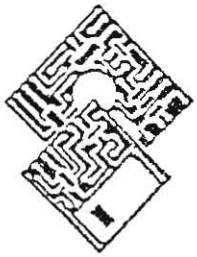


APPLE
PROGRAMMER'S
AND DEVELOPER'S
ASSOCIATION

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

APPLE IIGS Debugger Reference

Version 1.1.1

APDA# K2SBUG

Apple IIGS Debugger Reference

APDA Draft
August 20, 1987

Apple Technical Publications

This document does not include:

- ***final editorial corrections***
- ***final art work***
- ***an index***

APPLE COMPUTER, INC.

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

Apple, the Apple logo, and Macintosh are registered trademarks of Apple Computer, Inc.

Apple IIGSis a trademark of Apple Computer, Inc.

Simultaneously published in the United States and Canada.

Contents

List of Figures

List of Tables

Chapter 1

Overview

The Apple IIGS Debugger is designed to aid you in finding bugs in your programs by allowing you to control execution of any load file. The debugger includes the following features:

- You can step through your code one instruction at a time (in *single-step mode*).
- You can step continuously through your code under control of the debugger (in *trace mode*).
- You can execute your code, or any portion of your code, at full speed when timing is critical (in *real-time mode*).
- You can insert breakpoints in your code at which the debugger automatically suspends execution.
- You can set a breakpoint so that execution is suspended only after the breakpoint location has been passed through a given number of times.
- You can enter the Monitor, execute Monitor commands, and then return to the debugger.
- You can display the debugger's Master Display, which shows the contents of Apple IIGS registers, the breakpoints and memory-protection ranges you have set, portions of the stack and memory, and a disassembly of your program's code.
- You can display 368 contiguous bytes of the direct page.
- You can display any 368 contiguous bytes of RAM in any single memory bank.
- You can display your program's normal screen in any Apple IIGS display mode.
- You can call up on-line help screens for help with any of the debugger's functions.

This manual describes each of the Apple IIGS Debugger's displays and commands. First you are told how to start the debugger and load your program for testing. Next, the debugger's displays are described briefly, followed by an explanation of how to use the debugger. Finally, a reference section describes each of the debugger's displays and commands.

Following the description of the debugger, two desk accessories that are included on your debugger disk are described: Loader Dumper and Memory Mangler. Loader Dumper lets you see where in memory the System Loader has loaded each segment of your program and gives you information about the various tables and variables that the loader uses. Memory Mangler lets you execute a variety of Apple IIGS Memory Manager routines and provides lists of the memory blocks that are in use, purged, and disposed by the Memory Manager.

Once you are familiar with the operaton of the debugger, you can use the summary of debugger commands in the appendix for quick-reference purposes.

Getting Started

This section tells you how to load the debugger and how to load the program to be tested. It also describes the restrictions on the use of the debugger.

What You Need

The Apple IIGS Debugger is located on its own program disk. The program to be tested can be located on the same disk (if it's small enough to fit) or on another disk. Since the debugger is memory-resident, once it is loaded you can remove the debugger disk and load your program from another disk.

Debugger Restrictions

The Apple IIGS Debugger requires approximately 17900 ($4600) bytes of memory (not including the memory requirements of the program you are testing). Because the debugger is loaded into memory by the Apple IIGS System Loader and Memory Manager, you have no control over where in memory the debugger is loaded. If your program is relocatable, it will be loaded by the System Loader and Memory Manager so that conflicts between the debugger and your program are extremely unlikely. If you write absolute code, however, you will not be able to use it with the debugger if any of the following conditions occur:

- The application writes to the area of memory in which the debugger is loaded.
- The application write-protects the area of memory in which the debugger is loaded. For example, if the debugger is in the language card, the application must not write-disable the language card.
- The application "banks out" the area of memory in which the debugger is loaded. For example, if the debugger is in the $D000 space of the language card, the application cannot change language-card banks.
- The application assigns its direct page or stack into the debugger's code space.
- The application uses the same stack space as the debugger.

Important: In single-step and trace modes, if the application writes or steps to a location between 20 bytes before the beginning of the debugger's stack and 8 bytes after the end of the debugger's stack, the debugger stops executing the program and prints S= on the command line. To continue operation, you must change the value of the S register so that it is outside the debugger's stack range. In real-time mode, a stack conflict can cause the debugger to crash.

Loading the Debugger

To load the debugger from APW, type the full pathname of the debugger, ending in the debugger's filename, DEBUG, and press Return. If you are using a hard disk, you can place the debugger in the utility subdirectory of APW, replacing the Exec file named DEBUG that is present in that directory. Then to load the debugger, you can simply type DEBUG and press Return.

Loading Your Program

When you load the debugger, the debugger's copyright and version number are displayed on the command-input line at the bottom of the screen (the command-input line is shown in Figure 1.1). Type the following command:

LOAD *pathname*

Here *pathname* represents the full or partial pathname of the program you wish to debug.

If any ProDOS 16 errors are generated during program load, the ProDOS 16 error number is written to the command line. After a successful load, the following registers are set as indicated:

K/PC	The program bank register (K) and program counter (PC) are set to the starting address of the first segment of the program.
A	The accumulator is set to the User ID of the program loaded. The User ID is assigned by the User ID Manager, as described in the *Apple IIGS Toolbox Reference* manual.
X, Y	The X and Y index registers are set to 0.
P	The processor status register is set to 0.
D	The direct-page register is set to the bottom of the direct-page/stack space of the program.
S	The stack register is set to the top of the direct-page/stack space.

In addition, ProDOS 16 prefix 1 is set to the prefix of the file you loaded.

Note: When you load your program, be sure to make a note of the settings of the K/PC register and other registers (in the Register subdisplay at the top of the screen) before you do anything else. After you have used the debugger to run your program, or have reset any registers with debugger commands, you must know the starting location of your program in memory and starting register values in order to run your program again.

Debugger Display Screens

When you load and start the Apple IIGS Debugger, a display similar to the one shown in Figure 1.1 appears on the screen. This display, referred to as the *Master Display*, contains a command-input line, plus several subdisplays that contain the following types of information:

- the contents of the 65816's registers (the Register subdisplay)
- the contents of the stack (the Stack subdisplay)
- a disassembly of your program's object code (the Disassembly subdisplay)
- the contents of a portion of memory (RAM) that you specify (the RAM subdisplay)

- the breakpoints you have set (the Breakpoint subdisplay)
- the memory ranges you have protected (the Memory-Protection subdisplay)

In addition, you have the option of switching to a display of the contents of the direct page, to a display of the contents of any region of memory you choose, to on-line help, or to any of the display screens normally used by your program.

The commands you can use to select a display are described in the next section. Each of the displays and subdisplays is described briefly in the sections that follow and in detail in the reference section at the end of this chapter.

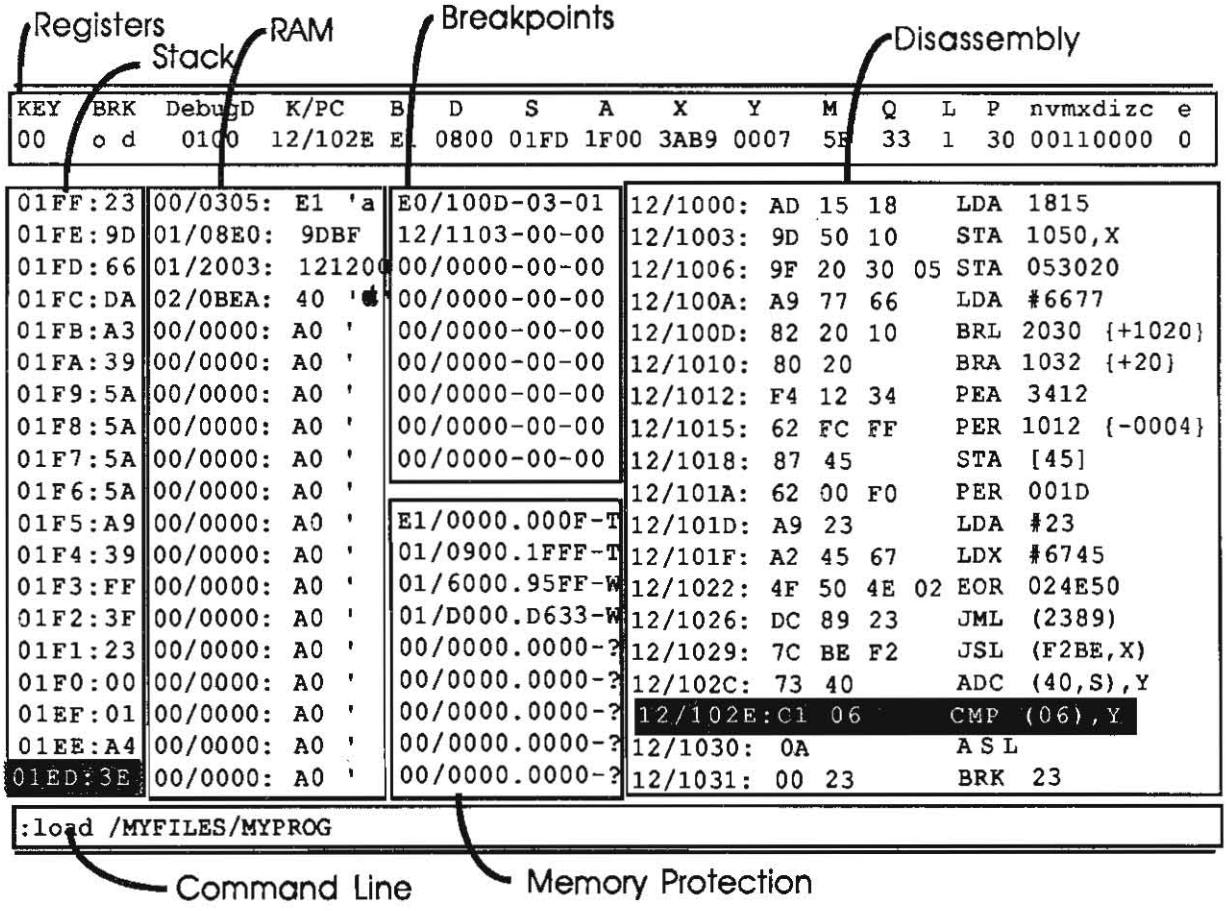


Figure 1.1. Master Display

Selecting Displays

When you start the Apple IIGS Debugger, the Master Display appears on the screen. Use the commands in Table 1.1 to call other displays.

Table 1.1. Selecting Debugger Displays

Display	How to Select
Help screen	From any display, type a question mark (?) and press Return.
Memory	From the Master Display, type the starting address of the memory block you wish to display, followed by a colon (:), and press Return.
Direct-Page	From the Master Display, type D and press Return.
Application	To see the display generated by your application, type OFF on the Master Display command line and press Return, and then start your application as described in the section "Running Your Program" later in this chapter. To change the display mode, press one of the following keys (*these keypress commands work while in single step or trace modes only*—see the section "Single Step and Trace Modes" in this chapter for more information on these keypress commands): 1 text page 1 2 text page 2 4 40-column screen 8 80-column screen T text mode F full-screen graphics M mixed text and graphics L low-resolution graphics H Hi-Res graphics D Double Hi-Res graphics B black-and-white (for Double-Hi-Res graphics)* C color (for Double-Hi-Res graphics)* S Super Hi-Res graphics *These commands work only on a color RGB monitor.
Monitor	To call the Monitor, type MON on the Master Display command line and press Return.
Master Display	In Direct-Page or Memory Display, press Esc. If your application is being displayed, type ON and press Return. From the Monitor, press Control-Y and press Return to return to the Master Display.

Note: If the command filter is in effect, you must hold down one or more keystroke-modifier keys in order to pass commands on to the debugger while your program is running. See the section "The Command Filter" later in this chapter for more information on this function.

The Master Display

The Master Display includes information on many aspects of the debugging process. When you start the Apple IIGS Debugger, a display similar to the one in Figure 1.1 appears. The exact contents of this display depend on the actual contents of memory and on the way in which you have configured the debugger.

In this section, the use of each subdisplay is briefly described. For explanations of all the commands that can be used with a particular subdisplay, see the section about that subdisplay in Chapter 3, "Command Reference."

The Register subdisplay shows the contents of several 65816 hardware registers, the M and Q pseudoregisters, and some flags and addresses used by the debugger. If you are uncertain about the significance of any of these registers and flags, consult the appropriate section in Chapter 3.

The Stack subdisplay shows the contents of 19 bytes of your program's stack. The default location for the stack pointer is the bottom line in the Stack subdisplay, but you can set it to any line you choose with the SET command, as described in the section "Configuring the Master Display" in Chapter 3.

The Disassembly subdisplay shows a disassembly of the machine code in memory using standard APW Assembler mnemonics and address-mode syntax. Disassembly-operand formats are shown in Table 3.1. When you start the debugger, this subdisplay is blank. You can assemble a single instruction and display it at the bottom of this subdisplay, or you can disassemble any 19 contiguous lines of code and list them in this field. When you enter trace or single-step modes, a running disassembly of your program is shown in the Disassembly subdisplay, with the current instruction highlighted. You can use the SET command to change the line that is used for the current instruction.

The RAM subdisplay shows the contents of any 19 memory locations you select. You can display each section as either a single hexadecimal byte with the equivalent ASCII character (or MouseText character if the high bit is set), as a 2-byte value, or as a 3-byte value.

The Breakpoints subdisplay shows from 0 to 17 breakpoint locations you have set. A *breakpoint* is a point in your code at which you want the debugger to suspend execution so you can examine the contents of memory and the registers. Each breakpoint includes a *trigger value*—that is, the number of times you want the code at that location to be executed before execution is interrupted, and the number of times the program has actually passed through this breakpoint so far. You can increase or decrease the number of lines in the Breakpoint subdisplay by simultaneously adjusting the number of lines in the Memory Protection subdisplay.

The Memory Protection subdisplay shows memory-address ranges that you have set either to be executed in real time (*code-trace ranges*, indicated by a T) or to be the only ranges within which code can be executed at all (*code-window ranges*, indicated by a W). You can increase or decrease the number of lines in the Memory Protection subdisplay by simultaneously adjusting the number of lines in the Breakpoint subdisplay.

The command-input line (or *command line*)is used for executing most debugger commands. The only commands *not* executed from this line are single-keystroke commands used to control code trace, cursor movement commands used to enter data into the Master Display, and the Esc key, used to return to the Master Display from other displays. See the section "Command-Line Commands" in Chapter 3 for a list of the commands available from the Master Display.

Test-Program Display

To turn off the debugger's Master Display and show the normal screen display of your program, type `OFF` on the command line and press Return. Although the Apple IIGS Debugger uses only the 80-column text mode, it remembers the last display mode your program was in and switches to that mode when you turn off the Master Display. To change to a different display mode, turn off the Master Display, enter trace or single-step mode, and use one of the display-mode commands listed in the section "Single-Step and Trace Modes" later in this chapter. To return to the Master Display, type `ON` and press Return.

> **Note:** If the command filter is in effect, you must hold down one or more keystroke-modifier keys in order to pass commands on to the debugger. See the section "The Command Filter" later in this chapter for more information on this function.

Memory Display

You can select a display of the contents of any 368 contiguous bytes of RAM in any single memory bank. To get a Memory Display, type the starting address of the memory block, followed by a colon (:), into the command line in the Master Display and press Return. For example, to obtain a display of the contents of the 368 bytes starting at address 1100 in bank 12, type the following (the slash (/) is optional):

```
12/1100:
```

A sample Memory display is shown in Figure 1.2. Each line begins with the memory address of the first byte shown on that line, followed by the contents of 16 memory locations. The memory contents are shown first as hexadecimal values and then as their equivalent ASCII characters. The character set is displayed as follows:

ASCII Value	Displayed As
$00–$1F	. (period)
$20–$7F	normal video
$80–$9F	₀ (inverse-video period)
$A0–$FF	inverse video

```
12/1100: 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 ................
12/1110: 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 ...............
12/1120: 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 !"#$%&'()*+,-./0
12/1130: 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 123456789:;<=>?@
12/1140: 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 ABCDEFGHIJKLMNOP
12/1150: 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 QRSTUVWXYZ[\]^_`
12/1160: 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 abcdefghijklmnop
12/1170: 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 qrstuvwxyz{|}~▒.
12/1180: 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 ................
12/1190: 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F A0 ................
12/11A0: A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 !"#$%&'()*+,-./0
12/11B0: B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 123456789:;<=>?@
12/11C0: C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 ABCDEFGHIJKLMNOP
12/11D0: D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 QRSTUVWXYZ[\]^_`
12/11E0: E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 abcdefghijklmnop
12/11F0: F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF 00 qrstuvwxyz{|}~▒.
12/1200: 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 ................
12/1210: 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 ...............
12/1220: 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 !"#$%&'()*+,-./0
12/1230: 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 123456789:;<=>?@
12/1240: 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 ABCDEFGHIJKLMNOP
12/1250: 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 QRSTUVWXYZ[\]^_`
12/1260: 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 abcdefghijklmnop
:12/1270:
```

Figure 1.2. Sample Memory Display

To change the contents of memory, use the commands described in the section "Altering the Contents of Memory" in Chapter 3. To return to the Master Display, press Esc.

Direct-Page Display

You can select the Apple IIGS Debugger's Direct-Page Display by typing `D` on the command line and pressing Return. This display shows 368 bytes of the direct page starting with the address in the D register. The Direct-Page Display is identical in appearance and function to the Memory Display that you would obtain for a block of memory starting at the address in the D register.

Help Screen

To display a help screen showing the commands available at any time, enter a question mark (?). To return from the help screen to the display from which you called it, press any key *except* Esc. To return to the Master Display, press Esc.

Running Your Program

You can step through your program one instruction at a time or continuously, with the Apple IIGS Debugger intercepting and interpreting each instruction. Executing your program in this fashion gives you maximum control over the process, allowing you to stop at any point and examine the contents of the registers or your program's display. For timing-critical programs and sections of programs, you can also execute the code at full speed. This section explains how to do the following:

- step through your program one instruction at a time (single-step mode)
- step continuously through your program (trace mode)
- execute your program at full speed (real-time mode)
- set and use breakpoints
- set and use memory-protection ranges

Important: In single-step and trace modes, if the application writes or steps to a location between 20 bytes before the beginning of the debugger's stack and 8 bytes after the end of the debugger's stack, the debugger stops executing the program and prints S= on the command line. To continue operation, you must change the value of the S register so that it is outside the debugger's stack range. In real-time mode, a stack conflict can cause the debugger to crash.

Single-Step and Trace Modes

In single-step mode, you can step through your program one instruction at a time. As each instruction is executed, the instruction is highlighted in the Disassembly subdisplay of the Master Display. In trace mode, the Apple IIGS Debugger automatically steps through each instruction in succession; other than being free-running, trace mode is identical to single-step mode. Use the following commands from the command line to initiate single-step and trace modes (the command line is still active if you turn the Master Display off to see your program's display):

Command	Action
S	Enter single-step mode at the current instruction. The current instruction is the next instruction to be executed as indicated by the K/PC register. The K/PC register is updated by the debugger each time an instruction is executed in single-step or trace modes, each time a new program is loaded, and each time you execute a K= , PC=, or K/PC= command. The current instruction appears at the highlighted line of the Disassembly subdisplay; press the Space bar to execute it, or press Return to enter trace mode.
*address*S	Enter single-step mode at address *address*. The K/PC register is set to *address* and the instruction at *address* appears at the highlighted line of the Disassembly subdisplay; press the Space bar to execute it, or press Return to enter trace mode.
T	Enter trace mode at the current instruction (as indicated by the K/PC register and the highlighted line of the disassembly subdisplay). The debugger begins executing code immediately, and continues to execute instructions until you press Esc to stop it or until it reaches a breakpoint or a BRK instruction.
*address*T	Enter trace mode at address *address*. The K/PC register is set to *address* and the debugger begins executing code immediately. The debugger continues to execute instructions until you press Esc to stop it, or until it reaches a breakpoint or a BRK instruction.

Note: When you load your program, be sure to make a note of the values of all the fields in the Register subdisplay before you do anything else. After you have used the debugger to run your program, or have reset any registers with a debugger command, you must know the starting location and register settings for your program in order to run it.

Once you are in either single-step or trace mode, you can use any of the following keypress commands. The commands that change display modes are intended for use with your program's display. Do not use them when the debugger's Master Display is on the screen.

Command	Action
Esc	Terminate trace or single-step mode and return to the command-line.
Space bar	Single-step one instruction.
Return	Start continuous tracing.
R	Trace until the next `RTS`, `RTI`, or `RTL`. This command allows you to trace through one subroutine at a time.
J	Begin to execute code in real time at the current instruction.
X	If the current instruction (the next to be executed) is a `JSL`, execute in real time until the matching `RTL` or until an `RTI` that returns to the following instruction. If the next instruction is not a `JSL`, this command is ignored.
↓	Skip the next instruction. You can use this command to skip over a `BRK` instruction, for example.
Q	Toggle the sound on or off. If the sound is on, the speaker beeps each time an instruction is executed.
1	Change the display to text page 1. Use this command when in 40-column text mode or mixed text and graphics mode.
2	Change the display to text page 2. Use this command when in 40-column text mode or mixed text and graphics mode.
4	Change the display to a 40-column screen. Use this command when in text mode.
8	Change the display to a 80-column screen. Use this command when in text mode.
T	Change the display to text mode.
F	Change the display to full-screen graphics mode.
M	Change the display to mixed text and graphics mode.
L	Change the display to low-resolution graphics mode.
H	Change the display to high-resolution graphics mode.
D	Change the display to double-high-resolution graphics mode.
S	Change the display to super-high-resolution graphics mode. This is the normal Apple IIGS display mode.
B	Change the display to black-and-white double-high-resolution graphics mode.
C	Change the display to color double-high-resolution graphics mode.
←	Change to the slow trace rate.
→	Change to the fast trace rate.
♁	Pause the trace until the Apple key is released.
?	Display a help screen.

Note: If the command filter is in effect, you must hold down one or more

keystroke-modifier keys in order to pass commands on to the debugger. See the section "The Command Filter" later in this chapter for more information on this function.

Real-Time Mode

In real-time mode, the debugger passes control of the computer to the code that you specify. The code runs at full speed, just as it would if you were running it without the debugger.

Before you try running your program in real-time mode, read through the rest of the "Running Your Program" section. As described in the following sections, you can still exercise considerable control over the execution of your program, even if it is running in real-time mode. You can pass keystrokes on to the debugger rather than to your program by specifying a command filter (see the section "The Command Filter"); you can specify memory ranges outside of which no code is executed (see the section "Memory Protection"); and you can cause execution to stop automatically at any breakpoints you specify (see the section "Breakpoints"). You can run your entire program in real-time mode, or you can specify that the code in certain memory ranges is to be run in real-time mode while the rest of the program is run in trace or single-step mode.

Use the following commands from the command line to initiate real-time mode:

Command	Action
*address*X	JSL directly to code at address *address*. If you omit *address*, the current K/PC address is used. (Note that if you omit *address*, you must use an uppercase X for this command.) This command assumes your routine ends in an RTL: the code is executed in real-time mode and when the RTL is executed control returns to the debugger. The debugger automatically turns off the Master Display before executing this command.
*address*J	JML directly to code at address *address* . If you omit *address*, then the current K/PC address is used. This command executes an unconditional jump to *address* and the code at that address is executed in real-time mode. Control does not return to the debugger unless you have set a real breakpoint or a nonbreakpoint BRK is executed (while breakpoints are set to DBRK). See the section "Breakpoints" in this chapter for information on the use of breakpoints. The debugger automatically turns off the Master Display before executing this command.

The Command Filter

The Apple IIGS Debugger normally intercepts all keystrokes and passes them to the debugger's command interpreter. If the program you are debugging requires input from the keyboard, you can set the debugger to pass all keystokes on to the application unless one or more keystroke-modifier keys are also pressed. To select the key or keys to be used as the keystroke modifier, use the KEY command as described in the section "Register Subdisplay" in Chapter 3.

Keystroke modifiers prevent debugger commands from interfering with your program. For example, suppose your program has menu options that are activated by key presses, and that Q causes the program to quit. You can set the keystroke modifier so that commands are passed to the debugger only when you hold down the Option key. Then to toggle sound on or off without causing your test program to quit, you would press Option-Q.

Remember that when you have set a keystroke modifier, you must use that key or key combination in order to send *any* command to the debugger while in trace or single-step mode. For example, to quit trace mode when Option is set as the keystroke modifier, press Option-Esc; to single-step one instruction, press Option-Space bar.

Memory Protection

The Apple IIGS Debugger allows you to specify address ranges within which code is executed at full speed (*code-trace ranges*) and to specify ranges outside which no code is executed (*code-window ranges*). Instructions for setting these ranges are given in the section "Memory-Protection Subdisplay" in Chapter 3.

All code outside a code-trace range (indicated by a `T` on the Memory-Protection subdisplay) is executed in single-step or trace mode. When your code executes a `JSL` to this memory-protection address range, the code inside this range is executed in real time. When the matching `RTL` is encountered, execution returns to single-step or trace mode.

Use code-trace ranges to specify subroutines that must be executed at full speed, such as disk I/O routines. Note that interpreted breakpoints do not function inside code-trace ranges; you must insert real breakpoints to stop execution within a code-trace range. When you start the debugger, a code-trace range is automatically set for the range E1/0000 through E1/000F so that Apple IIGS tool calls are run at full speed.

If one or more code-window address ranges (indicated by `W`'s on the Memory-Protection subdisplay) are specified, then code is executed only if it is inside one of the code-window ranges. Any time the program counter (K/PC) equals an address not in any of the code-window address ranges, execution stops. (If you don't specify any code-window address ranges, code can be executed at any address.)

When one or more code-window address ranges are set, execution is in trace or single-step mode unless a `JSL` is made to a code-trace address range that is completely contained in a code-window address range. Once such a `JSL` is made, execution is in real time until the matching `RTL` is encountered.

You can use code-window ranges to protect code outside the normal code for your program from being executed. Thus, for example, if a bug in your code is causing the system to crash by executing code in the wrong location in memory, you can restrict execution to protect that area of memory by making sure it lies outside any code-window ranges. Then, when your program jumps to that area of memory, execution stops and you can attempt to find the location and nature of the bug.

Breakpoints

A *breakpoint* is an instruction at which execution is suspended. The Apple IIGS Debugger allows you to set up to 17 memory addresses as breakpoints. Breakpoints can be either *real* or *interpreted*. A real breakpoint is a BRK instruction that the debugger has inserted in the code. An interpreted breakpoint is a memory address at which the debugger suspends execution in trace mode. When interpreted breakpoints are set, the debugger compares the address of the instruction about to be executed (that is, the program counter) to the breakpoint addresses before executing each instruction.

Instructions for setting breakpoints are given in the section "Breakpoint Subdisplay" in Chapter 3. In order for breakpoints to work when the debugger is running in real-time mode, you must insert real breakpoints into the code. To set real breakpoints, set the breakpoint addresses as described in the section on the Breakpoint subdisplay, press Esc to return to the command line, and then type IN and press Return. The letter i is displayed under BRK in the upper left corner of the Master Display to indicate that real breakpoints are in. To remove real breakpoints, type OUT on the command line and press Return. The letter o is displayed under BRK in the upper left corner of the Master Display to indicate that real breakpoints are out. Real breakpoints must be removed before you can set, change, or delete breakpoints.

For each breakpoint, you must set a *trigger value*. The trigger value specifies the number of times that the instruction at the breakpoint address must be encountered before the debugger suspends execution. For example, if you set the trigger value for a breakpoint to 2, the debugger executes that instruction the first time it is encountered but suspends execution, rather than executing the instruction, the second time it is arrived at. To disable a breakpoint without removing the breakpoint address, set the trigger value to 0; breakpoints with trigger values of 0 are ignored by the debugger.

In trace mode, execution stops when the trigger value is reached for both real and interpreted breakpoints. In real-time mode, execution stops when the trigger value is reached for real breakpoints only. In single-step mode, the computer beeps when the trigger value is reached.

In trace and single-step modes, execution stops any time a BRK instruction is encountered that is *not* set as a breakpoint. In real-time mode, you can select whether a nonbreakpoint BRK causes a return to the debugger or an exit to the Monitor. To have nonbreakpoint BRK instructions return you to the debugger, type DBRK on the command line and press Return. A d appears next to the i or o in the upper left corner of the Master Display. To have nonbreakpoint BRK instructions cause an exit to the Monitor, type UBRK on the command line and press Return. A u appears in the upper left corner of the Master Display. The default mode is DBRK: you are returned to the debugger when a nonbreakpoint BRK is encountered during real-time execution.

When a breakpoint is triggered, execution stops and you can check the contents of the registers and memory locations shown in the Master Display or Memory Display. To resume execution in trace mode, type T on the command line and press Return. When an interpreted breakpoint is triggered in single-step mode, the computer beeps. Continue pressing the Space bar to continue stepping through the code. When a BRK instruction that you did not set as a breakpoint is encountered in trace or single-step modes, execution stops, the computer beeps, and the Register subdisplay contains information about the state

of the machine when the break occurred. Use the ON command to see the Master Display, if necessary.

To clear all breakpoints (when real breakpoints are out), type CLR on the command line and press Return.

Printing

You can print any debugger text screen, including help screens and your program's display, by typing P on the command line and pressing Return. You cannot use the P command to print graphics screens.

To set the slot to which the debugger assumes the printer is connected, use the SET command as described in the section "Configuring the Master Display" in Chapter 3.

Although the debugger is launched by the Apple IIGS Programmer's Workshop, it is not an APW utility program. Therefore, the printer initialization string and other printer variables you set in APW are not active when you are in the debugger. If you use the Apple IIGS computer's built-in printer port, you can use the Control Panel to set printer communication options.

Using Monitor Routines

The Apple IIGS Monitor consists of a set of ROM-based routines that you can use to perform many functions not otherwise available from the debugger. The Monitor provides the following features:

- You can examine the contents of any locations in memory, including ROM routines.
- You can change the contents of any locations in RAM.
- You can copy a block of data from one location in memory into another and verify that the contents of the two blocks of memory are identical.
- You can clear a range of memory.
- You can search for one or more bytes within a range of memory addresses.
- You can examine and change the contents of Apple IIGS registers.
- You can convert hexadecimal numbers to decimal and vice versa.
- You can perform hexadecimal addition and subtraction.
- You can run a machine-language program that is in memory.
- You can enter machine-language programs directly from the keyboard, using standard 65816 mnemonics (this Monitor routine is called the Mini-Assembler).
- You can disassemble a range of addresses.
- You can start Applesoft BASIC.
- You can change the screen display.
- You can change the cursor symbol.
- You can redirect input and output links.

- You can call Apple IIGS tool sets.
- You can set and display the system clock time and date.

To enter the Apple IIGS Monitor, type MON on the debugger's command line and press Return. To return from the Monitor to the debugger, press Control-Y and then press Return. The Apple IIGS Monitor is described in detail in the *Apple IIGS Firmware Reference* manual.

Chapter 2

Using the Debugger

The Apple IIGS Debugger allows you to load your program into memory and to run through it under the debugger's control. As the program executes, you can examine the contents of the 65816's registers, of your program's direct page and stack, and of any locations in memory in which you are interested. You can interact with the program as required, returning to the debugger's display at breakpoints that you set or when the program crashes.

The Apple IIGS Debugger can display an assembly-language disassembly of your program's machine code. It cannot execute your source code or recreate your source code from machine code. Therefore, the debugger is easiest to use with assembly-language programs. However, even if your program was written in a higher level language and you have no knowledge of assembly language, you can use the debugger to determine in which load segment the problem lies. You can also gain a better understanding of the operation of your program by examining the contents of the stack, direct page, memory, and registers.

We do not have the space here to examine in detail all of the abilities of the Apple IIGS Debugger, but we will give you some hints that should help get you started debugging your program.

Debugging Segmented Programs

In order to use the Apple IIGS Debugger to debug a segmented program, you must know where in memory each segment has been loaded. In the case of dynamic segments, you must know not only where the segment has been loaded, but whether it has been loaded. This information is available through a desk accessory provided with the Apple IIGS Debugger called Loader Dumper.

To load your program using the debugger and to determine where in memory each segment is loaded, use the following procedure:

1. From the APW command line, type `DEBUG` and press Return to call the debugger. The debugger Master Display will appear on the screen.
2. Type `LOAD` *pathname*, where *pathname* is the pathname of the program you want to debug, and press Return. Your program is now loaded into memory.
3. Press Apple-Contol-Esc to get the Desk Accessories menu.
4. Select `Loader Dumper` from the Desk Accessories menu. The Loader Dumper main menu will appear on the screen.
5. Select `Dump Pathname Table` from the Loader Dumper main menu. The pathname table provides a cross-reference of pathnames and User IDs.

6. Scroll through the pathname table (by pressing Return for each pathname) until you find the pathname of the program you are testing. Make a note of the User ID of the program.
7. Press Esc to get back to the Loader Dumper main menu.
8. Select `Get UserID Information` from the main menu.
9. Type in the User ID of your program in response to the prompt that appears on the screen.
10. A listing of all the load segments in your program appears on the screen. Write down the memory locations of all the segments.
11. Press Esc to return to the main menu, press Esc again to quit Loader Dumper, and then select `Quit` from the Desk Accessories menu to return to the debugger.

You now have several possible courses of action open to you. If you do not have any idea in which load segment your program is crashing, you can start by running the program until it crashes and then examining the debugger display to determine the location of the problem instruction. If you know in which segment the problem lies, you can go immediately to that segment, or you can set a breakpoint at the beginning of that segment and run the program until it stops automatically at that breakpoint.

Watching a Running Disassembly

If your program does not require any input from the keyboard, you can watch a disassembly on the debugger screen as the program executes to find the exact location at which it goes astray. This technique will probably be useful only for short programs or programs that crash almost immediately upon execution, because the program will execute very slowly while the debugger display is on the screen.

To run your program under control of the Apple IIGS Debugger, with a running disassembly appearing on the screen, use the following procedure:

1. If you have not already done so, load your program as described above. Write down the information in the Register subdisplay of the debugger so that you can return the machine to its initial state each time you run your program.
2. Type `S` and press Return. The debugger is now in single-step mode, starting with the first instruction of your program. Each time you press the Space bar, the instruction highlighted in the Disassembly subdisplay is executed. To return to the debugger's command line, press Esc. Watch the contents of the registers and the stack as you execute commands. If any specific memory locations are critical to the execution of the program, you can display those locations in the RAM subdisplay of the Master Display.

   To execute commands automatically in quick succession (that is, to enter trace mode), press Return. To start trace mode from the debugger's command line, type `T` and press Return. Your program will begin executing under debugger control, one instruction at a time in rapid succession. The speaker beeps each time an instruction is executed. You can turn off the speaker by pressing Q. You can stop execution at any time by pressing Esc.
3. When your program executes a `BRK` instruction, the disassembly stops scrolling. The last execution executed (the `BRK` instruction) is highlighted. The last several

instructions executed appear above the current instruction. A BRK instruction is actually a null (a zero byte). Since a BRK instruction is not a normal part of a program, the fact that your program executed one means that some previous instruction (contrary to your intent when you wrote the program) sent the program off to the wrong place in memory. With luck, the instruction that sent your program off into Never Never land will still be on the screen.

If the errant instruction is no longer on the screen, set a code-window range as described in the section "Using Memory-Protection Ranges" later in this chapter and run the program again.

Important: Remember to restore all the fields in the debugger's Register subdisplay to their original values before attempting to rerun the program. If you do not, the program will probably not run correctly through no fault of its own.

Using Breakpoints

If you have to interact with your program in order for it to run, if you have some idea of which segment contains the bug, or if you just want to execute the program more quickly, you can set one or more *breakpoints* before running the program. A breakpoint is a location at which the debugger suspends execution of the program, giving you the opportunity to examine the disassembly and the state of the machine at that location.

To set breakpoints and run the program under debugger control, use the following procedure:

1. If you have not already done so, load your program as described above. Write down the information in the Register subdisplay of the debugger so that you can return the machine to its initial state each time you run your program.
2. As described above, use the Loader Dumper routine to determine the starting locations of the load segments of your program.
3. Back in the debugger, type BP and press Return in order to specify breakpoints. Following the instructions in the section "Breakpoint Subdisplay" in Chapter 3, set breakpoints at the beginning of each load segment (if you do not know in which segment the bug lies) or at the beginning of any segment that you want to examine more closely.
4. If you must interact with the program in order for it to run, set a keystroke modifier that will not interfere with your program. (The keystroke modifier is a key that you must press simultaneously with any key that you want the debugger to interpret as a debugger command. For example, if you set the Option key as the keystroke modifier, you must press Option-Esc to terminate trace mode. If you do not press the keystroke-modifier key, the debugger ignores the keypress and your program is free to act on it.)
5. Type OFF and press Return. The debugger display will be cleared so you can see the normal display of your program. Trace mode also runs more quickly with the debugger's display turned off.
6. Type T and press Return. Your program will begin executing under debugger control, one instruction at a time in rapid succession. The speaker beeps each time an instruction is executed. You can turn off the speaker by pressing Q (remember also to press the keystroke-modifier key that you have set, if any). Interact with

your program as you normally do (it will run more slowly than normal). You can stop execution at any time by pressing Esc (remember the keystroke modifier!).

7. When the debugger comes to a breakpoint, the debugger's Master Display appears on the screen. The number shown under K/PC in the Register subdisplay indicates the location of the instruction at which the program stopped. To see a disassembly of the program starting at the breakpoint location, type the K/PC address followed by an L (for example, 010240L) and press Return.
8. If you are narrowing in on the bug, you might want to use single-step mode with the debugger's Master Display on the screen. To step through the segment one instruction at a time, type S and press Return. Now each time you press the Space bar (remember the keystoke modifier), one instruction is executed. You can watch the contents of the stack and the machine's registers as each instruction executes. You can also display the contents of up to 19 memory locations, which you can watch as the program executes.
9. To return to executing the program automatically, press Esc to exit single-step mode, and then type OFF and T as before. Each time the debugger gets to a breakpoint, it will return you to the Master Display.

If at any time during execution of the program a dynamic segment is loaded, you can pause execution of your program and go back to Loader Dumper to find out where in memory it has been placed.

Breakpoints can be used for other purposes than finding a particular segment. Suppose, for example, that your program seems to run alright for awhile, then crashes after having lulled you into a false expectation of success. In this case, it is possible that some routine is failing, not the first time it is run, but only after going through several iterations. To handle such a situation without stopping the program every time the routine is executed, you can include a *trigger value* for a breakpoint. The debugger counts the number of times it encounters the breakpoint and suspends execution only when the trigger value is reached.

If you must execute a routine a full speed in order for it to work correctly, you can insert *real breakpoints* into the code. When you do so, the debugger actually inserts BRK instructions into memory at the breakpoint locations. Trigger values work for real breakpoints that you have set; the debugger will still suspend execution any time it encounters a BRK instruction that you did *not* set as a breakpoint.

Using Memory-Protection Ranges

It may be that certain portions of your code must be executed at the full speed of the 65816 microprocessor. To cause this to happen automatically every time you trace through the program, you can set any areas of memory you choose as *code-trace ranges*. When the program executes a jump to a location within a code-trace range, the debugger relinquishes control to your program and the code is executed at full speed. The portion of memory used to run tool calls is automatically set as a code-trace range when you load the debugger.

You can also set one or more portions of memory (the limits of your code as revealed by Loader Dumper, perhaps) as *code-window* ranges. If the program attempts to execute code outside the code-window ranges you have set, execution stops. You might want to set a code-window range, for example, if your program is executing a jump to execution at some

incorrect memory location and trashing memory before it stops, so that you have to reboot the machine every time you try to run the program with the debugger.

If your program loads a dynamic segment during execution and you want to pause as soon as control is transferred to the dynamic segment, you can set code window ranges to include all the static segments at the start of the program. Then when the dynamic segment is loaded and control is transferred to it, the program will be outside any code window range and execution will stop.

> **Important:** Once you have set any code window range, no code will be executed that is not in a code window range. Therefore, if you set a code window range equal to the memory location of one of your program segments, you must set code window ranges for all other segments that you want to run. Remember also to set the portion of memory used to run tool calls (E1/0000–E1/000F) to a code window range if your program makes any tool calls.

Debugging Multilanguage Programs

One of the advantages of using the APW development environment is that it allows you to link together routines written in different programming languages. This facility can lead to unique problems, however, especially when information is passed between routines written in different languages.

To use the Apple IIGS Debugger to debug parameter-passing problems, use the following procedure:

1. Set breakpoints at the beginning of the calling segment and at the beginning of the called segment.
2. Run the program in trace or real-time mode until the first breakpoint is reached. Search this segment to find the `JSL` that calls the other segment. If you do not need to interact with the program, the easiest way to do this is to run in trace mode with the Master Display on the screen until the second breakpoint is reached. Then both the `JSL` and the first instruction of the called segment will be on the screen. If you cannot do that, try listing a disassembly (using the *address*`L` command) until you see the appropriate `JSL`.
3. Set a breakpoint just before the `JSL` that calls the second segment. You can remove the other two breakpoints now if you wish.
4. Run the program until the `JSL` breakpoint is reached. Parameters are normally passed either on the stack or in the A, X, and Y registers. The actual information passed may be a pointer to the data rather than the data itself. By examining the contents of the registers, the stack, and memory, determine where the parameter is that is being passed and that it has the value you expect.
5. Execute the `JSL`. The return address should have been added to the stack.
6. Step through the segment in single step-mode. Is the called routine reading the parameter passed to it? If more than one parameter was passed, are the parameters being read in the correct order? Is an integer being handled as floating point, or is an ASCII string being handled as a number? Is a number being truncated or rounded inappropriately? By a careful study of the action of the called routine, you should be able to determine the source of the problem.

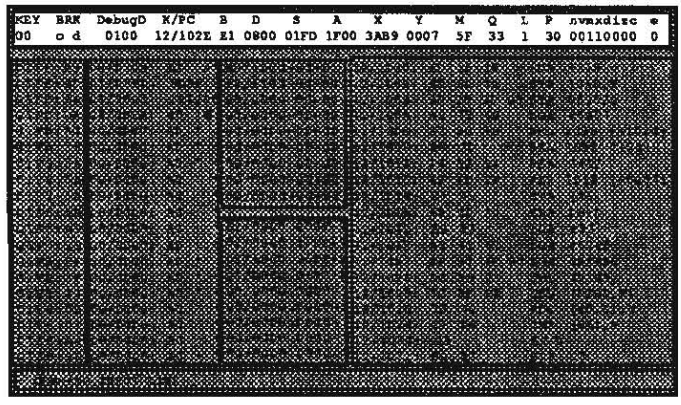
7. If all parameters are being passed correctly, perhaps the problem occurs when the results are passed back to the calling routine. To find the `RTL`, return to the `JSL` and start single-step mode. Then press the R key; the debugger will enter trace mode and automatically stop when the next `RTL` is reached. You might have to do this several times until you reach the right `RTL`. Study the stack and registers as before to determine whether the results are being passed correctly back to the calling routine.

Chapter 3

Command Reference

In this chapter, each of the subdisplays of the Master Display is described in detail. This chapter includes commands for customizing the Master Display and for setting memory addresses, memory-protection ranges, and breakpoints. It describes the use of the Disassembly subdisplay, as well as of all the commands that you can enter on the command-input line.

Register Subdisplay



The Register subdisplay, along the top of the Master Display, shows the contents of the Apple IIGS Debugger's registers and the 65816's registers.

Debugger Registers

The Apple IIGS Debugger's registers are displayed toward the left end of the Register subdisplay as follows:

Key: keystroke modifier. This hexadecimal number indicates the key or key combination that you can use as a command filter to prevent debugger commands from interfering with your test program when your program is running in trace or single-step modes. To select the key or keys to be used as the keystroke modifier, use the following command:

KEY=*keynum* Each bit of the binary number represented by the hexadecimal number *keynum* specifies one of the keys to be used as a keystroke modifier; set that bit to 1 to make that key part of the keystroke modifier. The bits are assigned as follows:

Bit:	7	6	5	4	3	2	1	0
Key:	A	O		K	R	CL	C	S
Hex Value:	80	40	20	10	08	04	02	01

The abbreviations in this diagram refer to the following keys:

Abbreviation	Key
S	Shift
C	Control
CL	Caps Lock
R	repeat: hold the key down until it repeats
K	any key on an external keypad (not the keypad on the Apple IIGS keyboard)
O	Option
A	♁

For example, to set both the Shift and Caps Lock keys as keystroke modifiers, use the following command:

```
KEY=05
```

The KEY value in the register subdisplay changes to 05 to indicate the key combination that is set as the keystroke modifier (01 for the Shift key plus 04 for the Caps Lock key). Now, when you want to send a command to the debugger while in trace or single-step modes, press both the Shift and Caps Lock keys while pressing the key that invokes the command. For example, to switch to the slow trace rate, press the following key combination:

Shift-Caps Lock-←.

BRK: breakpoint flags. The first flag reads `i` (for *in*) if you have used the debugger to set real breakpoints in the program. If you have transparent breakpoints set, this flag reads `o` (for *out*). The second flag reads `d` (for *debugger*) if BRK instructions (other than those that you have inserted with the debugger) return you to the debugger. If such BRKs cause an exit to the Monitor, this flag reads `u` (for *user*).

DebugD: debugger's direct page. This value indicates the starting location of the debugger's 1K byte direct-page/stack segment in bank $00. For example, if the Debugger's direct page begins at 00/1000, then `DebugD` reads `1000`.

65816 Registers

The 65816's registers are displayed as follows:

K/PC: program bank register (K) and program counter (PC). The program bank register serves as the upper 8 bits of the 24-bit address of the next instruction to be executed; the program counter holds the lower 16 bits of the address of the next instruction. There is no carry from the high bit of the PC into the low bit of the K register when the PC is

incremented. When you specify an address in a debugger command and do not specify the bank number, the current value of the K register is used.

B: data-bank register. This value provides the upper 8 bits of the address in addressing modes that generate only the lower 16 bits.

D: direct-page register. This value determines the location of the direct page in bank $00. For example, if the direct page begins at 00/1234, then D reads 1234.

S: stack pointer. This register indicates the next available location on the stack. If the emulation-mode flag e = 1, S must be between $0100 and $01FF.

A: accumulator: This register stores first one operand and then the result for most arithmetic and logical operations. This register is 2 bytes wide if the emulation-mode flag e = 0 and the memory/accumlator-mode flag m = 0; otherwise, it is considered to be 1 byte wide (though the high byte can still be accessed through an XBA instruction).

X: X register. This register is used to provide index values for address calculations, and it holds operands for some arithmetic and logical operations. This register is 2 bytes wide if e = 0 and the index-register-mode flag x = 0; otherwise, it is considered to be 1 byte wide (the high byte is forced to 0).

Y: Y register. This register is used to provide index values for address calculations, and it holds operands for some arithmetic and logical operations. This register is 2 bytes wide if e = 0 and x = 0; otherwise, it is considered to be 1 byte wide (the high byte is forced to 0).

M: machine-state register. This pseudoregister, located at $C068 (in any of banks $00, $01, $E0, or $E1), can be used to set a variety of Mega II-chip soft switches. The M register is described in detail in the *Apple IIGS Hardware Reference* manual. The bits that comprise this hexadecimal number indicate the status of the following machine states:

Bit	Name	Meaning
0	Slot Cx ROM	If this bit is 1, internal ROM at $Cx00 is active; if 0, external ROM (that is, ROM on the circuit board at $Cx00) is active.
1	ROMBANK	This bit is reserved; it must equal 0.
	Warning: Setting ROMBANK to 1 will almost certainly cause the system to crash.	
2	BANK2	If this bit is 0, bank 1 language-card RAM (at $D000 through $DFFF) is selected; if 1, bank 2 is selected. Switching banks with this bit does not write-enable language-card RAM. Use the L flag to both write-enable RAM and switch language-card banks.
3	RDROM	If this bit is 1, language-card ROM is read-enabled; if 0, language-card RAM is read-enabled.
4	RAMWRT	If this bit is 0, main-memory RAM is write-enabled; if 1, auxiliary-memory RAM is write-enabled.
5	RAMRD	If this bit is 0, main-memory RAM is read-enabled; if 1, auxiliary-memory RAM is read-enabled.
6	PAGE2	If this bit is 0, text page 1 is selected; if 1, text page 2 is selected.

7	ALTZP	If this bit is 0, bank-switched memory, stack, and zero page are in main memory; if 1, they are in auxiliary memory.

Q: "quagmire" register. This pseudoregister is composed of the lower 7 bits of the shadow register at $C035, and the high bit of the configuration register at $C036. The bits of this hexadecimal number set the following states:

0	text page 1	1 = shadowing off
1	Hi-Res graphics page 1	1 = shadowing off
2	Hi-Res graphics page 2	1 = shadowing off
3.	Super Hi-Res graphics	1 = shadowing off
4.	auxiliary-memory Hi-Res graphics	1 = shadowing off
5.	reserved	must be 0
6.	IOLC (I/O and language card)	1 = shadowing off
7.	high-speed operation	1 = high speed on

Shadowing is described in the *Technical Introduction to the Apple IIGS* manual and the *Apple IIGS Hardware Reference* manual.

L: language-card bank register. This flag emulates the Monitor *value*=L command. Set L = 0 to write-enable language-card RAM and select bank 1. Set L = 1 to write-enable language-card RAM and select bank 2. Changing L automatically changes bit 2 of the M register.

P: processor status register. This register contains status flags and mode-select bits. The individual bits are shown at the right end of the Register subdisplay, as follows:

Flag	Name	Meaning
n	negative-result flag	If n = 1, the result was negative.
v	overflow flag	If v = 1, an overflow occurred.
m		In 65816 native mode (e = 0), this bit is the memory/accumulator-mode select. If m = 0, memory and accumulator references are 16 bits; if m = 1, they are 8 bits. In 6502 emulation mode (e = 1), this bit has no significance and the label changes from m to 1.
1		See flag m.
x	index-register-mode select (65816 native mode; e = 0)	If x = 0, the X and Y registers are 16 bits; if x = 1, then 8 bits. In 6502 emulation mode (e = 1), this bit changes to b.
b	break flag (6502 emulation mode; e = 1)	If b = 1, the interrupt was a break. In native mode (e = 0), this bit changes to x.
d	decimal-mode select	If d = 0, binary mode is selected; if d =1, decimal mode is selected.
i	IRQ-disabled flag	If i = 1, interrupt requests (IRQ) are disabled.
z	zero-result flag	If z = 1, the result was zero.
c	carry flag	If c = 1, a carry occurred.
e	emulation-mode flag	If e = 1, 6502 emulation mode is selected.

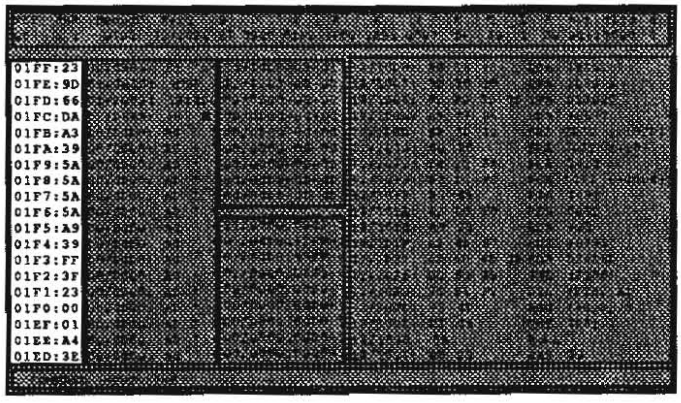
Altering the Contents of Registers

To alter the contents of the registers displayed in the Master Display, type one of the following commands on the command line and press Return (note that these commands are case sensitive):

Command	Action
e	Toggle the e flag: if this flag is set to 1, change it to 0; if it's set to 0, change it to 1.
x	Toggle the x flag: if this flag is set to 1, change it to 0; if it's set to 0, change it to 1. This command works only if e = 0.
m	Toggle the m flag: if this flag is set to 1, change it to 0; if it's set to 0, change it to 1. This command works only if e = 0.
register=value	Set the register specified by *register* to the value specified by *value*. The values for all registers are given as hexadecimal numbers, except for the processor status bits, which can be either 1 or 0. Register names are case sensitive. For example, to set the X index register to $12E0, use the following command: `X=12E0`

Note: The lengths of the X, Y, and A registers depend on the settings of the e, x, and m processor bits, as discussed in the register descriptions in the preceding section, "65816 Registers."

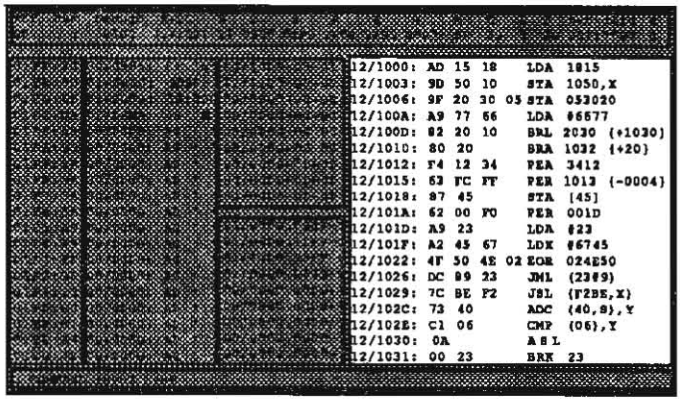
Stack Subdisplay



The Stack subdisplay, along the left side of the Master Display, shows the contents of a portion of the 65816's stack. This subdisplay shows the addresses and contents of the memory locations just before and just after the location pointed to by the stack pointer. The current location of the stack pointer is shown in the Register subdisplay (see the earlier section, "Register Subdisplay") and is highlighted in the Stack subdisplay. You can change the position of the current stack location within this subdisplay by using the `SET` command. See the section "Configuring the Master Display" later in this chapter for a discussion of the `SET` command.

See the section "RAM Subdisplay" later in this chapter for a discussion of commands you can use to change values in memory.

Disassembly Subdisplay



The Disassembly subdisplay, along the right side of the Master Display, shows a disassembly of your program's object code using standard APW 65816 assembly mnemonics and address-mode syntax. As shown in Figure 3.1, each line of this subdisplay is composed of three parts: the address, the bytes stored starting at that address, and the disassembled version of those bytes.

```
12/1000: AD 15 18    LDA 1815
12/1003: 9D 50 10    STA 1050,X
12/1006: 9F 20 30 05 STA 053020
12/100A: A9 77 66    LDA #6677
12/100D: 82 20 10    BRL 2030 {+1020}
12/1010: 80 20       BRA 1032 {+20}
12/1012: F4 12 34    PEA 3412
12/1015: 62 FC FF    PER 1012 {-0004}
12/1018: 87 45       STA [45]
12/101A: 62 00 F0    PER 001D
12/101D: A9 23       LDA #23
12/101F: A2 45 67    LDX #6745
12/1022: 4F 50 4E 02 EOR 024E50
12/1026: DC 89 23    JML (2389)
12/1029: 7C BE F2    JSL (F2BE,X)
12/102C: 73 40       ADC (40,S),Y
12/102E: C1 06       CMP (06),Y
12/1030: 0A          ASL
12/1031: 00 23       BRK 23
```

Figure 3.1. Disassembly Subdisplay

For example, look at the first line of the Disassembly subdisplay in Figure 3.1. The first part of the line, the address, is composed of the high-order byte ($12) that specifies the memory bank, followed by the 2-byte ($1000) location within that bank of the first byte in the instruction. The next 1 to 4 bytes (`AD 15 18` in this example) show the contents of memory starting at that location, corresponding to the instruction that the Apple IIGS Debugger interpreted to start at that location. The last part of the line (`LDA 1815`) shows the 65816 opcode and operand. Operand address formats, summarized in Table 3.1, are the same as those used by the Apple IIGS Monitor Mini-Assembler and described in the *Apple IIGS Firmware Reference* manual. All numbers are hexadecimal.

Table 3.1. Disassembly Operand Formats

Addressing Mode	Example Operand
Absolute	`1234`
Absolute indexed with X	`1234,X`
Absolute indexed with Y	`1234,Y`
Absolute indexed indirect	`(1234,X)`
Absolute indexed long	`081234,X`
Absolute indirect	`(1234)`
Absolute long	`081234`
Accumulator	
Block move	`5678`
Direct page	`12`
Direct page indexed with X	`12,X`
Direct page indexed with Y	`12,Y`
Direct page indexed indirect with X	`(12,X)`
Direct page indirect	`(12)`
Direct page indirect indexed with Y	`(12),Y`
Direct page indirect indexed long	`[12],Y`
Direct page indirect long	`[12]`
Immediate	`#12` or `#1234`
Implied	
Program counter relative	`1000 {+12}`
Program counter relative long	`1000 {-1234}`
Stack	
Stack relative	`10,S`
Stack relative indirect indexed with Y	`(10,S),Y`

You can change the position of the current instruction within the subdisplay by using the `SET` command. See the section "Configuring the Master Display" later in this chapter for a discussion of the `SET` command.

> **Note:** The Apple IIGS Debugger disassembler interprets all bytes in memory as 65816 instructions; it cannot distinguish between code and data. Strings of data therefore appear as nonsense instructions in the Disassembly subdisplay.

When you start the debugger, the Disassembly subdisplay is blank. The debugger enters values into this subdisplay in response to any of the following command-line commands. Whenever you enter an address, you can include the slash after the bank or not, as you wish. If you do not include a bank number, the current value of the K register is used for the bank.

Command	Action
*address*L	The contents of memory starting at *address* are disassembled, and the next 19 lines of that disassembly are displayed.
L	The next 19 lines of the disassembly are displayed, starting at the current K/PC address. If you have executed no previous disassembly commands, the disassembly starts at the starting address of the first segment of your program.
*address*T	Enter trace mode at address *address*. If you omit *address*, the current K/PC address is used. As the Apple IIGS Debugger traces the code, it disassembles it and lists the results in the Disassembly subdisplay. The currently executing instruction is highlighted. Trace mode is described in the section "Single-Step and Trace Modes" in Chapter 1.
*address*S	Enter single-step mode at address *address*. If omit *address*, the current K/PC address is used. As the Apple IIGS Debugger steps through the code, it disassembles it and lists the results in the Disassembly subdisplay. The currently executing instruction is highlighted. Single-step mode is described in the section "Running Your Program" in this chapter.
ASM	Clear the Disassembly subdisplay to prepare for entering a sequence of instructions using the *address*:*instruction* command.
address:*instruction*	This command causes the debugger to assemble the instruction *instruction* and place the opcode and operand in memory at *address*. Simultaneously, the instruction is placed on the last line of the Disassembly subdisplay. Use the ASM command before using this command. See the section "Altering the Contents of Memory" in this chapter for more information on this command and a discussion of other commands to change values in memory.
Space bar *instruction*	Once you have used the *address*:*instruction* command, pressing the Space bar causes the next available address to be printed on the command line. Type in the next instruction to be assembled and press Return.

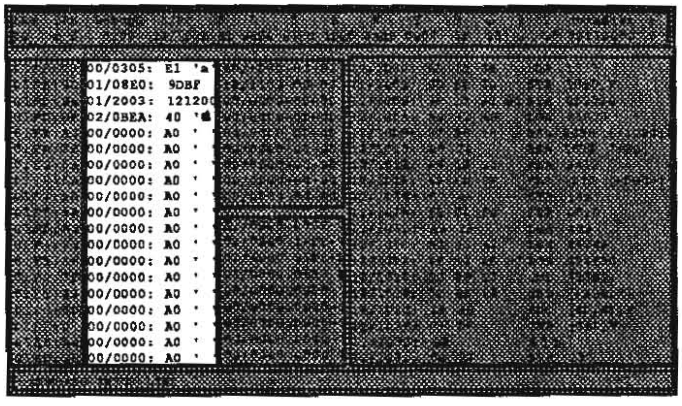
The Disassembly subdisplay shows the instructions following the current instruction in memory. If the current instruction jumps to or calls another routine, however, the address called or jumped to appears as the current instruction after the call or jump is executed. For example, assume the Disassembly subdisplay is as follows:

```
12/102E:C1 06        CMP (06),Y
12/1030:0A           ASL
12/1031:DC 89 23     JML 012389
12/1034:7C BE F2     JSL (F2BE,X)
12/1037:AD 15 18     LDA (1815)
```

When the JML instruction is executed, the current instruction becomes the one jumped to, and the display changes accordingly. It might look like this:

```
12/1030:0A          ASL
12/1031:DC 89 23    JML 012389
01/2389:9D 50 10    STA 1050,X
01/238C:9F 20 30 05 STA 053020
01/2390:A9 77 66    LDA #6677
```

RAM Subdisplay



The RAM subdisplay, located to the right of the Stack subdisplay, shows the contents of any 19 memory locations you select. As illustrated in Figure 3.2, each location can show either a single hexadecimal byte value and the equivalent ASCII character, a 2-byte value, or a 3-byte value. The 2- and 3-byte values are displayed as addresses; that is, the low byte (the one corresponding to the address in the left column) is displayed at the right. For example, if you place the value 1A in location 01/0100, 1B in 01/0101, and 1C in 01/0102, and display a 3-byte value at 01/0100, then the line of the RAM subdisplay looks like this:

```
01/0100: 1C1B1A
```

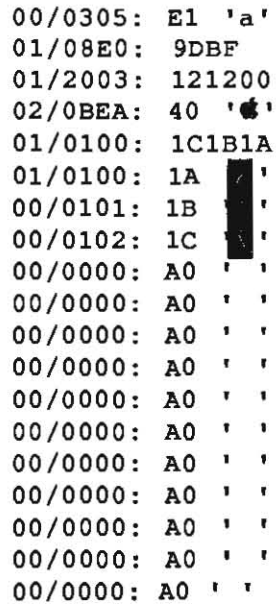


Figure 3.2 RAM Subdisplay

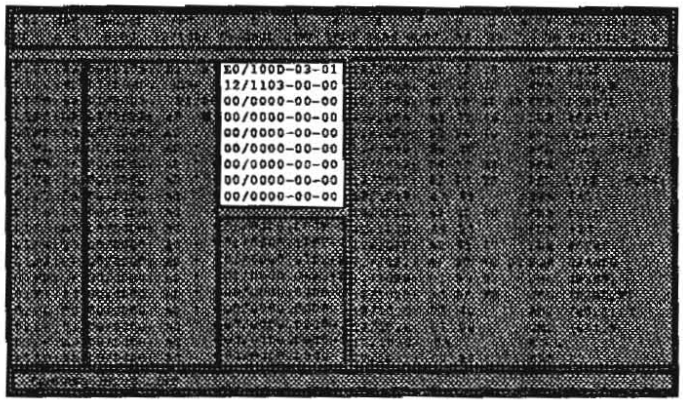
To modify the contents of the RAM subdisplay, type the following command on the command line and press Return:

```
MEM
```

The first line in the subdisplay will be highlighted. You can now use any of the following commands. (To enter specific values into memory locations, use the commands described in the section "Altering the Contents of Memory" later in this chapter.)

Command	Action
Return	Move to next address down.
↓	Move to next address down.
↑	Move to next address up.
address:	Display the contents of memory starting at *address*. You can include a slash (/) after the bank value or omit it when entering the address; either form works. If you do not include the bank number, the current value of the K register is used for the bank.
H	Display the contents of the cell as hexadecimal and ASCII.
P	Display the contents of the cell and next cell as a 2-byte address.
L	Display the contents of the cell and next two cells as a long (3-byte) address.
?	Display a help screen. Press any key except Esc to return to the RAM subdisplay.
Esc	Return to the command line.

Breakpoint Subdisplay



The Apple IIGS Debugger allows you to set from 0 to 17 breakpoints in your program. When you set a breakpoint, you indicate the location at which you want the program to suspend execution, and the number of times you want the breakpoint to be encountered before execution is interrupted. Each line of the Breakpoint subdisplay shows a breakpoint address (bank/location in bank), the number of times through the breakpoint before it triggers, and the number of times the program has actually passed through this breakpoint so far. For example, the first line in the Breakpoint subdisplay in Figure 1.1 is as follows:

```
E0/100D-03-01
```

This line indicates that the Apple IIGS Debugger is set to suspend execution of your program the third time it encounters the instruction located at address 100D in bank E0, and that it has already executed this instruction one time.

The default Master Display configuration provides nine lines for breakpoints. You can delete breakpoint lines, thus increasing the number of memory-protection lines, or you can delete memory-protection lines to increase the number of breakpoint lines. (The Memory-Protection subdisplay is directly below the Breakpoint subdisplay.)

To alter the contents of the Breakpoint subdisplay, type the following command and press Return:

```
BP
```

You can now use any of the following single-keystroke commands:

Command	Action
Return	Move to the next address down.
↓	Move to the next address down.
↑	Move to the next address up.
←	Move left to the address. Type in the starting address of the instruction at which you want the debugger to suspend execution. You can include a slash (/) after the bank value or omit it when entering the address; either form works. If you do not include the bank number, the current value of the K register is used for the bank.
→	Move right to the trigger value. Type in a 1-byte hexadecimal number to indicate the number of times the debugger should execute this instruction before suspending execution. If this value is 0, the debugger ignores the breakpoint. If this value is 1, the debugger stops each time the breakpoint is encountered. If this value is any number *n* from 2 to 255, the debugger stops every *n*th time the breakpoint is encountered.
Delete	Delete the currently highlighted breakpoint and increase the number of memory-protection lines by one.
?	Display a help screen. Press any key except Esc to return to the Breakpoint subdisplay.
Esc	Return to the command line.

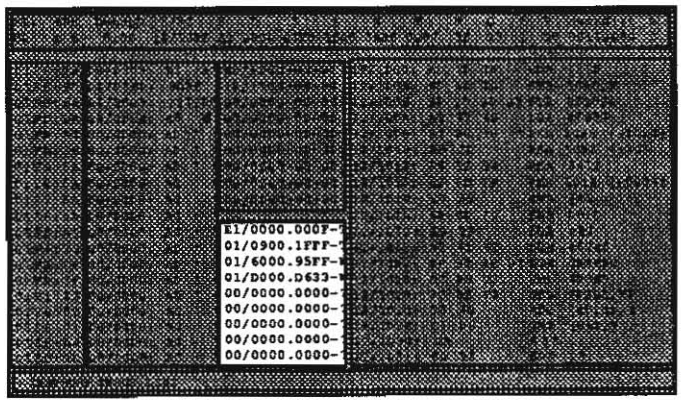
The following breakpoint commands can be entered from the Master Display command line. Press Return after typing each of these commands:

Command	Action
CLR	Zero all breakpoints to 00/0000-00-00. This command does *not* remove breakpoint lines from the screen. Use the Delete key as described in the preceding description of the BP command to remove breakpoint lines.
IN	Insert real breakpoints. The BRK register changes from o to i, and BRK instructions are inserted in memory at the addresses specified by the Breakpoint subdisplay. You must insert real breakpoints in the code in order to make the debugger suspend execution in real-time mode. Real and interpreted breakpoints are discussed in the section "Breakpoints" in Chapter 1.

Note: You cannot edit the Breakpoint subdisplay when real breakpoints are in. Use the OUT command before attempting to change breakpoints or trigger values.

OUT	Remove real breakpoints. The BRK register changes from i to o, and the BRK instructions that were inserted in memory by the IN command are replaced with interpreted breakpoints. Real and interpreted breakpoints are discussed in the section "Breakpoints" in Chapter 1.
DBRK	Return to the debugger when a BRK instruction that has not been set as a breakpoint is encountered while in real-time mode. A d appears next to the i or o in the BRK register display. DBRK is the default.
UBRK	Exit to the Monitor when a BRK instruction that has not been set as a breakpoint is encountered while your program is running in real-time mode. A u appears next to the i or o in the BRK register display.

Memory-Protection Subdisplay



The Apple IIGS Debugger allows you to specify address ranges that are protected during execution in trace or single-step modes. Each address range you have protected is shown in the Memory-Protection subdisplay, followed by a code that indicates the type of protection set, as follows:

Code	Meaning

`T` Code-trace range. All code outside this range is executed in single-step or trace mode. When your code executes a `JSL` to this memory-protection address range, the code inside this range is executed in real time. When the matching `RTL` is encountered, execution returns to single-step or trace mode. While the code inside a code-trace address range is being executed, the line in the Memory-Protection subdisplay specifying that range is highlighted on the screen.

`W` Code-window range. If one or more code-window address ranges are specified, code is executed only if it is inside one of the code-window ranges. Any time the program counter (K/PC) equals an address not in any of the code-window address ranges, execution stops. (If you don't specify any code-window address ranges, code can be executed at any address.) Execution is in trace or single-step mode unless a `JSL` is made to a code-trace address range that is completely contained in a code-window address range, in which case execution is in real time until the matching `RTL` is encountered. While the code inside a code-window address range is being executed, the line in the Memory-Protection subdisplay specifying that range is highlighted on the screen.

The default Master Display configuration provides nine lines for memory-protection ranges. When you start the Apple IIGS Debugger, the first memory-protection line is set to `E1/0000-000F T`; this code-trace range runs Apple IIGS tool calls in real-time mode. You can delete breakpoint lines, thus increasing the number of memory-protection lines, or you can delete memory-protection lines to increase the number of breakpoint lines. See the section "Configuring the Master Display" later in this chapter for more information on customizing the Master Display.

To alter the contents of the Memory-Protection subdisplay, type the following command on the command line and press Return:

```
MP
```

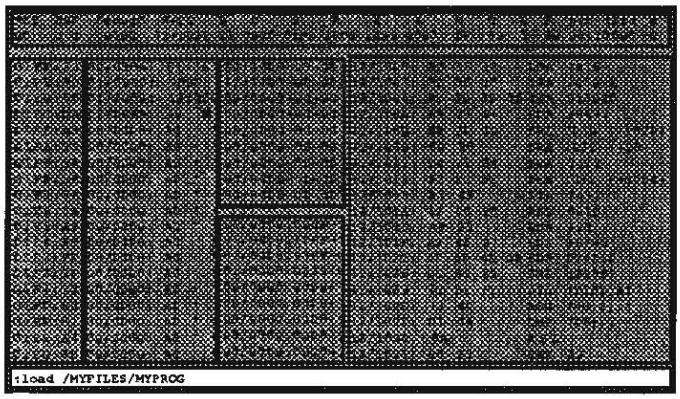
You can now use any of the following keypress commands:

Command	Action
Return	Move to the next address down.
↓	Move to the next address down.
↑	Move to the next address up.
←	Move left to the starting address. Type in the starting address of the code-trace or code-window range. You do not have to type a slash (/) after the bank value. If you do not include the bank number, the current value of the K register is used for the bank.
→	Move right to the ending address. Type in the ending address of the code-trace or code-window range. Do not include a bank value; the bank must be the same as that of the starting address.
T	Set this line as a code-trace range. You must enter either `T` or `W` for every memory-protection range.
W	Set this line as a code-window range. You must enter either `T` or `W` for every memory-protection range.
Delete	Delete the current memory-protection line and increase the number of breakpoint lines by one.
?	Display a help screen. Press any key except Esc to return to the Memory-Protection subdisplay.
Esc	Return to the command line.

For example, to enter a new code-window range—from 01/1220 to 01/12E5—on the second line of the Memory-Protection subdisplay, use the following sequence of commands:

Command	Meaning
`MP` Return	Begin editing the Memory-Protection subdisplay.
↓	Move down to the second address.
`11220`	Type in the starting address.
→	Move right to the ending address.
`12E5`	Type in the ending address.
W	Set this line as a code-window range.
Esc	Return to the command line.

Command Line



Many of the Apple IIGS Debugger's functions are executed by typing a command while in the Master Display. Commands are shown on the command line as you type them. Press Return to execute the command.

Several editing functions are available while entering commands, as follows:

Keystroke	Action
Control-E	Toggle between insert and replace modes. In insert mode, new characters are inserted between characters on the line, pushing the remaining characters to the right to make room. In replace mode, new characters replace the characters that the cursor is on.
←	Move the cursor one character to the left.
→	Move the cursor one character to the right.
Control-D	Delete the character to the left of the cursor.
Delete	Delete the character to the left of the cursor.
Control-F	Delete the character that the cursor is on.
Control-Y	Delete from the cursor position to the end of the line.
Control-X	Delete the entire line.
Esc	Delete the entire line.
Control-Z	Restore the last command typed.
Return	Execute the command that you typed on the command line. The entire line is sent to the command interpreter; the line is *not* truncated at the cursor position.

Altering the Contents of Memory

To alter the contents of a memory location, whether displayed in the RAM subdisplay or not, type one of the following commands on the command line and press Return. You can include a slash (/) after the bank value or omit it when entering an address; either form works. If you do not include the bank number, the current value of the K register is used for the bank. If you press the Space bar instead of typing an address, the next available

address is printed on the command line, followed by a colon (:). You can then type in the value you want to enter at that address and press Return.

Command	Action

address:value	Place the hexadecimal value *value* in memory starting at *address*. To enter a value of more than 1 byte, separate the values with spaces and enter the value that goes in the lowest address first. For example, to place the value $A0 at 01/04ED and the value $A1 at 01/04EE, you can use any of the following commands:
	`104ED:A0 A1`
	`0104ED:A0 A1`
	`1/04ED:A0 A1`
	`01/04ED:A0 A1`
	In addition, if the K register is already set to 01, you can use the following command:
	`04ED:A0 A1`

address:"string	Place values corresponding to *string* , with the high bit of each byte set, in memory starting at *address*. For example, the following command places the value $E1 at 01/04ED and the value $C1 at 01/04EE:
	`104ED:"aA`
	To include in a string one of the characters used in commands, precede the character with an exclamation mark (!). For example, to put the string `a"A` into memory at 0104ED, placing the value $E1 at 01/04ED, the value $A2 at 01/04EE, and the value $C1 at 01/04EF, use the following command:
	`104ED:"a!"A`

address:'string	Place values corresponding to *string* with the high bit of each byte cleared in memory at *address*. For example, the following command places the value $61 at 01/04ED and the value $41 at 01/04EE:
	`104ED:'aA`
	To include in a string one of the characters used in commands, precede the character with an exclamation mark (!). For example, to put the string `a"A` into memory at 0104ED, placing the value $61 at 01/04ED, the value $22 at 01/04EE, and the value $41 at 01/04EF, use the following command:
	`104ED:'a!"A`

address:instruction Assemble *instruction* and place the opcode and operand in memory starting at *address*. Simultaneously, the instruction is placed on the last line of the Disassembly subdisplay. For example, the following command places the value $A0 (the LDY immediate-address opcode) at 01/04ED and the value $A1 at 01/04EE:

```
104ED:LDY #A1
```

Enter accumulator-mode expressions like implied-mode expressions: for example, enter ROL rather than ROL A. Branch instructions take the absolute address to branch to, not an offset. If you enter a 1-byte immediate-mode operand, a second byte is *not* automatically inserted: for example, if you enter LDA #FF, the debugger places A9FF in memory, never A9FF00.

You can combine values and strings in one command. To do so, separate values with spaces and include trailing delimiters for strings (that is, if the string begins with a single quotation mark (') , end it with a single quotation mark; if it begins with a double quotation mark ("), end it with a double quotation mark). For example, the following command places the values $A0 $A1 $C1 $F0 $F0 $EC $E5 $20 $49 $49 in memory starting at address 01/04ED:

```
01/04ED:A0 A1 "Apple"' II'
```

Hexadecimal-Decimal Conversion

The Apple IIGS Debugger can convert hexadecimal numbers to decimal and *vice versa*. To convert a number, type one of the following commands on the command line and press Return.

Command	Action
value=	Convert *value* from hexadecimal to decimal. This command is identical to the $*value* command.
$*value*=	Convert *value* from hexadecimal to decimal. This command is identical to the *value* command.
+*value*=	Convert *value* from decimal to hexadecimal.
–*value*=	Convert *value* from decimal to hexadecimal. A negative decimal value is converted to a 2-byte two's complement hexadecimal equivalent; for example, –10 = $FFF6. (Note that if you put in $FFF6, you get 65526, not –10.)

Configuring the Master Display

You can configure the Apple IIGS Debugger Master Display to meet your needs by adusting the relative position of the stack pointer in the Stack subdisplay, the position of the current line in the Disassembly subdisplay, and the numbers of memory-protection lines and breakpoint lines. You can also select the slot used to send information to the printer.

To adjust the positions of the stack pointer and current-instruction line and to set the printer slot, type the following command on the command line and press Return:

```
SET
```

The following prompt appears on the command line:

```
Use arrow & number keys to set stack/disassembly bars &
printer slot = 1
```

Type any number from 1 to 7 to set the slot that the debugger will use to send data to the printer. You can also use the arrow keys to adjust the display. When you are done, press the Esc key to put into effect the changes and to clear the command line. The actions of the arrow keys are as follows:

Command	Action
←	Move the stack pointer up one line. All of the stack subdisplay lines move up one line, so that the highlighted line that indicates the current position of the stack pointer is one line higher in the display. You can now see the contents of one additional byte on the stack below (that is, with a lower address than) the stack pointer, and of one less byte above the stack pointer.
→	Move the stack pointer down one line. All of the stack subdisplay lines move down one line, so that the highlighted line that indicates the current position of the stack pointer is one line lower in the display. You can now see the contents of one additional byte on the stack above the stack pointer, and of one less byte below the stack pointer.
↑	Move the current instruction up one line. When you type SET and press Return, any disassembled code on the screen is cleared and an inverse-video bar appears at the location at which the current instruction would appear in the display. Each time you press the Up Arrow key, the highlighted bar moves up one line. You can now see the disassembly of one additional instruction following the current instruction, and of one less instruction preceding the current instruction when you continue single-stepping or tracing code.
↓	Move the current instruction down one line. Each time you press the Down Arrow key, the highlighted bar moves down one line. You can now see the disassembly of one less instruction following the current instruction and of one additional instruction preceding the current instruction when you continue single-stepping or tracing code.

Press Esc to put into effect the changes you made and to clear the command line.

The Breakpoint and Memory-Protection subdisplays can each occupy from one to 17 lines in the Master Display, but the total of both displays is always 18 lines. In other words, to increase the number of breakpoint lines, you must delete a corresponding number of memory-protection lines, and vice versa. To enter the Memory-Protection subdisplay, type MP on the command line and press Return. Then to delete a memory-protection line, use

the arrow keys to highlight the line you want to eliminate, and press Delete. You can delete as many lines as you wish, except that at least one memory-protection line must remain on the screen.

To delete a breakpoint line, type BP on the command line and press Return to enter the Breakpoint subdisplay. Then use the arrow keys to highlight the line you want to eliminate, and press Delete. You can delete as many lines as you wish, except that at least one breakpoint line must remain on the screen.

> **Note:** You cannot edit the Breakpoint subdisplay when real breakpoints are in. (When real breakpoints are in, the character displayed below the B of BRK in the Register subdisplay is an i; when real breakpoints are out, this character is an o.) If real breakpoints are in, use the OUT command before attempting to edit or delete breakpoints.

Saving a Display Configuration

Once you have customized the Apple IIGS Debugger display to suit your needs, you can save the configuration to disk in a display-configuration file. The following information is saved in a display-configuration file:

Information Saved	Associated Command
the position of the stack pointer in the Stack subdisplay	SET
the position of the current instruction in the Disassembly subdisplay	SET
the slot number for the printer	SET
memory addresses to be displayed and type of display for each (hexadecimal, short address, or long address)	MEM
the number of memory-protection lines	MP
memory protection ranges and the type of range for each (code trace or code window)	MP
the number of breakpoint lines	BP
breakpoints, including the address and trigger value	BP

You can save as many configurations as you wish. To save and restore display configurations, type the following commands on the command line, and press Return:

CSAVE *pathname* This command saves the current display configuration on disk to the file specified by *pathname*. Include the prefix for the file if you want to save it to a subdirectory other than the current ProDOS 16 system subdirectory. For example, to save the current configuration to the file `CONFIG.STORE` in the directory `/PROGRAMS/DEBUG/`, use the following command:

```
CSAVE /PROGRAMS/DEBUG/CONFIG.STORE
```

CLOAD *pathname* This command restores a previously saved display configuration from the disk file specified by *pathname*. Include the prefix for the file you want to use if the pathname is different from the current ProDOS 16 system subdirectory.

Command-Line Commands

This section lists all of the commands available from the Master Display command line. Most of these commands are described in detail elsewhere in this chapter, but they are included here for your convenience. You can include a slash (/) after the bank value or omit it when entering an address; either form works. If you do not include the bank number, the current value of the K register is used for the bank. Press Return after each command-line command.

?	Display a help screen.
address:	Display 368 contiguous bytes of memory starting at *address*.
address:*instruction*	Assemble *instruction* and place the opcode and operand in memory starting at *address*. Simultaneously, the instruction is placed on the last line of the Disassembly subdisplay.
address:'*string*	Place values corresponding to *string*, with the high bit of each byte cleared, in memory at *address*.
address:"*string*	Place values corresponding to *string* , with the high bit of each byte set, in memory starting at *address*.
address:*value*	Place the hexadecimal value *value* in memory starting at *address*. To enter a value of more than one byte, enter the byte that goes in the highest address first.
register=*value*	Set the register specified by *register* to the value specified by *value*. This command is case sensitive.
value=	Convert *value* from hexadecimal to decimal. This command is identical to the $*value* command.
$*value*=	Convert *value* from hexadecimal to decimal. This command is identical to the *value* command.
+*value*=	Convert *value* from decimal to hexadecimal.

-value=	Convert *value* from decimal to hexadecimal. A negative decimal value is converted to a 2-byte two's complement hexadecimal equivalent.
Space bar	Write the next available address on the command line, followed by a colon. Use this command to get the next address after using any command starting with *address*:.
ASM	Clear the Disassembly subdisplay to prepare for entering a sequence of instructions using the *address:instruction* command.
CLOAD *pathname*	Restore a previously saved display configuration from the disk file specified by *pathname*.
CLR	Clear all breakpoints to 00/0000-00-00.
CSAVE *pathname*	Save the current display configuration on disk to the file specified by *pathname*.
D	Display the direct page.
DBRK	Return to the debugger when a BRK instruction that has not been set as a breakpoint is encountered while your program is running in real-time mode.
e	Toggle the e flag: if it's set to 1, change it to 0; if it's set to 0, change it to 1. This command is case sensitive.
IN	Insert real breakpoints.
*address*J	JML directly to code at address *address*. If you omit *address*, the current K/PC address is used.
KEY=*keynum*	Each bit of the binary number represented by the hexadecimal number *keynum* specifies one key to be used as a keystroke modifier; set that bit to 1 to make that key a keystroke modifier. The bit assignments are described in the section "Register Subdisplay" in this chapter.
LOAD *pathname*	Load the program to debug.
m	Toggle the m flag: if it's set to 1, change it to 0; if it's set to 0, change it to 1. This command works only if e = 0. This command is case sensitive.
MON	Exit from the debugger into the Monitor. Press Control-Y and then press Return to return to the debugger.
OFF	Turn off the Master Display and show the display of your program.
ON	Turn off your program's display and turn on the Master Display.

OUT	Remove real breakpoints.
P	Print the current text screen. You can use this command with the Master Display on to print the current Master Display, or with the Master Display off to print your program's display (80-column text only). You can also print the Memory Display or help screens with this command.
PREFIX *n pathname*	Change ProDOS 16 prefix *n* to *pathname*. This command has the same effect as the APW Shell PREFIX command; see the sections "Standard Prefixes" and "Command Descriptions" in Chapter 3 of the *Apple IIGS Programmer's Workshop Reference* for details. If you omit *n*, prefix 0 is changed.

Important: When you quit the debugger, these prefix assignments are retained. If you change prefixes 2, 4, 5, or 6, APW may not be able to find the files it needs to function. See the section "Standard Prefixes" in Chapter 3 of the *Apple IIGS Programmer's Workshop Reference* for a discussion of the prefixes used by APW.

Q	Exit the debugger. This command terminates the Apple IIGS Debugger, unlike the MON command, which allows you to return from the Monitor to the debugger. If you called the debugger from the APW Shell, Q returns you to the shell.
QUIT	Exit the debugger. This is an alias for Q.
*address*S	Enter single-step mode at address *address*. If you omit *address*, the current setting of the K/PC register is used.
SET	Adjust the positions of the stack pointer and current-instruction line and set the printer slot.
*address*T	Enter trace mode at address *address*. If you omit *address*, the current setting of the K/PC register is used.
UBRK	Exit to the Monitor when a BRK instruction that has not been set as a breakpoint is encountered while in real-time mode.
V	Display the current version number and copyright of the Apple IIGS Debugger.
x	Toggle the x flag: if it's set to 1, change it to 0; if it's set to 0, change it to 1. This command works only if e = 0. This command is case sensitive.
*address*X	JSL directly to code at address *address*. If you omit *address*, the current setting of the K/PC register is used. If you omit *address*, the X must be uppercase.

Chapter 4

Desk Accessories

Two desk accessories are provided with the Apple IIGS Debugger: Loader Dumper and Memory Mangler. Both are described in this chapter.

Loader Dumper

Loader Dumper is a desk accessory included on your Apple IIGS Programmer's Workshop system disk. You can use Loader Dumper together with the Apple IIGS Debugger and the Memory Mangler desk accessory as an aid to debugging relocatable and dynamic code. Loader Dumper lets you see where in memory the System Loader has loaded each segment of your program and gives you information about the various tables and variables that the loader uses. Memory Mangler is described in the section "Memory Mangler" later in this chapter. The System Loader is described in the *Apple IIGS ProDOS 16 Reference.*

To get the Desk Accessories menu, press Apple-Contol-Esc. When you select Loader Dumper from the Desk Accessories menu, the following menu appears on the screen:

```
1. Dump Memory Segment Table
2. Dump Pathname Table
3. Dump Jump Table
4. Dump Loader Globals
5. Dump ProDOS Packets
6. Dump File Buffer Variables
7. Get Load Segment Information
8. Dump Address

What do you want to dump ?
```

All of these selections are described in the following sections.

Dump Memory Segment Table

The memory segment table is a linked list, each entry of which describes a memory block known to the System Loader. Each memory block corresponds to a single load segment. Note that dynamic segments do not appear in the memory segment table when the program is initially loaded because they are not loaded into memory until the program needs them.

You can use the memory segment table to get the starting address of every segment currently in memory. One entry in the table is shown at a time; press Return to see the next entry. Press Esc to return to the Loader Dumper main menu. To see memory segment table information on one specific segment (instead of scrolling through the entire memory segment table), use selection 7, `Get Load Segment Information`.

Before using the memory segment table to get the starting addresses of segments, you must know the User ID and file number of the program in which you are interested. This information is available from selection 2, `Dump Pathname Table`. If you load a program with the Apple IIGS Debugger, the User ID is also displayed in the A register immediately after the program is loaded.

The starting address of the segment is shown in the memory segment table in parentheses after the `Memory Handle` field. For example, the starting address in memory of load segment 1 of load file 1 for UserID $A001 (as shown in Figure 4.1) is $1188D3.

The type of segment is shown in parentheses after the `Load Segment Kind` field. Segment types are described in Chapter 7 of the *Apple IIGS Programmer's Workshop* .

```
Memory Segment Table
$E11854 (1188BF)
------------------------
Next Handle = $E118B8 (11FD8B)
Prev Handle = $000000
------------------------
UserID              = $A001
Memory Handle       = $E11840 (1188D3)
Load File Number    = $0001
Load Segment Number = $0001
Load Segment Kind   = $2000
                            (Position Independent)

Press RETURN to continue

$E118B8 (11FD8B)
------------------------
Next Handle = $E118E0 (11FD6F)
Prev Handle = $E11854 (1188BF)
------------------------
UserID              = $5002
Memory Handle       = $E118A4 (020000)
Load File Number    = $0001
Load Segment Number = $0001
Load Segment Kind   = $0402 (Jump Table Segment)
                            (Reload)

Press RETURN to continue
```

Figure 4.1. Memory Segment Table Output

If your program has unloaded a memory block (that is, made it purgeable), you can use the memory segment table to find out if it has been purged. To do so, check the address in parentheses after the memory handle: if the address is 000000, the block has been purged.

Dump Pathname Table

The pathname table provides a cross-reference between file numbers, file pathnames, and User IDs. The pathname table is a linked list of individual pathname entries. One entry in the table is shown at a time; press Return to see the next entry. Press Esc to return to the Loader Dumper main menu.

You can use the pathname table to get the User ID and file number of every program in memory. You need this information to use the memory segment table to find the starting memory address of a segment. The pathname table also gives you the starting address and size of the direct-page/stack space requested by the loader for each program. (The loader requests a direct-page/stack space only if you include a direct-page/stack segment in your program; otherwise, ProDOS either assigns the direct-page/stack space as a default or your program can request one through the Memory Manager.)

The Pathname Table display of the Loader Dumper is illustrated in Figure 4.2.

```
Pathname Table
$E11890 (118527)
------------------------
Next Handle = $E118F4 (11F9F6)
Prev Handle = $000000
------------------------
UserID                  = $A001
File Number             = $0001
File Date               = $AD11
File Time               = $0B37
Direct Page/Stack Addr  = $0000
Direct Page/Stack Size  = $0000
File Pathname           = /APW/SYSTEM/SYSTEM.SETUP/TOOL.SETUP

Press RETURN to continue

$E118F4 (11F9F6)
------------------------
Next Handle = $E11930 (11FAD9)
Prev Handle = $E11890 (118527)
------------------------
UserID                  = $5002
File Number             = $0001
File Date               = $AD2C
File Time               = $0930
Direct Page/Stack Addr  = $0000
Direct Page/Stack Size  = $0000
File Pathname           = /APW/SYSTEM/DESK.ACCS/MANGLER.DA

Press RETURN to continue
```

Figure 4.2. Pathname Table Output

Dump Jump Table

All references to dynamic segments are made through the jump table. The jump table in memory consists of the jump table directory and one or more jump table segments. The jump table directory is a linked list, each entry of which points to a single jump table segment encountered by the loader. The Loader Dumper displays each jump table directory entry followed by the jump table segment to which the entry points. Each jump table segment contains one entry for each reference to a dynamic segment in the program.

One entry in the table is shown at a time; press Return to see the next entry. Press Esc to jump to the next directory entry. If you are at the last directory entry, Esc returns you to the Loader Dumper main menu.

You can use the jump table to determine whether a dynamic segment has been loaded into memory; if it has been loaded, you can use the memory segment table to find the starting address of the segment in memory. A sample jump table display is shown in Figure 4.3. The first entry in Figure 4.3 is for a dynamic segment that has been loaded into memory. You can tell that the segment has been loaded because the jump table segment entry is in its loaded state: it ends in a `JML` to the referenced subroutine. The operand of the `JML` statement is the location in memory of the subroutine being referenced (if there is more than one routine or entry point in the segment, there will be more than one jump table entry for that segment). The number in parentheses after the `Handle to Segment` field shows the location in memory of the jump table segment itself.

The second entry in Figure 4.3 is for a dynamic segment that has not been loaded into memory. The jump table segment entry ends in a `JSL` to the System Loader's Jump Table Load function.

Before using the jump table to get information about dynamic segments, you must know the User ID and file number of the program in which you are interested. This information is available from selection 2, `Dump Pathname Table`.

The jump table and jump table segments are described in Chapter 16, "System Loader," of the *Apple IIGS ProDOS 16 Reference.*

```
Jump Table

$E11AC0 (11FA2D)
------------------------
Next Handle = $E11818 (11E2AA)
Prev Handle = $E11840 (11FA3B)
------------------------
UserID                    = $1007
Handle to Segment         = $E11A34 (010A6B)

UserID                    = $1007
Load File Number          = $0001
Load Segment Number       = $0002
Load Segment Offset       = $00000000
Jump to Loader/Function   = JML 010A85

Press RETURN to continue

$E11818 (11E2AA)
------------------------
Next Handle = $000000
Prev Handle = $E11AC0 (11FA2D)
------------------------
UserID                    = $1008
Handle to Segment         = $E118E0 (010B13)

UserID                    = $1008
Load File Number          = $0001
Load Segment Number       = $0002
Load Segment Offset       = $00000000
Jump to Loader/Function   = JSL 11FF10

Press RETURN to continue
```

Figure 4.3. Jump Table Output

Dump Loader Globals

The Loader Globals display shows the values of some loader variables and some statistics associated with the last load operation performed. This table was included primarily for use by Apple engineers when they were debugging the System Loader. If you have trouble using the loader, or believe you have found a bug in the loader, copy down the information in the `Last Function`, `Total Errors`, and `Error Addresses` fields before calling technical support, or include this information in your bug report.

The `BUSY` field shows the status of the loader's busy flag: the flag is 1 if any loader function is currently being executed. Every loader function checks the busy flag before executing.

The fields beginning with a lowercase `n` (`nLCONST` or `nRELOC`—for example, see Figure 4.4) show the number of certain kinds of records, the number of segments, and the number

of bytes loaded. See Chapter 7, "File Formats," of the *Apple IIGS Programmer's Workshop Reference* for a description of segment records.

The `tStart` and `tEnd` fields show the settings of the clock cycle counter at the beginning and at the end of the load for an Initial Load or Restart function call.

Press Return or Esc to return to the Loader Dumper menu.

```
Loader Globals
--------------
$01E700 BUSY            = $0000
$01E702 SEGTBL          = $E11854 (1188BF)
$01E706 JMPTBL          = $000000
$01E70A PATHTBL         = $E11890 (118527)
$01E70E USERID          = $1001
$01E72C Last Function   = $0006
$01E710 Total Errors    = $0000
$01E712 Error Addresses = $0000 E164 D400 0000 0000
$01E72E nLCONST         = $0012
$01E730 nRELOC          = $0000
$01E732 nINTERSEG       = $0000
$01E734 nDS             = $0011
$01E736 ncRELOC         = $1262
$01E738 ncINTERSEG      = $0000
$01E73A nSegments       = $0001
$01E73C nBytes          = $0000A1CD
$01E740 tStart          = $56080126
$01E744 tEnd            = $56080133

Press RETURN to continue
```

Figure 4.4. Loader Globals Output

Dump ProDOS Packets

This display shows the ProDOS 16 calls, including parameter blocks, used most recently by the loader. This information is primarily for use by Apple engineers in debugging the loader and ProDOS 16. The ProDOS Packets display is illustrated in Figure 4.5.

ProDOS 16 calls are described in the *Apple IIGS ProDOS 16 Reference.*

```
ProDOS Packets
--------------
$01E7B4 PGetInfo   = $00000240 00E3 00B3 000000100 0002 AD43 0007 AD43 0C02
                   = $00000093
$01E7CE POpen      = $0001 00000240 00009788
$01E7D8 PRead      = $0001 00114527 00004000 0000236B
$01E7E6 PClose     = $0001
$01E7E8 PGetEOF    = $0001 00000000
$01E7EE PGetMark   = $0001 00010000
$01E7F4 PSetMark   = $0001 0000A2BC
$01E7FA PGetPrefix = $0000 000000000

Press RETURN to continue
```

Figure 4.5. ProDOS Packets Output

Dump File Buffer Variables

The file buffer is used by the System Loader to buffer data being loaded from disk into memory. One of two file buffer sizes is used, depending on the amount of memory currently available. If more than 5 banks of memory are available, the larger buffer is used; otherwise, the smaller buffer is used. The default values for these buffers are currently $4000 bytes and $400 bytes, respectively.

If you wish, you can change these buffer sizes by using the Monitor or the debugger to alter the contents of memory at the addresses indicated at the left of the `Max_File_Buff1` and `Max_File_Buff1` fields (see Figure 4.6). You can experiment with different buffer sizes to see if it speeds up the load or decreases the amount of memory used by the loader.

The `File_Pt` field shows the next location to be read from the file buffer. The `File_EOB` field shows the location of the last valid data currently in the buffer. These fields are equal after a load is complete and any time during a load that the buffer is full, indicating that all the data in the buffer has been read. In this case, the next read operation on the buffer will cause the buffer to be refreshed.

The `File_Mark` field shows the last location read from within the ProDOS file being loaded. The `Header_Mark` field shows the location of the beginning of the next segment header in the file being loaded.

You can use the File Buffer Variables display to monitor the progress of a load.

Press Return or Esc to return to the Loader Dumper main menu.

```
File Buffer Variables
---------------------
$01E79E File_Buff       = 00114527
$01E7A2 File_Buff_Size  = 4000
$01E7A4 Max_File_Buff1  = 0400
$01E7A6 Max_File_Buff2  = 4000
$01E7A8 File_Pt         = 236B
$01E7AA File_EOB        = 236B
$01E7AC File_Mark       = 00010000
$01E7B0 Header_Mark     = 00012400

Press RETURN to continue
```

Figure 4.6. File Buffer Variables Output

Dump Load Segment Information

The Load Segment Information selection provides the same information as the Memory Segment Table selection, except that information is provided on only the segment you specify. Figure 4.7 illustrates the Load Segment Information selection; the characters shown in boldface are the ones you type in.

Before using the Load Segment Information selection to get information about static segments, you must know the User ID and file number of the program in which you are interested. This information is available from selection 2, `Dump Pathname Table`.

Press Return to be prompted for the next load segment. Press Esc to return to the Loader Dumper main menu.

```
Key in UserID of Load Segment - 5002
Key in File Number of Load Segment - 1
Key in Segment Number of Load Segment - 1
------------------------
UserID               = $5002
Memory Handle        = $E118A4 (020000)
Load File Number     = $0001
Load Segment Number  = $0001
Load Segment Kind    = $0000
------------------------

Press RETURN to continue


Key in UserID of Load Segment -
```

Figure 4.7. Load Segment Information Output

Dump Address

The Dump Address selection writes out 16 hexademical values stored in memory starting at the address you specify. Press Return to be prompted for the next starting address. To get

the next 16 addresses, press Return again in response to the prompt instead of entering a new address.

Press Esc to return to the Loader Dumper main menu.

Figure 4.8 illustrates the use of the Dump Address selection; characters you type in are shown in boldface.

```
What address do you want to dump ? 020000 [press Return]
00020000:07 4D 61 6E 67 6C 65 72 4F 01 02 00 10 00 02 00
What address do you want to dump ? [press Return]
00020010:6B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
What address do you want to dump ? [press Esc]
```

Figure 4.8. Dump Address Output

Memory Mangler

Memory Mangler is a desk accessory included on your Apple IIGS Programmer's Workshop system disk. Memory Mangler lets you execute a variety of Memory Manager routines and provides lists of the memory blocks that are in use, purged, and disposed by the Memory Manager. You can use Memory Mangler together with the Apple IIGS Debugger and the Loader Dumper desk accessory as an aid to debugging relocatable and dynamic code. The System Loader is described in the *Apple IIGS ProDOS 16 Reference.*

To get the Desk Accessories menu, press Apple-Contol-Esc. When you select Mangler from the Desk Accessories menu, a percent sign (%) prompt appears at the left edge of the screen. To get a list of commands available, type `COMMANDS` and press Return. You can stop the listing by pressing any key (the commands list very quickly; you'll have to be fast to stop the listing before the first commands scroll off the screen). Press any key to resume the listing. Most of the commands listed are Memory Manager commands; Memory Mangler lets you execute any of the Memory Manager commands listed. In addition, you can use any of the following commands:

- `LIST`
- `MON`
- `PRINT`
- `QUIT`

These commands are described in the following sections, followed by a general description of the use of Memory Manager commands.

LIST

The `LIST` command has three options, as follows:

`LIST U` List memory handles being used. A memory handle is a pointer to the location in memory where the address of the memory block is stored. Each handle has several attributes associated with it (see below).

`LIST P` List memory handles that have been *purged*. When a memory handle is purged, the handle points to $000000 (that is, that handle is no longer assigned to a memory block), but the handle is still associated with a specific User ID and it retains its attributes. The Memory Manager can assign a new memory block to that handle at any time, but cannot use the handle for any other purpose.

`LIST F` List memory handles that have been *disposed*. When a memory handle is disposed, the handle has no attributes or User ID assigned to it; it becomes simply a region of memory reserved for use by the Memory Manager. The Memory Manager can assign a disposed handle to any User ID it chooses at any time. The Memory Manager can also create additional free memory handles when it needs them.

When you type any of these commands and press Return, a table of values is displayed on the screen. Press any key to stop the listing; after stopping the listing, press any key to resume the listing. When the listing is complete, the percent sign prompt reappears.

The columns in the memory-block listing are as follows:

Line Number	Handle	Location	Attributes	UserID	Length	Previous Handle	Next Handle

Line Number

This column displays sequential line numbers for the list of memory handles. These line numbers are for your convenience only; they are not used by the Memory Manager.

Handle

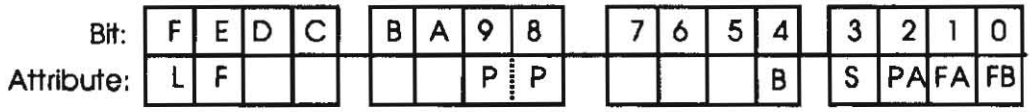
The handle to the memory block. Handles are used in most Memory Manager calls that refer to specific memory blocks.

Location

The actual address of the memory block referred to by the handle. You can use the Monitor or the debugger to examine the contents of memory at this address.

Attributes

The attributes of this memory block. Figure 4.9 illustrates the attributes word. Memory-block attributes are described in Chapter 4 of of the *Apple IIGS Toolbox Reference: Volume 1*.



Key:

FB	fixed bank	If 1, the block must start in a specified bank.
FA	fixed address	If 1, the block must start at a specified address.
PA	page-aligned	If 1, the block must be aligned to a page boundary.
S	special-memory usable	If 1, the block may not be allocated in special memory: banks 0 and 1 and the video screens.
B	bank boundary	If 1, the block may not cross a bank boundary.
P	purge level	0 to 3: if 0, the block cannot be purged; 3 means most purgeable.
F	fixed	If 1, the block is nonmovable.
L	locked	If 1, the block is locked: temporarily nonmovable and unpurgeable.

Figure 4.9. Memory-Block Attributes

UserID

The User ID of the program that owns this block of memory. You can use the Loader Dumper desk accessory to list the pathname table, which provides a cross-reference between file pathnames and User IDs.

Length

The length of the memory block in bytes.

Previous Handle

The handle of the memory block that precedes this block in the list.

Next Handle

The handle of the memory block that follows this block in the list.

MON

When you type MON and press Return, you enter the Monitor. You can use the Monitor to examine the contents of memory, to change the contents of memory, to examine the contents of hardware registers, and to perform a variety of other functions useful when debugging programs.

To return to the Memory Mangler, press Control-Y, and then press Return.

The Monitor is described in the *Apple IIGS Firmware Reference*.

PRINT

The `PRINT` command is identical to the `LIST` command, except that the data is sent to slot (or port) 1. If you have a printer attached to that slot, the table should be printed. You can use the `U`, `P`, and `F` options with the `PRINT` command just as with the `LIST` command.

QUIT

When you type `QUIT` and press Return, you quit the Memory Mangler and return to the Desk Accessories menu.

Memory Manager Commands

When you type `COMMANDS` and press Return, a list of all the Memory Manager commands the Memory Mangler recognizes is written on the screen. Each command includes the parameters it takes (if any). For example, the line for the `NewHandle` command looks like this:

```
NEWHANDLE size ownerid attributes location
```

You can use the Loader Dumper desk accessory to get the User IDs of programs and the memory handles of program segments. You can use the `LIST` and `PRINT` commands of the Memory Mangler to get the purge level and attributes of blocks of memory. You can use the `LIST P` and `PRINT P` commands to find out which blocks of memory are purgeable, and the `LIST F` and `PRINT F` commands to find out which blocks of memory are free.

Memory Manager calls are described in Chapter 4 of of the *Apple IIGS Toolbox Reference: Volume 1*.

Appendix

Command Summary

This section lists all of the commands that you can use in the Apple IIGS Debugger.

Keystroke Modifier

If the command filter is in effect, you must hold down a keystroke-modifier key in order to pass commands on to the debugger. The keystroke-modifier setting is shown in the KEY field of the Register subdisplay. To set the keystoke modifier, use the following command:

`KEY=`*keynum* Each bit of the binary number represented by the hexadecimal number *keynum* specifies one key to be used as a keystroke modifier; set that bit to 1 to make that key a keystroke modifier. The bits are assigned as follows:

Bit:	7	6	5	4	3	2	1	0
Key:	A	O		K	R	CL	C	S
Hex Value:	80	40	20	10	08	04	02	01

The abbreviations in this diagram refer to the following keys:

Abbreviation	Key
S	Shift
C	Control
CL	Caps Lock
R	Repeat: hold the key down until it repeats
K	Any key on an external keypad (not the keypad on the Apple IIGS keyboard)
O	Option
A	⍝

Selecting a Display

Display	How to Select
Help Screen	From any display, type a question mark (?) and press Return.
Memory	From the Master Display, type the starting address of the memory block you wish to display, followed by a colon (:), and press Return.
Direct Page	From the Master Display, type D and press Return.
Application	To see the display generated by your application, type OFF on the Master Display command line and press Return, and then start your application as described in the section "Running Your Program" in this chapter. To change the display mode, use the following commands (*these commands work while in single step or trace modes only*—see the section "Single Step and Trace Modes" in this chapter for more information on these commands): 1 text page 1 2 text page 2 4 40-column screen 8 80-column screen T text mode F full-screen graphics M mixed text and graphics L low-resolution graphics H Hi-Res graphics D Double Hi-Res graphics B black-and-white (for Double-Hi-Res graphics) C color (for Double-Hi-Res graphics) S Super Hi-Res graphics
Monitor	To call the Monitor, type MON on the Master Display command line and press Return.
Master Display	In Direct-Page or Memory Display, press Esc. If your application is being displayed, type ON and press Return. From the Monitor, press Control-Y and press Return to return to the Master Display.

Note: If the command filter is in effect, you must hold down one or more keystroke-modifier keys in order to pass commands on to the debugger while your program is running. See the section "The Command Filter" in this chapter for more information on this function.

Editing the Master Display

Use the following commands to alter the contents or setup of the master display.

Display Configuration

SET	Set printer slot, adjust stack-pointer highlight and number of instructions below highlight in Disassembly subdisplay.
	← Move the stack pointer up one line.
	→ Move the stack pointer down one line.
	↑ Move the current instruction up one line.
	↓ Move the current instruction down one line.

Disassembly Subdisplay

*address*L	Disassemble the contents of memory starting at *address* and display the next 19 lines of code.
L	Disassemble the contents of memory starting at the current address and display the next 19 lines of code.
address:*instruction*	Assemble the instruction *instruction* and place the opcode and operand in memory at *address*. Simultaneously, the instruction is placed on the last line of the Disassembly subdisplay.
ASM	Clear the disassembly subdisplay to prepare for entering a sequence of instructions using the *address*:*instruction* command.

RAM subdisplay

Return	Move to next address down.
↓	Move to next address down.
↑	Move to next address up.
address:	Display the contents of memory starting at *address*.
H	Display the contents of the cell as hex and ASCII.
P	Display the contents of the cell and next cell as a 2-byte address.
L	Display the contents of the cell and next two cells as a long (3-byte) address.
?	Display a help screen. Press any key except Esc to return to the RAM subdisplay.
Esc	Return to the command line.

Breakpoints Subdisplay

Return	Move to the next address down.
↓	Move to the next address down.
↑	Move to the next address up.
←	Move left to the address. Type in the starting address of the instruction at which you want the debugger to suspend execution.
→	Move right to the trigger value. Type in a one-byte hexadecimal number to indicate the number of times the debugger should execute this instruction before suspending execution.
Delete	Delete the currently highlighted breakpoint and increase the number of memory protection lines by one.
?	Display a help screen. Press any key except Esc to return to the breakpoint subdisplay.
Esc	Return to command line.

The following breakpoint commands can be entered from the master-display command line. Press Return after typing in each command.

`CLR`	Zero all breakpoints to 00/0000-00-00.
`IN`	Insert real breakpoints. **Note:** You cannot edit the breakpoint subdisplay when real breakpoints are in.
`OUT`	Remove real breakpoints.
`DBRK`	Return to the debugger when a `BRK` instruction that has not been set as a breakpoint is encountered while in real-time mode.
`UBRK`	Exit to the monitor when a `BRK` instruction that has not been set as a breakpoint is encountered while in real-time mode.

Memory Protection Subdisplay

Return	Move to the next address down.
↓	Move to the next address down.
↑	Move to the next address up.

←	Move left to the starting address. Type in the starting address of the code-trace or code-window range.
→	Move right to the ending address. Type in the ending address of the code-trace or code-window range. Do not include a bank value; the bank must be the same as that of the starting address.
T	Set this line as a code-trace range.
W	Set this line as a code-window range.
Delete	Delete the current memory-protection line and increase the number of breakpoint lines by one.
?	Display a help screen. Press any key except Esc to return to the memory protection subdisplay.
Esc	Return to the command line.

Command Line Commands

These commands are used on or entered from the debugger command line. Press Return to execute these commands.

Command-Editing Commands

These commands are used for editing commands that you are typing on the command line.

Control-E	Toggle insert/replace mode.
←	Move the cursor one character to the left.
→	Move the cursor one character to the right.
Control-D Delete	Delete the character to the left of the cursor.
Control-F	Delete the character that the cursor is on.
Control-Y	Delete from the cursor position to the end of line.
Control-X Esc	Delete the entire line.
Control-Z	Restore the last command typed.
Return	Execute the command on the line. The entire line is sent to the command interpreter; the line is not truncated at the cursor position.

Setting Registers and Memory Values

`e`	Toggle the `e` flag: if it's set to 1, change it to 0; if it's set to 0, change it to 1.
`x`	Toggle the `x` flag: if it's set to 1, change it to 0; if it's set to 0, change it to 1. This command works only if e=0.
`m`	Toggle the `m` flag: if it's set to 1, change it to 0; if it's set to 0, change it to 1. This command works only if e=0.
register=value	Set the register specified by *register* to the value specified by *value*. The values for all registers are given as hexadecimal numbers, except for the processor-status bits, which can be either 1 or 0. Register names are case sensitive.
address:value	Place the hexadecimal value *value* in memory starting at *address*.
address:"string	Place values corresponding to *string* , with the high bit of each byte set, in memory starting at *address*.
address:'string	Place values corresponding to *string* with the high bit of each byte cleared in memory at *address*.
address:instruction	Assemble *instruction* and place the opcode and operand in memory starting at *address*.

Breakpoints

`CLR`	Zero all breakpoints to 00/0000-00-00.
`IN`	Insert real breakpoints. **Note:** You cannot edit the breakpoint subdisplay when real breakpoints are in.
`OUT`	Remove real breakpoints.

Hexadecimal—Decimal Conversion

value=	Convert *value* from hexadecimal to decimal. This command is identical to the *$value* command.
$value=	Convert *value* from hexadecimal to decimal. This command is identical to the *value* command.
+value=	Convert *value* from decimal to hexadecimal.
−value=	Convert *value* from decimal to hexadecimal. A negative decimal value is converted to a two-byte twos complement hexadecimal

equivalent. For example, –10 = $FFF6. (Note that, if you put in $FFF6, you get 65526, not –10.)

Saving Display Configurations

CSAVE *pathname*	Save the current display configuration on disk to the file specified by *pathname*.
CLOAD *pathname*	Restore a previously saved display configuration from the disk file specified by *pathname*.

Printing

P	Print the current text screen.

Loading and Running Your Program

LOAD *pathname*	Load the program to debug.
S	Enter single-step mode at the current instruction.
*address*S	Enter single-step mode at address *address*.
T	Enter trace mode at the current instruction
*address*T	Enter trace mode at address *address*.
G	JSL directly to code at the current K/PC address.
*address*X	JSL directly to code at address *address* . The debugger automatically turns off the Master Display before executing this command.
*address*J	JML directly to code at address *address* . If you omit *address*, then the current K/PC address is used. The debugger automatically turns off the Master Display before executing this command.

Other Command-Line Commands

KEY=*keynum*	Each bit of the binary number represented by the hexadecimal number *keynum* specifies one key to be used as a keystroke modifier; set that bit to 1 to make that key a keystroke modifier. The bits assignments are described in the section "Keystroke Modifier" in this appendix.

PREFIX *n pathname* Change ProDOS 16 prefix Prefix *n* to *pathname*.

V	Display the current version number and copyright of the Apple IIGS Debugger.
MON	Enter the Apple IIGS Monitor. From the Monitor, press Control-Y and then press Return to return to the debugger.
Q	Exit debugger.
QUIT	Exit debugger

Trace and Single-Step Mode Commands

These are single-keystroke commands: do not press Return after these keystrokes.

Esc	Terminate trace or single-step mode and return to the command-line.
Space	Single-step one instruction.
Return	Start continuous tracing.
R	Trace until the next RTS, RTI, or RTL.
J	If the current instruction (the next to be executed) is a JSL, execute in real time until an RTL or RTI. If the next instruction is not a JSL, the command is ignored.
X	If the current instruction (the next to be executed) is a JSL, execute in real time until the matching RTL or an RTI that returns to the following instruction. If the next instruction is not a JSL, the command is ignored.
↓	Skip the next instruction.
Q	Toggle the sound on or off. If the sound is on, the speaker beeps each time an instruction is executed.
1	Change the display to text page 1. Use this command when in 40-column text mode or mixed text and graphics mode.
2	Change the display to text page 2. Use this command when in 40-column text mode or mixed text and graphics mode.
4	Change the display to a 40-column screen. Use this command when in text mode.
8	Change the display to a 80-column screen. Use this command when in text mode.
T	Change the display to text mode.
F	Change the display to full-screen graphics mode.

M	Change the display to mixed text and graphics mode.
L	Change the display to low-resolution graphics mode.
H	Change the display to high-resolution graphics mode.
D	Change the display to double-high-resolution graphics mode.
S	Change the display to super-high-resolution graphics. This is the normal Apple IIGS display mode.
B	Change the display to black and white double-high-resolution graphics.
C	Change the display to color double high-resolution graphics.
←	Change to the slow trace rate.
→	Change to the fast trace rate.
♁	Pause the trace until the ♁ key is released.