) | | -1 | 1 | μA |
| Address, Data, R/$\overline{\text{W}}$ (Off State, BE = 0) | | -10 | 10 | μA |
| Output High Voltage ($I_{OH}$ = -100μA) SYNC, Data, Address, R/$\overline{\text{W}}$, $\overline{\text{ML}}$, $\overline{\text{VP}}$, M/X, E, VDA, VPA, ϕ1 (OUT), ϕ2 (OUT) | $V_{OH}$ | 0.7 $V_{DD}$ | — | V |
| Output Low Voltage ($I_{OL}$ = 1.6mA) SYNC, Data, Address, R/$\overline{\text{W}}$, $\overline{\text{ML}}$, $\overline{\text{VP}}$, M/X, E, VDA, VPA, ϕ1 (OUT), ϕ2 (OUT) | $V_{OL}$ | — | 0.4 | V |
| Supply Current (No Load) f = 2 MHz | $I_{DD}$ | — | 10 | mA |
| f = 4 MHz | | — | 20 | mA |
| f = 6 MHz | | — | 30 | mA |
| f = 8 MHz | | — | 40 | mA |
| Standby Current (No Load; Data Bus = $V_{SS}$ or $V_{DD}$; ϕ2(IN) = $\overline{\text{ABORT}}$ = $\overline{\text{RES}}$ = $\overline{\text{NMI}}$ = $\overline{\text{IRQ}}$ = $\overline{\text{SO}}$ = BE = $V_{DD}$) | $I_{SB}$ | — | 10 | μA |
| Capacitance ($V_{IN}$ = 0V, $T_A$ = 25°C, f = 2 MHz) | | | | |
| Logic, ϕ2 (IN) | $C_{IN}$ | — | 10 | pF |
| Address, Data, R/$\overline{\text{W}}$ (Off State) | $C_{TS}$ | — | 15 | pF |

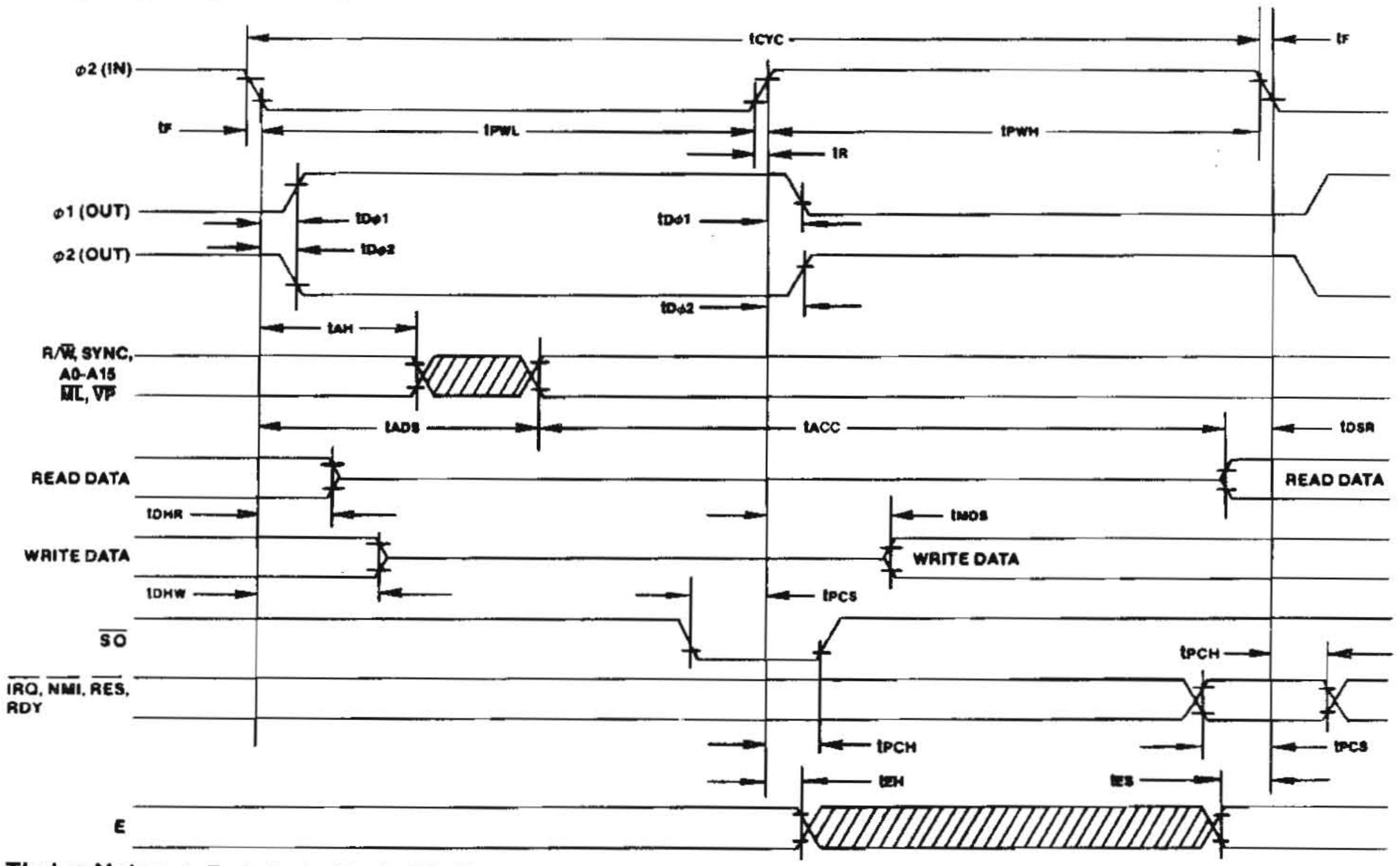
**AC Characteristics (G65SC802):** $V_{DD}$ = 5.0V ±5%, $V_{SS}$ = 0V, $T_A$ = 0°C to +70°C

| | | 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 | 0.080 | 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) | $t_{Dϕ1}$ | — | 40 | — | 40 | — | 40 | — | 40 | nS |
| Delay Time, ϕ2 (IN) to ϕ2 (OUT) | $t_{Dϕ2}$ | — | 40 | — | 40 | — | 40 | — | 40 | nS |
| Address Hold Time | $t_{AH}$ | 10 | — | 10 | — | 10 | — | 10 | — | nS |
| Address Setup Time | $t_{AOS}$ | — | 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 |
| E Output Hold Time | $t_{EH}$ | 10 | — | 10 | — | 5 | — | 5 | — | nS |
| E Output Setup Time | $t_{ES}$ | 50 | — | 50 | — | 25 | — | 15 | — | nS |
| Capacitive Load (Address, Data, and R/W) | $C_{EXT}$ | — | 100 | — | 100 | — | 35 | — | 35 | pF |

**AC Characteristics (G65SC816):** $V_{DD}$ = 5.0V ±5%, $V_{SS}$ = 0V, $T_A$ = 0°C to +70°C

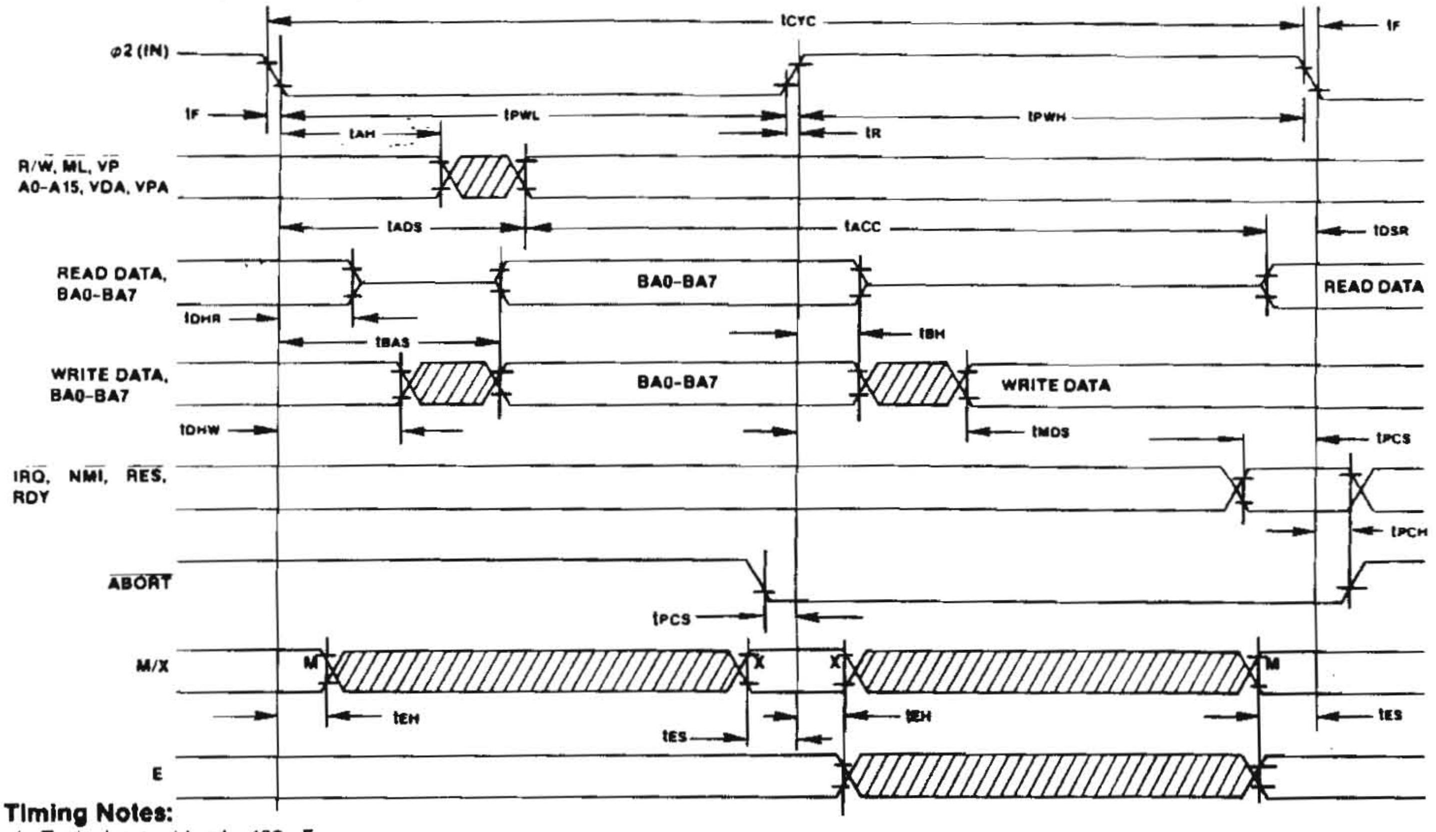
| | | 2 MHz | | 4 MHz | | 6 MHz | | 8 MHz | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Parameter** | **Symbol** | **Min** | **Max** | **Min** | **Max** | **Min** | **Max** | **Min** | **Max** | **Unit** |
| Cycle Time | $t_{CYC}$ | 500 | DC | 250 | DC | 167 | DC | 125 | DC | nS |
| Clock Pulse 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 |
| 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 (G65SC802)



**Timing Notes:**
1. Typical output load = 100 pF
2. Voltage levels are $V_L < 0.4V$, $V_H > 2.4V$
3. Timing measurement points are 0.8V and 2.0V

## Timing Diagram (G65SC816)



## Timing Notes:

1. Typical output load = 100 pF
2. Voltage levels are $V_L < 0.4V$, $V_H > 2.4V$
3. Timing measurement points are 0.8V and 2.0V

## Functional Description

The G65SC802 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 G65SC802's "ease of use" design and implementation features provide the designer with increased flexibility and reduced implementation costs. In the Emulation mode, the G65SC802 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 G65SC802 is an excellent direct replacement microprocessor for 6502 designs.

The G65SC816 provides the design engineer with upward mobility and software compatibility in applications where a 16-bit system configuration is desired. The G65SC816's 16-bit hardware configuration, coupled with current software allows a wide selection of system applications. In the Emulation mode, the G65SC816 offers many advantages, including full software compatibility with 6502 coding. In addition, the G65SC816's powerful instruction set and addressing modes make it an excellent choice for new 16-bit designs.

Internal organization of the G65SC802 and G65SC816 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 G65SC802 and the G65SC816 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 Figure 2, Programming Model)

### Accumulator (A)

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. When the Accumulator Select Bit (M) equals one, the Accumulator is 6 bits wide. In this case, the upper 8 bits (AH) may be used for temporary storage in conjunction with the Exchange AH and AL instruction.

### Data Bank (DB)

During the Native mode (E=0), 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 G65SC816. 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.

## 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 2, G65SC802 and G65SC816 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 (PB)

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.

## 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 01. The Bank Address is 00 for all Stack operations.

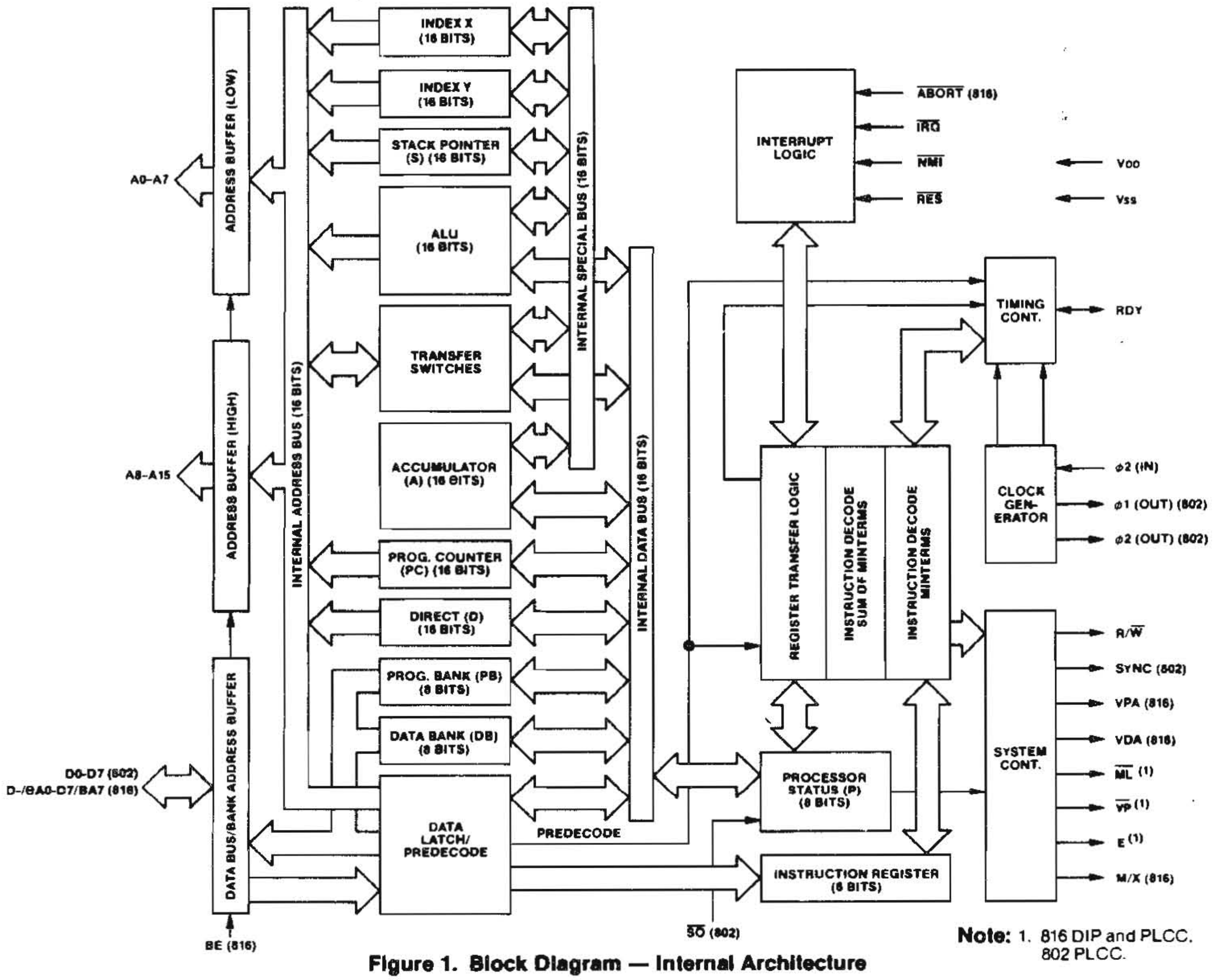


**Figure 1. Block Diagram — Internal Architecture**

**Note:** 1. 816 DIP and PLCC, 802 PLCC.

## Signal Description

The following Signal Description applies to both the G65SC802 and the G65SC816 except as otherwise noted.

### Abort ($\overline{ABORT}$)—G65SC816

The Abort input prevents modification of any internal registers during execution of 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). Abort is asserted whenever there is a low level on the Abort input, and the $\phi 2$ clock is high. The Abort internal latch is cleared during the second cycle of the interrupt sequence. This signal may be used to handle out-of-bounds memory references in virtual memory systems.

### 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 G65SC816, the address lines may be set to the high impedance state by the Bus Enable (BE) signal.

### Bus Enable (BE)

The Bus Enable input signal allows external control of the Address and Data Buffers, as well as the R/$\overline{W}$ signal. With Bus Enable high, the R/$\overline{W}$ 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)—G65SC802

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)—G65SC816

These eight lines multiplex bits BA0-BA7 with the data value. The Bank 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. The Bank address external transparent latch should be latched when the $\phi 2$ clock is high or RDY is low. 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)—G65SC816 (Also Applies to G65SC802, 44-Pin Version)

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 ($\overline{ML}$)—G65SC816 (Also Applies to G65SC802, 44-Pin Version)

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)—G65SC816

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 $\phi 2$ clock positive transition. Instructions PLP, REP, RTI and SEP may change the state of these bits. Note that the M/X output may be invalid in the cycle following a change in the M or X bits. These bits may be thought of as opcode extensions and may be used for memory and system management.

### Non-Maskable Interrupt ($\overline{NMI}$)

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 $\overline{NMI}$ 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 $\overline{NMI}$ remains low.

### Phase 1 Out ($\phi 1$ (OUT))—G65SC802

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 ($\phi 2$ (IN))

This is the system clock input to the microprocessor internal clock generator (equivalent to $\phi 0$ (IN) on the 6502). During the low power Standby Mode, $\phi 2$ (IN) should be held in the high state to preserve the contents of internal registers.

### Phase 2 Out ($\phi 2$ (OUT))—G65SC802

This clock output signal provides timing for external read and write operations. Addresses are valid (after the Address Setup Time ($T_{AOS}$)) 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 G65SC816, 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 G65SC802 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 mey be used with the internal pullup device. The $\overline{RES}$ signal must be held low for at least two clock cycles after $V_{DD}$ reaches operating voltage. Ready (RDY) has no effect while $\overline{RES}$ is being held low. During this Reset conditioning period, the following processor initialization takes plece:

**Registers**

| | | | | | |
|---|---|---|---|---|---|
| D | = | 0000 | SH | = | 01 |
| DB | = | 00 | XH | = | 00 |
| PB | = | 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 ($\overline{SO}$)—G65SC802**

A negative transition on this input sets the Overflow (V) flag, bit 6 of the Processor Status (P) Register.

**Synchronize (SYNC)—G65SC802**

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)—G65SC816**

These two output signals indicate the type of memory being accessed by the address bus. The following coding applies:

| VDA | VPA | |
|---|---|---|
| 0 | 0 | Internal Operation—Address and Data Bus available. Address outputs may be invalid due to low byte additions only. |
| 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 ground. When using only one ground on the G65SC802 DIP package, pin 21 is preferred.

**Vector Pull (VP)—G65SC816 (Also Applies to G65SC802, 44-Pin Version)**

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.

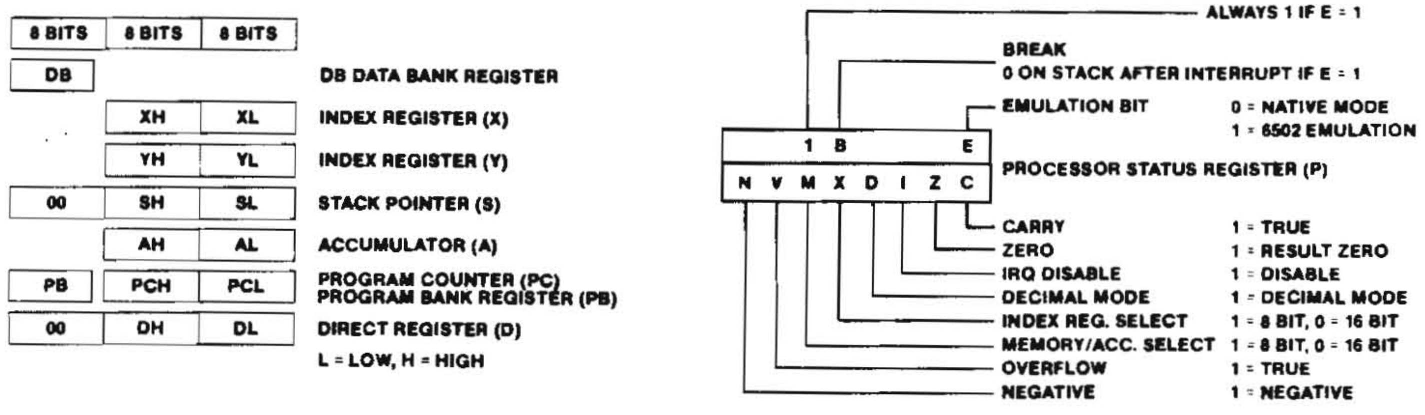


**Figure 2. Programming Model**

**Table 1. G65SC802 and G65SC816 Compatibility**

| Function | G65SC802/818 Emulation | G65SC02 | NMOS 6502 |
|---|---|---|---|
| Decimal Mode:<br>• After Interrupts<br>• N, Z Flags<br>• ADC, SBC | <br>0 → D<br>Valid<br>No added cycle | <br>0 → D<br>Valid<br>Add 1 cycle | <br>Not initialized<br>Undefined<br>No added cycle |
| Read-Modify-Write:<br>• Absolute Indexed, No Page Crossing<br>• Write<br>• Memory Lock | <br>7 cycles<br>Last 2 cycles<br>Last 3 cycles | <br>6 cycles<br>Last cycle<br>Last 2 cycles | <br>7 cycles<br>Last 2 cycles<br>Not available |
| Jump Indirect:<br>• Cycles<br>• Jump Address, Operand = XXFF | <br>5 cycles<br>Correct | <br>6 cycles<br>Correct | <br>5 cycles<br>Invalid |
| Branch or Index Across Page Boundary | Read last program byte | Read last program byte | Read invalid address |
| 0 → RDY During Write | G65SC802: Ignored until read<br>G65SC816: Processor stops | Processor stops | Ignored until read |
| Write During Reset | No | Yes | No |
| Unused Opcodes | No operation | No operation | Undefined |
| $\phi$1 (OUT), $\phi$2 (OUT), $\overline{SO}$, SYNC Signals | Available with G65SC802 only | Available | Available |
| RDY Signal | Bidirectional | Input | Input |

**Table 2. G65SC802 and G65SC816 Mode Comparison**

| Function | Emulation (E = 1) | Native (E = 0) |
|---|---|---|
| Stack Pointer (S) | 8 bits in page 1 | 16 bits |
| Direct Index Address | Wrap within page | Crosses page boundary |
| Processor Status (P): | | |
| • Bit 4 | Always one, except zero in stack after hardware interrupt | X flag (8/16-bit Index) |
| • Bit 5 | Always one | M flag (8/16-bit Accumulator) |
| Branch Across Page Boundary | 4 cycles | 3 cycles |
| Vector Locations: | | |
| ABORT | 00FFF8,9 | 00FFE8,9 |
| BRK | 00FFFE,F | 00FFE6,7 |
| COP | 00FFF4,5 | 00FFE4,5 |
| IRQ | 00FFFE,F | 00FFEE,F |
| NMI | 00FFFA,B | 00FFEA,B |
| RES | 00FFFC,D | 00FFFC,D (1 → E) |
| Program Bank (PB) During Interrupt, RTI | Not pushed, pulled | Pushed and pulled |
| 0 → RDY During Write | G65SC802: Ignored until read<br>G65SC816: Processor stops | Processor stops |
| Write During Read-Modify-Write | Last 2 cycles | Last 1 or 2 cycles depending on M flag |

## G65SC802 and G65SC816 Microprocessor Addressing Modes

The G65SC816 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 Native mode Stack address will always be within the range 000000 to 00FFFF. In the Emulation mode, the Stack address range is 000100 to 0001FF. The following opcodes and addressing modes can 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**
The Direct addressing modes are often used to access memory registers and pointers. The contents of the Direct Register (D) is added to the offset contained in the instruction operand to produce an address in the range 000000 to 00FFFF. Note that in the Emulation mode, [Direct] and [Direct],y addressing modes and the PEI instruction will increment from 0000FE or 0000FF into the Stack area, even if D=0.

**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. As a result, indexing from page FF in the G65SC802 may result in data accessed in page zero. 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 Indexed Long [d],y
- Absolute
- Absolute,x
- Absolute,y
- Absolute long
- Absolute long indexed
- 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 G65SC802 and G65SC816 microprocessors. The "long" addressing modes may be used with the G65SC802; 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:** | DB | 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.

## 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.

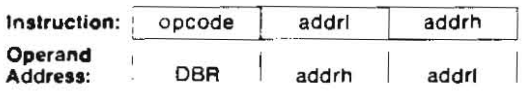
Twenty-four addressing modes are available for use with the W65C802 and W65C816 microprocessors. The "long" addressing modes may be used with the W65C802; however, the high byte of the address is not available to the hardware. Detailed descriptions of the 24 addressing modes are as follows:

### 1. Immediate Addressing—#

The operand is the second byte (second and third bytes when in the 16-bit mode) of the instruction.

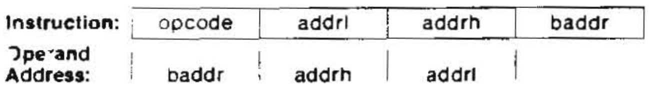
### 2. Absolute—a

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.



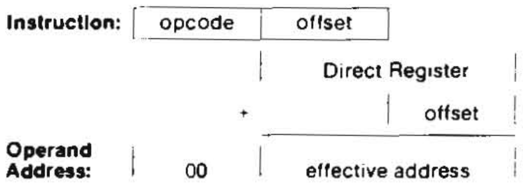
### 3. Absolute Long—al

The second, third, and fourth byte of the instruction form the 24-bit effective address



### 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.



### 5. Accumulator—A

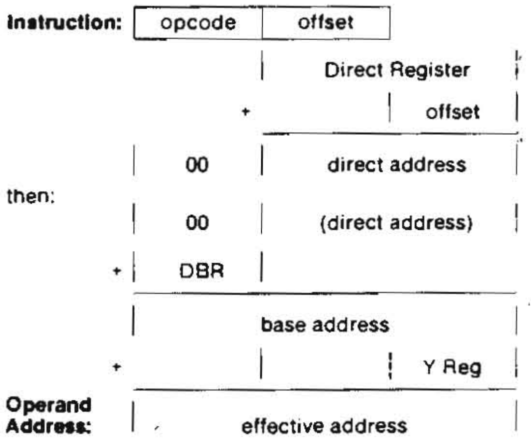
This form of addressing always uses a single byte instruction. The operand is the Accumulator.

### 6. Implied—i

Implied addressing uses a single byte instruction. The operand is implicitly defined by the instruction.

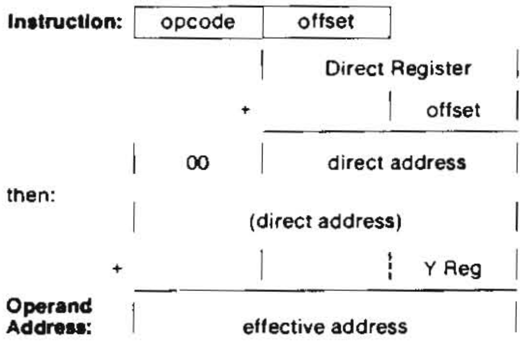
### 7. Direct Indirect Indexed—(d),y

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.



### 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.



### 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.

**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 and result are both an unsigned 16-bit quantity in the range 0 to 65535.

**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.

```
Instruction: | opcode | addrl  | addrh  |

  Indirect Address =   |   00   | addrh  | addrl  |
  New PC = (indirect address)
with JML:
  New PC = (indirect address)
  New PB = (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.

```
Instruction: | opcode | offset |
                      |   Direct Register   |
                    +          |   offset   |
                      ----------------------
             |   00   |   direct address    |
then:
             |   00   |  (direct address)   |
           + |   DB   |
             -------------------------------
Operand
Address:     |        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.

```
Instruction: | opcode | offset |
                      |   Direct Register   |
                    +          |   offset   |
                      ----------------------
             |   00   |   direct address    |
then:
Operand
Address:     |       (direct 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.

```
Instruction: | opcode | addrl  | addrh  |
                      | addrh  | addrl  |
                      |        | X Reg  |
                      -------------------
             |   00   |     address     |
then:
        PC = (address)
```

**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.

```
Instruction: | opcode | offset |
                      |    Stack Pointer    |
                    +          |   offset   |
                      ----------------------
Operand
Address:     |   00   |  effective address  |
```

**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.

```
Instruction: | opcode | offset |
                      |    Stack Pointer    |
                    +          |   offset   |
                      ----------------------
             |   00   |     S + offset      |
then:
                      |     S + offset      |
           + |   DB   |
             -------------------------------
             |         base address         |
           +          |        |   Y Reg    |
             -------------------------------
Operand
Address:     |       effective address      |
```

**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 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.

```
Instruction: | opcode | dstbnk | srcbnk |
                        dstbnk  →  DB
Source
Address:              | srcbnk |      X Reg      |
Destination
Address:              |   DB   |      Y Reg      |
```

Increment (MVN) or decrement (MVP) X and Y.
Decrement A, (if greater than zero), then PC-3 → PC.

## Notes on G65SC802/816 Instructions

### 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 G65SC816 24-bit addressing and are therefore less useful for the G65SC802. The following is a list of instructions and addressing modes which are primarily intended for G65SC816 use:

JSL; RTL; [d]; [d],y; JMP al; JML; al; al,x

The following instructions may be used with the G65SC802 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 only move date in page zero since X and Y Index Register high byte is zero.

### 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 AL and AH are not affected by a mode change.

### 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 $\overline{\text{ABORT}}$ which occurs during the WAI instruction, the processor will return to the WAI instruction. Other than $\overline{\text{RES}}$ (highest priority), $\overline{\text{ABORT}}$ is the next highest priority, followed by $\overline{\text{NMI}}$ or $\overline{\text{IRQ}}$ 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 $\overline{\text{RES}}$ 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 $\overline{\text{RES}}$ input. Note that the external oscillator must be stable and operating properly before $\overline{\text{RES}}$ goes high.

### Tranfers 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 Accumulator, regardless of the state of the M bit in the Status Register.

## Interrupt Processing Sequence

The interrupt processing sequence is initiated as the direct result of hardware Abort, Interrupt Request, Non-Maskable Interrupt, or Reset inputs. The interrupt sequence can also be initiated as a result of the Break or Co-Processor instructions within the software. The following listings describe the function of each cycle in the interrupt processing sequence:

### Hardware Interrupt—$\overline{\text{ABORT}}$, $\overline{\text{IRQ}}$, $\overline{\text{NMI}}$, $\overline{\text{RES}}$ Inputs

| Cycle No. | | Address | Data | R/$\overline{\text{W}}$ | SYNC | VDA | VPA | $\overline{\text{VP}}$ | Description |
|---|---|---|---|---|---|---|---|---|---|
| E = 0 | E = 1 | | | | | | | | |
| 1 | 1 | PC | X | 1 | 1 | 1 | 1 | 1 | Internal Operation |
| 2 | 2 | PC | X | 1 | 0 | 0 | 0 | 1 | Internal Operation |
| 3 | [1] | S | PB | 0 | 0 | 1 | 0 | 1 | Write PB to Stack, S-1 → S |
| 4 | 3 | S | PCH [2] | 0 [3] | 0 | 1 | 0 | 1 | Write PCH to Stack, S-1 → S |
| 5 | 4 | S | PCL [2] | 0 [3] | 0 | 1 | 0 | 1 | Write PCL to Stack, S-1 → S |
| 6 | 5 | S | P [4] | 0 [3] | 0 | 1 | 0 | 1 | Write P to Stack, S-1 → S |
| 7 | 6 | VL | (VL) | 1 | 0 | 1 | 0 | 0 | Read Vector Low Byte, 0 → $P_D$, 1 → $P_I$, 00 → PB |
| 8 | 7 | VH | (VH) | 1 | 0 | 1 | 0 | 0 | Read Vector High Byte |

### Software Interrupt—BRK, COP Instructions

| Cycle No. | | Address | Data | R/$\overline{\text{W}}$ | SYNC | VDA | VPA | $\overline{\text{VP}}$ | Description |
|---|---|---|---|---|---|---|---|---|---|
| E = 0 | E = 1 | | | | | | | | |
| 1 | 1 | PC-2 | X | 1 | 1 | 1 | 1 | 1 | Opcode |
| 2 | 2 | PC-1 | X | 1 | 0 | 0 | 1 | 1 | Signature |
| 3 | [1] | S | PB | 0 | 0 | 1 | 0 | 1 | Write PB to Stack, S-1 → S |
| 4 | 3 | S | PCH | 0 | 0 | 1 | 0 | 1 | Write PCH to Stack, S-1 → S |
| 5 | 4 | S | PCL | 0 | 0 | 1 | 0 | 1 | Write PCL to Stack, S-1 → S |
| 6 | 5 | S | P | 0 | 0 | 1 | 0 | 1 | Write P to Stack, S-1 → S |
| 7 | 6 | VL | (VL) | 1 | 0 | 1 | 0 | 0 | Read Vector Low Byte, 0 → $P_D$, 1 → $P_I$, 00 → PB |
| 8 | 7 | VH | (VH) | 1 | 0 | 1 | 0 | 0 | Read Vector High Byte |

Notes:
[1] Delete this cycle in Emulation mode.
[2] Abort writes address of aborted opcode.
[3] R/$\overline{\text{W}}$ remains in the high state during Reset.
[4] In Emulation mode, bit 4 written to stack is changed to 0.

**Table 3. Vector Locations**

| Name | Source | Emulation (E = 1) | Native (E = 0) | Priority Level |
|---|---|---|---|---|
| $\overline{\text{ABORT}}$ | Hardware | 00FFF8,9 | 00FFE8,9 | 2 |
| BRK | Softwara | 00FFFE,F | 00FFE6,7 | N/A |
| COP | Software | 00FFF4,5 | 00FFE4,5 | N/A |
| $\overline{\text{IRQ}}$ | Hardware | 00FFFE,F | 00FFEE,F | 4 |
| $\overline{\text{NMI}}$ | Hardware | 00FFFA,B | 00FFEA,B | 3 |
| $\overline{\text{RES}}$ | Hardware | 00FFFC,D | 00FFFC,D (1 → E) | 1 |

## Table 4. G65SC802 and G65SC816 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 (Pc = 0) |
| BCS* | Branch on Carry Set (Pc = 1) |
| BEQ | Branch if Equal (Pz = 1) |
| BIT | Bit Test |
| BMI | Branch if Result Minus (PN = 1) |
| BNE | Branch if Not Equal (Pz = 0) |
| BPL | Branch if Result Plus (PN = 0) |
| BRA | Branch Always |
| BRK | Force Break |
| BRL | Branch Always Long |
| BVC | Branch on Overflow Clear (Pv = 0) |
| BVS | Branch on Overflow Set (Pv = 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 (add one cycle if DL ≠ 0) |
| 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 Bits |
| 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 Accumulator to Direct Register |
| TCS* | Transfer Accumulator to Stack Pointer Register |
| TDC* | Transfer Direct Register to Accumulator |
| TRB | Test and Reset Bit |
| TSB | Test and Set Bit |
| TSC* | Transfer Stack Pointer Register to 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 |
| XBA* | Exchange AH and AL |
| XCE | Exchange Carry and Emulation Bits |

***Common Mnemonic Aliases**

| **Mnemonic** | **Alias** |
|---|---|
| BCC | BLT |
| BCS | BGE |
| CMP | CPA |
| DEC A | DEA |
| INC A | INA |
| TCD | TAD |
| TCS | TAS |
| TDC | TDA |
| TSC | TSA |
| XBA | SWA |

**JSL should be recognized as equivalent to JSR when it is specified with long absolute addresses.

JML is equivalent to JMP with long addressing forced.

## Table 5. Arithmetic and Logical Instructions

Addressing Mode

| MNE-MONIC | M/X | OPERATION: E=1 or E=0 and M/X=1 | OPERATION: E=0 and M/X=0 | immed | accum | dir | dir,x | dir,y | (dir) | (dir,x) | (dir),y | [dir] | [dir],y | abs | abs,x | abs,y | abs1 | abs1,x | d,s | (d,s),y | STATUS N V M X D I Z C | MNEMONIC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADC | Pm | AL+B+Pc→AL | A+W+Pc→A | 69 |  | 65 | 75 |  | 72 | 61 | 71 | 67 | 77 | 6D | 7D | 79 | 6F | 7F | 63 | 73 | N V . . . . Z C | ADC |
| AND | Pm | AL∧B→AL | A∧W→A | 29 |  | 25 | 35 |  | 32 | 21 | 31 | 27 | 37 | 2D | 3D | 39 | 2F | 3F | 23 | 33 | N . . . . . Z . | AND |
| ASL (2) | Pm | Pc←B←0 | Pc←W←0 |  | 0A | 06 | 16 |  |  |  |  |  |  | 0E | 1E |  |  |  |  |  | N . . . . . Z C | ASL |
| BIT (1) | Pm | AL∧B | A∧W | 89 |  | 24 | 34 |  |  |  |  |  |  | 2C | 3C |  |  |  |  |  | N V . . . . Z . | BIT |
| CMP | Pm | AL-B | A-W | C9 |  | C5 | D5 |  | D2 | C1 | D1 | C7 | D7 | CD | DD | D9 | CF | DF | C3 | D3 | N . . . . . Z C | CMP |
| CPX | Px | XL-B | X-W | E0 |  | E4 |  |  |  |  |  |  |  | EC |  |  |  |  |  |  | N . . . . . Z C | CPX |
| CPY | Px | YL-B | Y-W | C0 |  | C4 |  |  |  |  |  |  |  | CC |  |  |  |  |  |  | N . . . . . Z C | CPY |
| DEC (2) | Pm | B-1→B | W-1→W |  | 3A | C6 | D6 |  |  |  |  |  |  | CE | DE |  |  |  |  |  | N . . . . . Z . | DEC |
| EOR | Pm | AL⊻B→AL | A⊻W→A | 49 |  | 45 | 55 |  | 52 | 41 | 51 | 47 | 57 | 4D | 5D | 59 | 4F | 5F | 43 | 53 | N . . . . . Z | EOR |
| INC (2) | Pm | B+1→B | W+1→W |  | 1A | E6 | F6 |  |  |  |  |  |  | EE | FE |  |  |  |  |  | N . . . . . Z | INC |
| LDA | Pm | B→AL | W→A | A9 |  | A5 | B5 |  | B2 | A1 | B1 | A7 | B7 | AD | BD | B9 | AF | BF | A3 | B3 | N . . . . . Z | LDA |
| LDX | Px | B→XL | W→X | A2 |  | A6 |  | B6 |  |  |  |  |  | AE |  | BE |  |  |  |  | N . . . . . Z | LDX |
| LDY | Px | B→YL | W→Y | A0 |  | A4 | B4 |  |  |  |  |  |  | AC | BC |  |  |  |  |  | N . . . . . Z . | LDY |
| LSR (2) | Pm | 0→B→Pc | 0→W→Pc |  | 4A | 46 | 56 |  |  |  |  |  |  | 4E | 5E |  |  |  |  |  | 0 . . . . . Z C | LSR |
| ORA | Pm | AL∨B→AL | A∨W→A | 09 |  | 05 | 15 |  | 12 | 01 | 11 | 07 | 17 | 0D | 1D | 19 | 0F | 1F | 03 | 13 | N . . . . . Z . | ORA |
| ROL (2) | Pm | Pc←B←Pc | PC←W←Pc |  | 2A | 26 | 36 |  |  |  |  |  |  | 2E | 3E |  |  |  |  |  | N . . . . . Z C | ROL |
| ROR (2) | Pm | PC→B→Pc | Pc→W→Pc |  | 6A | 66 | 76 |  |  |  |  |  |  | 6E | 7E |  |  |  |  |  | N . . . . . Z C | ROR |
| SBC | Pm | AL-B-$\overline{Pc}$→AL | A-W-$\overline{Pc}$→A | E9 |  | E5 | F5 |  | F2 | E1 | F1 | E7 | F7 | ED | FD | F9 | EF | FF | E3 | F3 | N V . . . . Z C | SBC |
| STA (7) | Pm | AL→B | A→W |  |  | 85 | 95 |  | 92 | 81 | 91 | 87 | 97 | 8D | 9D | 99 | 8F | 9F | 83 | 93 | . . . . . . . . | STA |
| STX | Px | XL→B | X→W |  |  | 86 |  | 96 |  |  |  |  |  | 8E |  |  |  |  |  |  | . . . . . . . . | STX |
| STY | Px | YL→B | Y→W |  |  | 84 | 94 |  |  |  |  |  |  | 8C |  |  |  |  |  |  | . . . . . . . . | STY |
| STZ (7) | Pm | 0→B | 0→W |  |  | 64 | 74 |  |  |  |  |  |  | 9C | 9E |  |  |  |  |  | . . . . . . . . | STZ |
| TRB (8) | Pm | $\overline{AL}$∧B→B | $\overline{A}$∧W→W |  |  | 14 |  |  |  |  |  |  |  | 1C |  |  |  |  |  |  | . . . . . . Z . | TRB |
| TSB (8) | Pm | AL∨B→B | A∨W→W |  |  | 04 |  |  |  |  |  |  |  | 0C |  |  |  |  |  |  | . . . . . . Z . | TSB |

|  |  | immed | accum | dir | dir,x | dir,y | (dir) | (dir,x) | (dir),y | [dir] | [dir],y | abs | abs,x | abs,y | abs1 | abs1,x | d,s | (d,s),y |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  | ← add one cycle if DL ≠ 0 → |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Emulation (E=1) or Native (E=0) Mode. 8 bit (M/X=1) | cycles | 2 | 2 | 3 | 4 | 4 | 5 | 6 | 5 (3) | 8 | 6 | 4 | 4 (3) | 4 (3) | 5 | 5 | 4 | 7 |
|  | bytes | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 2 | 2 |
| Native Mode (E=0), 16 bit (M/X=0) | cycles | 3 | 2 | 4 | 5 | 5 | 6 | 7 | 6 | 7 | 7 | 5 | 5 | 5 | 6 | 5 | 5 | 8 |
|  | bytes | 3 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 2 | 2 |

|  |  |
|---|---|
| V | logical OR |
| Λ | logical AND |
| ⊻ | logical exclusive OR |
| + | arithmetic addition |
| - | arithmetic subtraction |
| ≠ | not equal |
| • | status bit not affected |

|  |  |
|---|---|
| B | byte per effective address |
| W | word per effective address |
| r | relative offset |
| A | Accumulator, AL low half of Accumulator |
| X | Index Register, XL low half of X register |
| Y | Index Register, YL low half of Y register |
| Pc | carry bit |
| M/X | effective mode bit in Status Register (Pm or Px) |
| Ws | word per stack pointer |
| Bs | byte per stack pointer |

Notes:

1. BIT instruction does not affect N and V flags when using immediate addressing mode. When using other addressing modes, the N and V flags are respectively set to bits 7 and 6 or 15 and 14 of the addressed memory depending on mode (byte or word).
2. For all Read/Modify/Write instruction addressing modes except accumulator—
   Add 2 cycles for E=1 or E=0 and Pm=1 (8-bit mode).
   Add 3 cycles for E=0 and Pm=0 (16-bit mode).
3. Add one cycle when indexing across page boundary and E=1 except for STA and STZ instructions.
4. If E=1 then 1→SH and XL→SL. If E=0 then X→S regardless of Pm or Px.
5. Exchanges the carry (Pc) and E bits. Whenever the E bit is set the following registers and status bits are locked into the indicated state: XH=0, YH=0, SH=1, Pm=1, Px=1.
6. Add 1 cycle if branch is taken. In Emulation (E=1) mode only—add 1 cycle if the branch is taken and crosses a page boundary.
7. Add 1 cycle in Emulation mode (E=1) for (dir),y; abs,x; and abs,y addressing modes.
8. With TSB and TRB instruction, the Z flag is set or cleared by the result of A∧B or A∧W.
   For all Read/Modify/Write instruction addressing modes except accumulator—
   Add 2 cycles for E=1 or E=0 and Pm=1 (8-bit mode).
   Add 3 cycles for E=0 and Pm=0 (16-bit mode).

## Table 6. Branch, Transfer, Push, Pull, and Implied Addressing Mode Instructions

| Mnemonic | Bytes | M/X | Cycles | Operation 8 Bit | Cycles | Operation 16 Bit | Implied | Stack | Relative | Status N V M X D I Z C | Mnemonic |
|---|---|---|---|---|---|---|---|---|---|---|---|
| BCC (6) | 2 | — | 2 | PC+r→PC | 2 | PC+r→PC | | | 90 | . . . . . . . . | BCC |
| BCS (6) | 2 | — | 2 | PC+r→PC | 2 | PC+r→PC | | | B0 | . . . . . . . . | BCS |
| BEQ (6) | 2 | — | 2 | PC+r→PC | 2 | PC+r→PC | | | F0 | . . . . . . . . | BEQ |
| BMI (6) | 2 | — | 2 | PC+r→PC | 2 | PC+r→PC | | | 30 | . . . . . . . . | BMI |
| BNE (6) | 2 | — | 2 | PC+r→PC | 2 | PC+r→PC | | | D0 | . . . . . . . . | BNE |
| BPL (6) | 2 | — | 2 | PC+r→PC | 2 | PC+r→PC | | | 10 | . . . . . . . . | BPL |
| BRA (6) | 2 | — | 2 | PC+r→PC | 2 | PC+r→PC | | | 80 | . . . . . . . . | BRA |
| BVC (6) | 2 | — | 2 | PC+r→PC | 2 | PC+r→PC | | | 50 | . . . . . . . . | BVC |
| BVS (6) | 2 | — | 2 | PC+r→PC | 2 | PC+r→PC | | | 70 | . . . . . . . . | BVS |
| CLC | 1 | — | 2 | 0→Pc | 2 | 0→Pc | 18 | | | . . . . . . . 0 | CLC |
| CLD | 1 | — | 2 | 0→Pd | 2 | 0→Pd | D8 | | | . . . . 0 . . . | CLD |
| CLI | 1 | — | 2 | 0→Pi | 2 | 0→Pi | 58 | | | . . . . . 0 . . | CLI |
| CLV | 1 | — | 2 | 0→Pv | 2 | 0→Pv | B8 | | | . 0 . . . . . . | CLV |
| DEX | 1 | Px | 2 | XL-1→XL | 2 | X-1→X | CA | | | N . . . . . Z . | DEX |
| DEY | 1 | Px | 2 | YL-1→YL | 2 | Y-1→Y | 88 | | | N . . . . . Z . | DEY |
| INX | 1 | Px | 2 | XL+1→XL | 2 | X+1→X | E8 | | | N . . . . . Z . | INX |
| INY | 1 | Px | 2 | YL+1→YL | 2 | Y+1→Y | C8 | | | N . . . . . Z . | INY |
| NOP | 1 | — | 2 | no operation | 2 | no operation | EA | | | . . . . . . . . | NOP |
| PEA | 3 | — | 5 | W→Ws, S-2→S | 5 | same | | F4 | | . . . . . . . . | PEA |
| PEI | 2 | — | 6 | W→Ws, S-2→S | 6 | same | | D4 | | . . . . . . . . | PEI |
| PER | 3 | — | 6 | W→Ws, S-2→S | 6 | same | | 62 | | . . . . . . . . | PER |
| PHA | 1 | Pm | 3 | AL→Bs, S-1→S | 4 | A→Ws, S-2→S | | 48 | | . . . . . . . . | PHA |
| PHB | 1 | — | 3 | DB→Bs, S-1→S | 3 | same | | 8B | | . . . . . . . . | PHB |
| PHD | 1 | — | 4 | D→Ws, S-2→S | 4 | same | | 0B | | . . . . . . . . | PHD |
| PHK | 1 | — | 3 | PB→Bs, S-1→S | 3 | same | | 4B | | . . . . . . . . | PHK |
| PHP | 1 | — | 3 | P→Bs, S-1→S | 3 | same | | 08 | | . . . . . . . . | PHP |
| PHX | 1 | Px | 3 | XL→Bs, S-1→S | 4 | X→Ws, S-2→S | | DA | | . . . . . . . . | PHX |
| PHY | 1 | Px | 3 | YL→Bs, S-1→S | 4 | Y→Ws, S-2→S | | 5A | | . . . . . . . . | PHY |
| PLA | 1 | Pm | 4 | S+1→S, Bs→AL | 5 | S+2→S, Ws→A | | 68 | | N . . . . . Z . | PLA |
| PLB | 1 | — | 4 | S+1→S, Bs→DB | 4 | same | | AB | | N . . . . . Z . | PLB |
| PLD | 1 | — | 5 | S+2→S, Ws→D | 5 | same | | 2B | | N . . . . . Z . | PLD |
| PLP | 1 | — | 4 | S+1→S, Bs→P | 4 | same | | 28 | | N V M X D I Z C | PLP |
| PLX | 1 | Px | 4 | S+1→S, Bs→XL | 5 | S+2→S, Ws→X | | FA | | N . . . . . Z . | PLX |
| PLY | 1 | Px | 4 | S+1→S, Bs→YL | 5 | S+2→S, Ws→Y | | 7A | | N . . . . . Z . | PLY |
| SEC | 1 | — | 2 | 1→Pc | 2 | 1→Pc | 38 | | | . . . . . . . 1 | SEC |
| SED | 1 | — | 2 | 1→Pd | 2 | 1→Pd | F8 | | | . . . . 1 . . . | SED |
| SEI | 1 | — | 2 | 1→Pi | 2 | 1→Pi | 78 | | | . . . . . 1 . . | SEI |
| TAX | 1 | Px | 2 | AL→XL | 2 | A→X | AA | | | N . . . . . Z . | TAX |
| TAY | 1 | Px | 2 | AL→YL | 2 | A→Y | A8 | | | N . . . . . Z . | TAY |
| TCD | 1 | — | 2 | A→D | 2 | A→D | 5B | | | N . . . . . Z . | TCD |
| TCS | 1 | — | 2 | A→S | 2 | A→S | 1B | | | . . . . . . . . | TCS |
| TDC | 1 | — | 2 | D→A | 2 | D→A | 7B | | | N . . . . . Z . | TDC |
| TSC | 1 | — | 2 | S→A | 2 | S→A | 3B | | | N . . . . . Z . | TSC |
| TSX | 1 | Px | 2 | SL→XL | 2 | S→X | BA | | | N . . . . . Z . | TSX |
| TXA | 1 | Pm | 2 | XL→AL | 2 | X→A | 8A | | | N . . . . . Z . | TXA |
| TXS | 1 | — | 2 | see note 4 | 2 | X→S | 9A | | | . . . . . . . . | TXS |
| TXY | 1 | Px | 2 | XL→YL | 2 | X→Y | 9B | | | N . . . . . Z . | TXY |
| TYA | 1 | Pm | 2 | YL→AL | 2 | Y→A | 98 | | | N . . . . . Z . | TYA |
| TYX | 1 | Px | 2 | YL→XL | 2 | Y→X | BB | | | N . . . . . Z . | TYX |
| XCE | 1 | — | 2 | see note 5 | 2 | see note 5 | FB | | | . . . . . . . C | XCE |

See Notes on page 13.

## Table 7. Other Addressing Mode Instructions

| Mnemonic | Addressing Mode | Op Code | Cycles | Bytes | Status N V M X D I Z C | Mnemonic | Function |
|---|---|---|---|---|---|---|---|
| BRK | stack | 00 | 7/8 | 2 | . . . . 0 1 . . | BRK | See discussion in Interrupt Processing Sequence section. |
| BRL | relative long | 82 | 3 | 3 | . . . . . . . . | BRL | PC+r→PC where -32768<r<32767. |
| COP | stack | 02 | 7/8 | 2 | . . . . 0 1 . . | COP | See discussion in Interrupt Processing Sequence section. |
| JML | absolute indirect | DC | 6 | 3 | . . . . . . . . | JML | W→PC, B→PB |
| JMP | absolute | 4C | 3 | 3 | . . . . . . . . | JMP | W→PC |
| JMP | absolute indirect | 6C | 5 | 3 | . . . . . . . . | JMP | W→PC |
| JMP | absolute indexed indirect | 7C | 6 | 3 | . . . . . . . . | JMP | W→PC |
| JMP | absolute long | 5C | 4 | 4 | . . . . . . . . | JMP | W→PC, B→PB |
| JSL | absolute long | 22 | 8 | 4 | . . . . . . . . | JSL | PB→Bs, S-1→S, PC→Ws, S-2→S, W→PC, B→PB |
| JSR | absolute | 20 | 6 | 3 | . . . . . . . . | JSR | PC→Ws, S-2→S, W→PC |
| JSR | absolute indexed indirect | FC | 6 | 3 | . . . . . . . . | JSR | PC→Ws, S-2→S, W→PC |
| MVN | block | 54 | 7/byte | 3 | . . . . . . . . | MVN | See discussion in Addressing Mode section |
| MVP | block | 44 | 7/byte | 3 | . . . . . . . . | MVP | |
| REP | immediate | C2 | 3 | 2 | N V M X D I Z C | REP | PΛB̄→P |
| RTI | stack | 40 | 6/7 | 1 | N V M X D I Z C | RTI | S+1→S, Bs→P, S+2→S, Ws→PC, if E=0 then S+1→S, Bs→PB |
| RTL | stack | 6B | 6 | 1 | . . . . . . . . | RTL | S+2→S, Ws+1→PC, S+1→S, Bs→PB |
| RTS | stack | 60 | 6 | 1 | . . . . . . . . | RTS | S+2→S, Ws+1→PC |
| SEP | immediate | E2 | 3 | 2 | N V M X D I Z C | SEP | PVB→P |
| STP | implied | DB | 3 + | 1 | . . . . . . . . | STP | Stop the clock. Requires reset to continue. |
| WAI | implied | CB | 3 + | 1 | . . . . . . . . | WAI | Wait for interrupt. RDY held low until interrupt. |
| XBA | implied | EB | 3 | 1 | N . . . . . Z . | XBA | Swap AH and AL. Status bits reflect final condition of AL. |

See Notes on page 13.

## Table 8. Opcode Matrix

| MSD \ LSD | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | BRK s<br>2 8 | ORA (d,x)<br>2 6 | COP s<br>2 8 | ORA d,s<br>2 4 | TSB d<br>2 5 | ORA d<br>2 3 | ASL d<br>2 5 | ORA [d]<br>2 6 | PHP s<br>1 3 | ORA #<br>2 2 | ASL A<br>1 2 | PHD s<br>1 4 | TSB a<br>3 6 | ORA a<br>3 4 | ASL a<br>3 6 | ORA al<br>4 5 |
| 1 | BPL r<br>2 2 | ORA (d),y<br>2 5 | ORA (d)<br>2 5 | ORA (d,s),y<br>2 7 | TRB d<br>2 5 | ORA d,x<br>2 4 | ASL d,x<br>2 6 | ORA [d],y<br>2 6 | CLC i<br>1 2 | ORA a,y<br>3 4 | INC A<br>1 2 | TCS i<br>1 2 | TRB a<br>3 6 | ORA a,x<br>3 4 | ASL a,x<br>3 7 | ORA al,x<br>4 5 |
| 2 | JSR a<br>3 6 | AND (d,x)<br>2 6 | JSL al<br>4 8 | AND d,s<br>2 4 | BIT d<br>2 3 | AND d<br>2 3 | ROL d<br>2 5 | AND [d]<br>2 6 | PLP s<br>1 4 | AND #<br>2 2 | ROL A<br>1 2 | PLD s<br>1 5 | BIT a<br>3 4 | AND a<br>3 4 | ROL a<br>3 6 | AND al<br>4 5 |
| 3 | BMI r<br>2 2 | AND (d),y<br>2 5 | AND (d)<br>2 5 | AND (d,s),y<br>2 7 | BIT d,x<br>2 4 | AND d,x<br>2 4 | ROL d,x<br>2 6 | AND [d],y<br>2 6 | SEC i<br>1 2 | AND a,y<br>3 4 | DEC A<br>1 2 | TSC i<br>1 2 | BIT a,x<br>3 4 | AND a,x<br>3 4 | ROL a,x<br>3 7 | AND al,x<br>4 5 |
| 4 | RTI s<br>1 7 | EOR (d,x)<br>2 6 | reserve<br>2 2 | EOR d,s<br>2 4 | MVP xya<br>3 7 | EOR d<br>2 3 | LSR d<br>2 5 | EOR [d]<br>2 6 | PHA s<br>1 3 | EOR #<br>2 2 | LSR A<br>1 2 | PHK s<br>1 3 | JMP a<br>3 3 | EOR a<br>3 4 | LSR a<br>3 6 | EOR al<br>4 5 |
| 5 | BVC r<br>2 2 | EOR (d),y<br>2 5 | EOR (d)<br>2 5 | EOR (d,s),y<br>2 7 | MVN xya<br>3 7 | EOR d,x<br>2 4 | LSR d,x<br>2 6 | EOR [d],y<br>2 6 | CLI i<br>1 2 | EOR a,y<br>3 4 | PHY s<br>1 3 | TCD i<br>1 2 | JMP al<br>4 4 | EOR a,x<br>3 4 | LSR a,x<br>3 7 | EOR al,x<br>4 5 |
| 6 | RTS s<br>1 6 | ADC (d,x)<br>2 6 | PER s<br>3 6 | ADC d,s<br>2 4 | STZ d<br>2 3 | ADC d<br>2 3 | ROR d<br>2 5 | ADC [d]<br>2 6 | PLA s<br>1 4 | ADC #<br>2 2 | ROR A<br>1 2 | RTL s<br>1 6 | JMP (a)<br>3 5 | ADC a<br>3 4 | ROR a<br>3 6 | ADC al<br>4 5 |
| 7 | BVS r<br>2 2 | ADC (d),y<br>2 5 | ADC (d)<br>2 5 | ADC (d,s),y<br>2 7 | STZ d,x<br>2 4 | ADC d,x<br>2 4 | ROR d,x<br>2 6 | ADC [d],y<br>2 6 | SEI i<br>1 2 | ADC a,y<br>3 4 | PLY s<br>1 4 | TDC i<br>1 2 | JMP (a,x)<br>3 6 | ADC a,x<br>3 4 | ROR a,x<br>3 7 | ADC al,x<br>4 5 |
| 8 | BRA r<br>2 2 | STA (d,x)<br>2 6 | BRL rl<br>3 3 | STA d,s<br>2 4 | STY d<br>2 3 | STA d<br>2 3 | STX d<br>2 3 | STA [d]<br>2 6 | DEY i<br>1 2 | BIT #<br>2 2 | TXA i<br>1 2 | PHB s<br>1 3 | STY a<br>3 4 | STA a<br>3 4 | STX a<br>3 4 | STA al<br>4 5 |
| 9 | BCC r<br>2 2 | STA (d),y<br>2 6 | STA (d)<br>2 5 | STA (d,s),y<br>2 7 | STY d,x<br>2 4 | STA d,x<br>2 4 | STX d,y<br>2 4 | STA [d],y<br>2 6 | TYA i<br>1 2 | STA a,y<br>3 5 | TXS i<br>1 2 | TXY i<br>1 2 | STZ a<br>3 4 | STA a,x<br>3 5 | STZ a,x<br>3 5 | STA al,x<br>4 5 |
| A | LDY #<br>2 2 | LDA (d,x)<br>2 6 | LDX #<br>2 2 | LDA d,s<br>2 4 | LDY d<br>2 3 | LDA d<br>2 3 | LDX d<br>2 3 | LDA [d]<br>2 6 | TAY i<br>1 2 | LDA #<br>2 2 | TAX i<br>1 2 | PLB s<br>1 4 | LDY a<br>3 4 | LDA a<br>3 4 | LDX a<br>3 4 | LDA al<br>4 5 |
| B | BCS r<br>2 2 | LDA (d),y<br>2 5 | LDA (d)<br>2 5 | LDA (d,s),y<br>2 7 | LDY d,x<br>2 4 | LDA d,x<br>2 4 | LDX d,y<br>2 4 | LDA [d],y<br>2 6 | CLV i<br>1 2 | LDA a,y<br>3 4 | TSX i<br>1 2 | TYX i<br>1 2 | LDY a,x<br>3 4 | LDA a,x<br>3 4 | LDX a,y<br>3 4 | LDA al,x<br>4 5 |
| C | CPY #<br>2 2 | CMP (d,x)<br>2 6 | REP #<br>2 3 | CMP d,s<br>2 4 | CPY d<br>2 3 | CMP d<br>2 3 | DEC d<br>2 5 | CMP [d]<br>2 6 | INY i<br>1 2 | CMP #<br>2 2 | DEX i<br>1 2 | WAI i<br>1 3 | CPY a<br>3 4 | CMP a<br>3 4 | DEC a<br>3 6 | CMP al<br>4 5 |
| D | BNE r<br>2 2 | CMP (d),y<br>2 5 | CMP (d)<br>2 5 | CMP (d,s),y<br>2 7 | PEI s<br>2 6 | CMP d,x<br>2 4 | DEC d,x<br>2 6 | CMP [d],y<br>2 6 | CLD i<br>1 2 | CMP a,y<br>3 4 | PHX s<br>1 3 | STP i<br>1 3 | JML (a)<br>3 6 | CMP a,x<br>3 4 | DEC a,x<br>3 7 | CMP al,x<br>4 5 |
| E | CPX #<br>2 2 | SBC (d,x)<br>2 6 | SEP #<br>2 3 | SBC d,s<br>2 4 | CPX d<br>2 3 | SBC d<br>2 3 | INC d<br>2 5 | SBC [d]<br>2 6 | INX i<br>1 2 | SBC #<br>2 2 | NOP i<br>1 2 | XBA i<br>1 3 | CPX a<br>3 4 | SBC a<br>3 4 | INC a<br>3 6 | SBC al<br>4 5 |
| F | BEQ r<br>2 2 | SBC (d),y<br>2 5 | SBC (d)<br>2 5 | SBC (d,s),y<br>2 7 | PEA s<br>3 5 | SBC d,x<br>2 4 | INC d,x<br>2 6 | SBC [d],y<br>2 6 | SED i<br>1 2 | SBC a,y<br>3 4 | PLX s<br>1 4 | XCE i<br>1 2 | JSR (a,x)<br>3 6 | SBC a,x<br>3 4 | INC a,x<br>3 7 | SBC al,x<br>4 5 |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |

| symbol | addressing mode | symbol | addressing mode |
|---|---|---|---|
| # | immediate | [d] | direct indirect long |
| A | accumulator | [d],y | direct indirect indexed long |
| 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 indexed long |
| 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 | xya | block move |

| legend | |
|---|---|
| instruction mnemonic | addressing mode |
| base number of bytes | base number of cycles |

# Table 9. Detailed Instruction Operation

| ADDRESS MODE | | CYCLE | $\overline{VP}$ | $\overline{ML}$ | VDA | VPA | ADDRESS BUS | DATA BUS | R/$\overline{W}$ |
|---|---|---|---|---|---|---|---|---|---|
| 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 |
| (16 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.RDL 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 |
| (8 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 | New 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 | New 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 | New 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 | New 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 |
| (16 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) | $\overline{IRQ}$.$\overline{NMI}$ | 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) | $\overline{RES}$=1 | 3 | 1 | 1 | 0 | 0 | 1 | PBR.PC+1 | IO | 1 |
| (3 cycles) | $\overline{RES}$=0 | 1c | 1 | 1 | 0 | 0 | 1 | PBR.PC+1 | RES(BRK) | 1 |
| | $\overline{RES}$=0 | 1b | 1 | 1 | 0 | 0 | 1 | PBR.PC+1 | RES(BRK) | 1 |
| | $\overline{RES}$=1 | 1a | 1 | 1 | 0 | 0 | 1 | PBR.PC+1 | RES(BRK) | 1 |
| See 21a Stack | | 1 | 1 | 1 | 1 | 1 | 1 | PBR.PC+1 | BEGIN | 1 |
| (Hardware interrupt) | | | | | | | | | | |

| ADDRESS MODE | | CYCLE | $\overline{VP}$ | $\overline{ML}$ | VDA | VPA | ADDRESS BUS | DATA BUS | R/$\overline{W}$ |
|---|---|---|---|---|---|---|---|---|---|
| 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+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 |
| (12 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+2 | IO | 1 |
| (9 Op Codes) | (6) | 2b. | 1 | 1 | 0 | 0 | PBR.PC+2+OFF | IO | 1 |
| (2 bytes) | | 1 | 1 | 1 | 1 | 1 | PBR.New PC | New 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.New PC | New 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 | New 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 |

ι α

## Table 9. Detailed Instruction Operation (continued)

| | ADDRESS MODE | | CYCLE | $\overline{VP}$, | $\overline{ML}$, | VDA, | VPA | ADDRESS BUS | DATA BUS | R/$\overline{W}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| ★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 | New 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 |

| | ADDRESS MODE | | CYCLE | $\overline{VP}$, | $\overline{ML}$, | VDA, | VPA | ADDRESS BUS | DATA BUS | R/$\overline{W}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| ★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,SBC) | | 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 |

## Table 9. Detailed Instruction Operation (continued)

| ADDRESS MODE | | CYCLE | $\overline{\text{VP}}$, | $\overline{\text{ML}}$, | VDA, | VPA | ADDRESS BUS | DATA BUS | R/$\overline{\text{W}}$ |
|---|---|---|---|---|---|---|---|---|---|
| ★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 | New 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 | New 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) | (10) | 4. | 1 | 1 | 1 | 0 | 0,S-1 | PCH | 0 |
| (0 bytes) | (10) | 5. | 1 | 1 | 1 | 0 | 0,S-2 | PCL | 0 |
| (7 and 8 cycles) | (10)(11) | 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 | New 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 Codes) | | 4. | 1 | 1 | 1 | 0 | 0,S-1 | PCH | 0 |
| (2 bytes) | | 5. | 1 | 1 | 1 | 0 | 0,S-2 | PCL | 0 |
| (7 and 8 cycles) | | 8. | 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 | New 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 | New PCL | 1 |
| (6 and 7 cycles) | | 6. | 1 | 1 | 1 | 0 | 0,S+3 | New PCH | 1 |
| (different order from N6502) | (7) | 7 | 1 | 1 | 1 | 0 | 0,S+4 | PBR | 1 |
| | | 1 | 1 | 1 | 1 | 1 | PBR,New 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 | New PCL-1 | 1 |
| (1 byte) | | 5. | 1 | 1 | 1 | 0 | 0,S+2 | New PCH | 1 |
| (6 cycles) | | 8. | 1 | 1 | 0 | 0 | 0,S+2 | IO | 1 |
| | | 1 | 1 | 1 | 1 | 1 | PBR,New PC | New 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 | New 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 | 0 |
| (7 Op Codes) | | 3 | 1 | 1 | 1 | 0 | 0,S-1 | Register Low | 0 |
| (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 + Offset + 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 ADL, | | 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) | | | | | | | | | |
| ★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,SBC) | | 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 |
| | | 1 | 1 | 1 | 1 | 1 | PBR,PC | Op Code | 1 |
| FFFFFF | | 2 | 1 | 1 | 0 | 1 | PBR,PC+1 | DBA | 1 |
| Dest. Start | N Byte | 3. | 1 | 1 | 0 | 1 | PBR,PC+2 | SBA | 1 |
| | Last | 4 | 1 | 1 | 1 | 0 | SBA,X-2 | Source Data | 1 |
| Source Start | C=0 | 5. | 1 | 1 | 1 | 0 | DBA,Y-2 | Dest Data | 0 |
| Dest. End | | 6 | 1 | 1 | 0 | 0 | DBA,Y-2 | IO | 1 |
| Source End | | 7 | 1 | 1 | 0 | 0 | DBA,Y-2 | IO | 1 |
| 000000 | | 1 | 1 | 1 | 1 | 1 | PBR,PC+3 | New 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 |
| | C=1 | 5. | 1 | 1 | 1 | 0 | DBA,Y+1 | Dest Data | 0 |
| Dest.End | | 6. | 1 | 1 | 0 | 0 | DBA,Y+1 | IO | 1 |
| Source Start | | 7 | 1 | 1 | 0 | 0 | DBA,Y+1 | IO | 1 |
| Dest. Start | | 1. | 1 | 1 | 1 | 1 | PBR,PC | Op Code | 1 |
| | | 2 | 1 | 1 | 0 | 1 | PBR,PC+1 | DBA | 1 |
| 000000 | N Byte | 3 | 1 | 1 | 0 | 1 | PBR,PC+2 | SBA | 1 |
| | C=0 | 4. | 1 | 1 | 1 | 0 | SBA,X+2 | Source Data | 1 |
| MVN is used when the | | 5. | 1 | 1 | 1 | 0 | DBA,Y+2 | Dest Data | 0 |
| destination start address | | 6. | 1 | 1 | 0 | 0 | DBA,Y+2 | IO | 1 |
| is lower (more negative) | | 7 | 1 | 1 | 0 | 0 | DBA,Y+2 | IO | 1 |
| than the source start address. | | 1 | 1 | 1 | 1 | 1 | PBR,PC+3 | New 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 $\overline{\text{NMI}}$ or $\overline{\text{IRQ}}$ active input

(10) R/$\overline{\text{W}}$ remains high during Reset.

(11) BRK bit 4 equals "0" in Emulation mode.

Abbreviations.

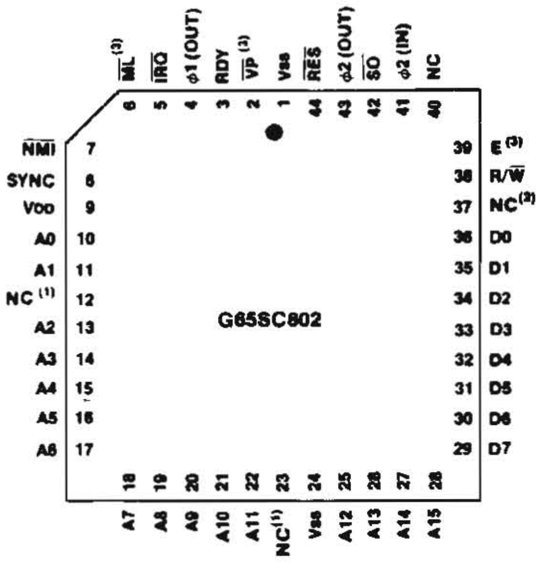
| | |
|---|---|
| AAB | Absolute Address Bank |
| AAH | Absolute Address High |
| AAL | Absolute Address Low |
| AAVH | Absolute Address Vector High |
| AAVL | Absolute Address Vector Low |
| C | Accumulator |
| D | Direct Register |
| DBA | Destination Bank Address |
| DBR | Data Bank Register |
| DO | 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 G65SC816/802 Addressing Modes |
| ● = | New G65SC02 Addressing Modes |
| Blank = | NMOS 6502 Addressing Modes |

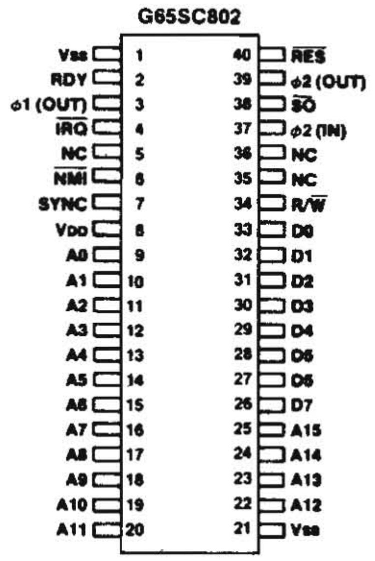
## Pin Function Table

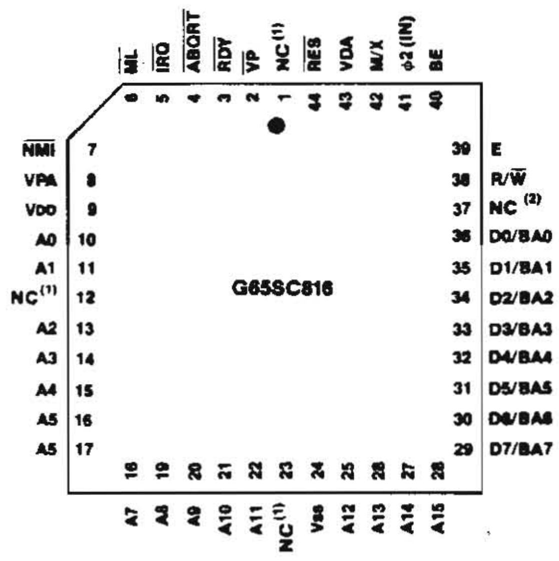
| Pin | Description |
|---|---|
| A0–A15 | Address Bus |
| $\overline{\text{ABORT}}$ | Abort Input |
| BE | Bus Enable |
| φ2 (IN) | Phase 2 In Clock |
| φ1 (OUT) | Phase 1 Out Clock |
| φ2 (OUT) | Phase 2 Out Clock |
| D0–D7 | Data Bus (G65SC802) |
| D0/BA0–D7/BA7 | Data Bus, Multiplexed (G65SC816) |
| E | Emulation Select |
| $\overline{\text{IRQ}}$ | Interrupt Request |
| $\overline{\text{ML}}$ | Memory Lock |
| M/X | Mode Select ($P_M$ or $P_X$) |

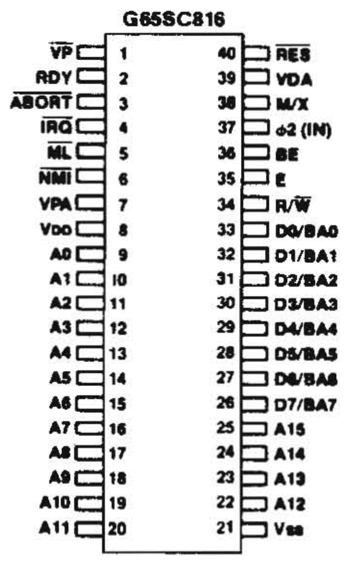
| Pin | Description |
|---|---|
| NC | No Connection |
| $\overline{\text{NMI}}$ | Non-Maskable Interrupt |
| RDY | Ready |
| $\overline{\text{RES}}$ | Reset |
| R/$\overline{\text{W}}$ | Read/Write |
| $\overline{\text{SO}}$ | Set Overflow |
| SYNC | Synchronize |
| VDA | Valid Data Address |
| $\overline{\text{VP}}$ | Vector Pull |
| VPA | Valid Program Address |
| VDD | Positive Power Supply (+5 Volts) |
| Vss | Internal Logic Ground |

## Pin Configuration







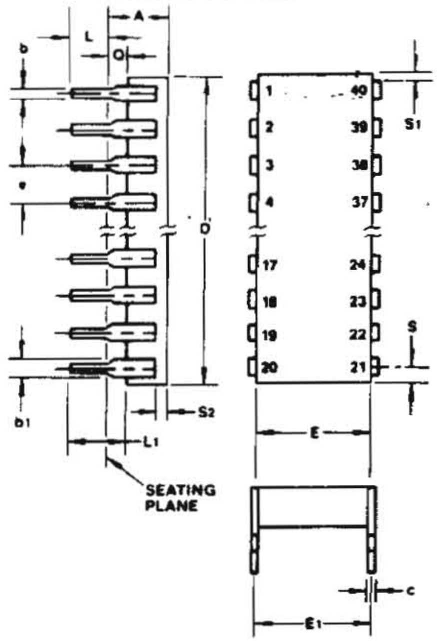


Notes:

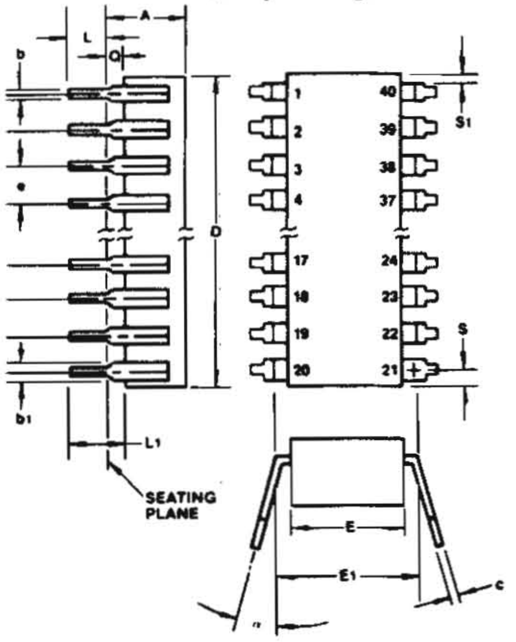
1. Future G65SC802/816 PLCC devices will have Vss added to this pin.
2. Future G65SC802/816 PLCC devices will have VDD added to this pin.
3. New signal pins on G65SC802 not available on 40-pin DIP version.

## Packaging Information

### Ceramic Package

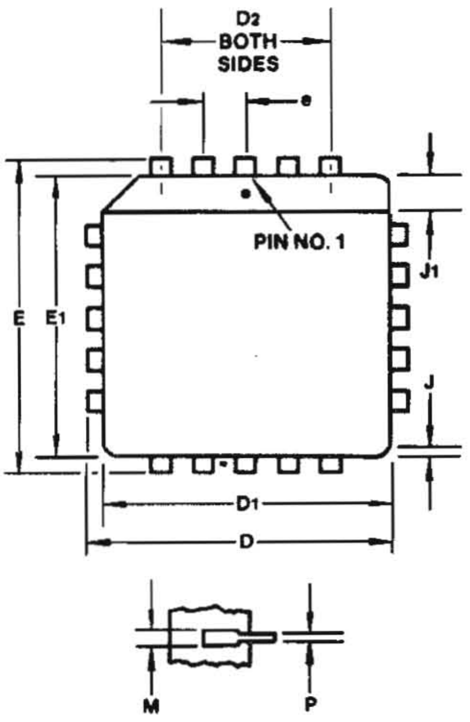


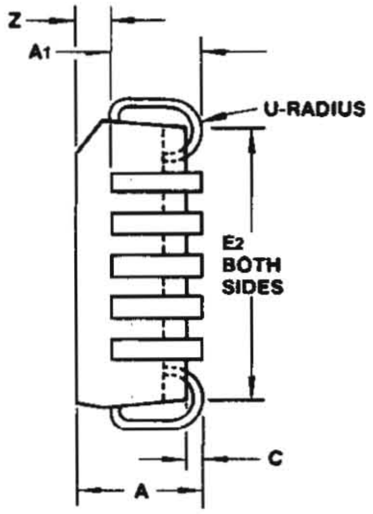
### Plastic & Cerdip Package



| 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 |

*YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THIS SOFTWARE. ANY DOWNLOADING, REPRODUCTION, COPYING OR OTHER USE OF THE SOFTWARE WILL CONSTITUTE ACCEPTANCE OF THESE TERMS AND CONDITIONS.*

## SINGLE-COMPUTER END USER SOFTWARE LICENSE AGREEMENT

APPLE COMPUTER, INC. ("Apple") provides this software and licenses its use. You assume responsibility for the selection of the software to achieve your intended results, and for the installation and use of, and results obtained from, the software.

### LICENSE

Pursuant to this license you may:

1. Use the software only on a single Apple computer. You must obtain a supplementary license from Apple before using the software in connection with systems and multiple central processing units, computer networks or emulations on mainframe or minicomputers.
2. Download the software only on media that is compatible with Apple manufactured computers.
3. Copy the software into any machine readable form for backup purposes in support of your use of the software on the single Apple computer.
4. Transfer the software and license to another party with a copy of this Agreement provided the other party reads and agrees to accept the terms and conditions of this Agreement. If you transfer the software, you must at the same time either transfer all copies, whether in printed or machine-readable form, to the same party or destroy any copies not transferred. Apple grants a license to such other party under this Agreement and the other party will accept such license by its initial use of the software. If you transfer possession of any copy of the software, in whole or in part, to another party, your license is automatically terminated.

This software is protected by United States copyright law. You must reproduce the Apple copyright notice on any copy of the software.

THIS SOFTWARE MAY BE ELECTRONICALLY DISTRIBUTED ONLY BY AUTHORIZED ELECTRONIC DISTRIBUTORS. IT MAY BE DOWNLOADED ONLY FOR PERSONAL OR NON-COMMERCIAL USES ON APPLE COMPUTERS AND MAY NOT BE REDISTRIBUTED OR USED FOR COMMERCIAL PURPOSES WITHOUT AN EXPRESS SOFTWARE DISTRIBUTION LICENSE FROM APPLE. These licenses are available from Apple's Software Licensing Department.

YOU MAY NOT MODIFY, REVERSE COMPILE, DISASSEMBLE, NETWORK, RENT, LEASE, LOAN OR DISTRIBUTE THE SOFTWARE, OR ANY COPY, IN WHOLE OR IN PART. YOU UNDERSTAND THAT UNAUTHORIZED REPRODUCTION OF COPIES OF THE SOFTWARE OR UNAUTHORIZED TRANSFER OF ANY COPY OF THE SOFTWARE MAY SUBJECT YOU TO A LAWSUIT FOR DAMAGES, INJUNCTIVE RELIEF, AND ATTORNEY'S FEES.

Apple reserves all rights not expressly granted to you.

### Export law assurances

You agree and certify that neither the software and documentation nor any direct product thereof (1) is intended to be used for nuclear proliferation or any other purpose prohibited by the United States Export Administration Act of 1979, as amended (the "Act") and the regulations promulgated thereunder, and (2) is being or will be downloaded, shipped, transferred or reexported, directly or indirectly, into any country prohibited by the Act and the regulations promulgated thereunder.

### Government End Users

If you are acquiring the software on behalf of any unit or agency of the United States government, you agree that: (a) the software is "Commercial Computer Software" as that term is defined in Paragraph 27.401 of the DoD Supplement to the Federal Acquisition Regulations (the "Supplement") or is within the equivalent classification of any other federal agencies' regulations; (b) the software was developed at private expense, and no part of it was developed with goverment funds; (c) the government's use of the software is subject to "Restricted Rights" as that term is defined in clause 52.227-7013 (b)(3)(ii) of the Supplement or in the equivalent clause of any other federal agencies' regulations; (d) the software is a "trade secret" of Apple for all purposes of the Freedom of Information Act; and (e) each copy of the software will contain the following Restricted Rights Legend:

"Restricted Rights Legend"

Use, duplication or disclosure is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software clause at FAR 52.227-7013. Manufacturer: Apple Computer, Inc. 20525 Mariani Avenue, Cupertino, Calfornia 95014.

You agree to indemnify Apple for any liability, loss, costs and expense (including court costs and reasonable attorneys' fees) arising out of any breach of the provisions of this Agreement relating to use by the government.

### Term

The license is effective until terminated. You may terminate it at any time by destroying the software together with all copies. The license will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any of the terms or conditions of this Agreement. You agree upon such termination to destroy all copies of the software.

### Disclaimer of Warranty

The software is provided "as is" without warranty of any kind, either express or implied, with respect to its merchantability or its fitness for any particular purpose. The entire risk as to the quality and performance of the software is with you. Should the software prove defective, you (and not Apple or an Apple authorized representative) assume the entire cost of all necessary servicing, repair or correction.

Apple does not warrant that the functions contained in the software will meet your requirements or that the operation of the software will be uninterrupted or error free or that defects in the software will be corrected.

Some states do not allow the cxclusion of implied warranties, so the above exelusion 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.

### Limitation of Remedies

In no event will Apple be liable to you for any lost profits, lost savings or other incidental, special or consequential damages arising out of the use of or inability to use any software even if Apple or an authorized Apple representative has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

Apple's liability to you for actual damages for any cause whatsoever, and regardless of the form of the action, will be limited to the greater of $500 or the money paid for the software that caused the damages or that is the subject matter of, or is directly related to, the cause of action.

### General

This Agreement, if any attempt to network, rent, lease, or sublicense the software, or, except as expressly provided in this Agreement, to transfer any of the rights, duties or obligations under this Agreement, becomes void.

The Agreement will be construed under the laws of the state of California, except for that body of laws dealing with conflict of laws. If any provision of this Agreement shall be held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent permissible, and the remaining provisions of this Agreement shall remain in full force and effect.