



Apple® IIe Technical Reference Manual



Includes ROM Listings.

Copyright

This manual is copyrighted by Apple or by Apple's suppliers, with all rights reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of Apple. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

©Apple Computer, Inc., 1985
20525 Mariani Avenue
Cupertino, California 95014

Apple, the Apple logo, ProDOS, ProFile, and Disk II are trademarks of Apple Computer, Inc.

CP/M is a registered trademark of Digital Research, Inc.

SOFTCARD is a registered trademark of Microsoft Corporation.

Z-80 is a registered trademark of Zilog, Inc.

Z-Engine is a trademark of Advanced Logic Systems, Inc.

Simultaneously published in the United States and Canada.

Limited Warranty on Media and Replacement

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for Program coverage and details. In some countries the replacement period may be different; check with your authorized Apple dealer.

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

Even though Apple has tested the software and reviewed the documentation, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD "AS IS," AND YOU THE PURCHASER ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, even if advised of the possibility of such damages. In particular, Apple shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering such programs or data.

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

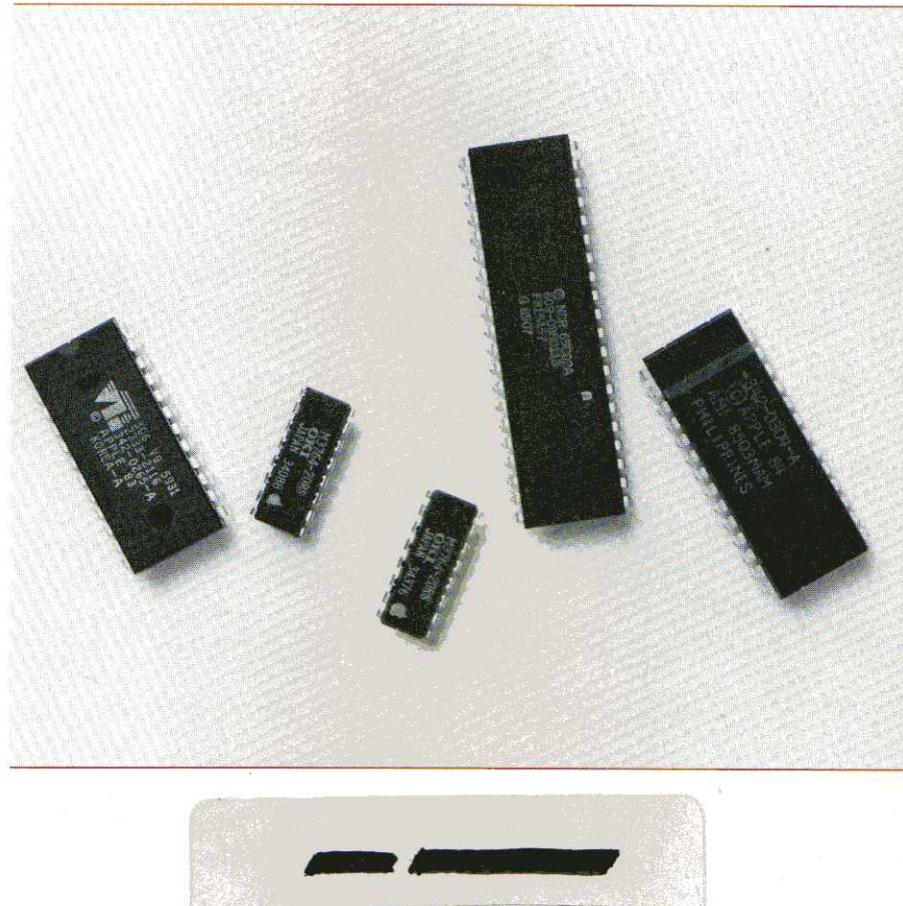
Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Warning

This equipment has been certified to comply with the limits for a Class B computing device pursuant to Subpart J of Part 15 of FCC rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and television reception.



Apple® IIe Technical Reference Manual



Addison-Wesley Publishing Company, Inc.
Reading, Massachusetts Menlo Park, California
Don Mills, Ontario Wokingham, England Amsterdam
Sydney Singapore Tokyo Mexico City Bogotá
Santiago San Juan

Copyright © 1985 by Apple Computer, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

ISBN 0-201-17720-X

ABCDEFGHIJ-DO-898765
First printing, July 1985



Apple® IIe Technical
Reference Manual

Contents

PREFACE**About This Manual**

Contents of This Manual	xxvii
The Enhanced Apple IIe	xxix
Physical Changes	xxix
Startup Drives	xxix
Video Firmware	xxx
Video Enhancements	xxx
Applesoft 80-Column Support	xxx
Applesoft Lowercase Support	xxxi
Apple II Pascal	xxxi
System Monitor Enhancements	xxxi
Interrupt Handling	xxxi
Symbols Used in This Manual	xxxii

xviii

xxv

xxvii

CHAPTER 1**Introduction**

Removing the Cover	2
The Keyboard	3
The Speaker	3
The Power Supply	4
The Circuit Board	4
Connectors on the Circuit Board	6
Connectors on the Back Panel	8

1

The Keyboard	10
Reading the Keyboard	12
The Video Display Generator	16
Text Modes	18
Text Character Sets	19
40-Column Versus 80-Column Text	20
Graphics Modes	22
Low-Resolution Graphics	22
High-Resolution Graphics	23
Double-High-Resolution Graphics	25
Video Display Pages	26
Display Mode Switching	28
Addressing Display Pages Directly	30
Secondary Inputs and Outputs	37
The Speaker	37
Cassette Input and Output	38
The Hand Control Connector Signals	39
Annunciator Outputs	40
Strobe Output	40
Switch Inputs	41
Analog Inputs	42
Summary of Secondary I/O Locations	42

CHAPTER 3

Built-in I/O Firmware

45

- Using the I/O Subroutines 47
 - Apple II Compatibility 48
 - The 80-Column Firmware 49
 - The Old Monitor 50
 - The Standard I/O Links 50
- Standard Output Features 51
 - COUT Output Subroutine 51
 - Control Characters With COUT1 and BASICOUT 52
 - The Stop-List Feature 54
 - The Text Window 54
 - Inverse and Flashing Text 56
- Standard Input Features 57
 - RDKEY Input Subroutine 57
 - KEYIN Input Subroutine 58
 - Escape Codes 58
 - Cursor Motion in Escape Mode 58
 - GETLN Input Subroutine 60
 - Editing With GETLN 61
 - Cancel Line 61
 - Backspace 61
 - Retype 62
 - Monitor Firmware Support 62
 - BASICOUT 63
 - CLREOL 63
 - CLEOLZ 64
 - CLREOP 64
 - CLRSCR 64

CLRTOP	64
COUT	64
COUT1	64
CROUT	64
CROUT1	65
HLINE	65
HOME	65
PLOT	65
PRBL2	65
PRBYTE	65
PRERR	65
PRHEX	66
PRNTAX	66
SCRN	66
SETCOL	66
VTABZ	66
VLINE	66
I/O Firmware Support	67
PINIT	67
PREAD	67
PWRITE	68
PSTATUS	69

CHAPTER 4

Memory Organization	71
Main Memory Map	72
RAM Memory Allocation	74
Reserved Memory Pages	75
Page Zero	75
The 65C02 Stack	75
The Input Buffer	76
Link-Address Storage	76
The Display Buffers	76
Bank-Switched Memory	79
Setting Bank Switches	80
Reading Bank Switches	83
Auxiliary Memory and Firmware	84
Memory Mode Switching	86
Auxiliary-Memory Subroutines	88
Moving Data to Auxiliary Memory	89
Transferring Control to Auxiliary Memory	90
The Reset Routine	91
The Cold-Start Procedure	92
The Warm-Start Procedure	92
Forced Cold Start	93
The Reset Vector	93
Automatic Self-Test	95



CHAPTER 5

Using the Monitor	97
Invoking the Monitor	98
Syntax of Monitor Commands	99
Monitor Memory Commands	100
Examining Memory Contents	100
Memory Dump	100
Changing Memory Contents	103
Changing One Byte	103
Changing Consecutive Locations	104
ASCII Input Mode	104
Moving Data in Memory	105
Comparing Data in Memory	107
Searching for Bytes in Memory	108
Examining and Changing Registers	108
Monitor Cassette Tape Commands	109
Saving Data on Tape	109
Reading Data From Tape	110
Miscellaneous Monitor Commands	112
Inverse and Normal Display	112
Back to BASIC	112
Redirecting Input and Output	113
Hexadecimal Arithmetic	114
Special Tricks With the Monitor	114
Multiple Commands	114
Filling Memory	115
Repeating Commands	116
Creating Your Own Commands	117

Machine-Language Programs	118
Running a Program	118
Disassembled Programs	119
The Mini-Assembler	121
Starting the Mini-Assembler	121
Restrictions	121
Using the Mini-Assembler	122
Mini-Assembler Instruction Formats	124
Summary of Monitor Commands	125
Examining Memory	125
Changing the Contents of Memory	126
Moving and Comparing	126
The Examine Command	126
The Search Command	126
Cassette Tape Commands	126
Miscellaneous Monitor Commands	127
Running and Listing Programs	127
The Mini-Assembler	128

CHAPTER 6

Programming for Peripheral Cards	129
Peripheral-Card Memory Spaces	130
Peripheral-Card I/O Space	130
Peripheral-Card ROM Space	131
Expansion ROM Space	132
Peripheral-Card RAM Space	134

I/O Programming Suggestions	135
Finding the Slot Number With ROM Switched In	136
I/O Addressing	136
RAM Addressing	138
Changing the Standard I/O Links	139
Other Uses of I/O Memory Space	140
Switching I/O Memory	141
Developing Cards for Slot 3	143
Pascal 1.1 Firmware Protocol	144
Device Identification	144
I/O Routine Entry Points	145
Interrupts on the Enhanced Apple IIe	146
What Is an Interrupt?	147
Interrupts on Apple IIe Series Computers	148
Rules of the Interrupt Handler	149
Interrupt Handling on the 65C02 and 6502	150
The Interrupt Vector at \$FFFE	150
The Built-in Interrupt Handler	151
Saving the Apple IIe's Memory Configuration	152
Managing Main and Auxiliary Stacks	152
The User's Interrupt Handler at \$3FE	154
Handling Break Instructions	155
Interrupt Differences: Apple IIe Versus Apple IIc	156

CHAPTER 7

Hardware Implementation	157
Environmental Specifications	158
The Power Supply	159
The Power Connector	161
The 65C02 Microprocessor	161
65C02 Timing	162
The Custom Integrated Circuits	164
The Memory Management Unit	164
The Input/Output Unit	166
The PAL Device	168
Memory Addressing	168
ROM Addressing	169
RAM Addressing	170
Dynamic-RAM Refreshment	170
Dynamic-RAM Timing	171
The Video Display	173
The Video Counters	174
Display Memory Addressing	175
Display Address Mapping	176
Video Display Modes	179
Text Displays	179
Low-Resolution Display	182
High-Resolution Display	183
Double-High-Resolution Display	185
Video Output Signals	186



Built-in I/O Circuits 187
 The Keyboard 187
 Connecting a Keypad 188
 Cassette I/O 189
 The Speaker 189
 Game I/O Signals 190
Expanding the Apple IIe 192
 The Expansion Slots 192
 The Peripheral Address Bus 192
 The Peripheral Data Bus 193
 Loading and Driving Rules 193
 Interrupt and DMA Daisy Chains 193
 Auxiliary Slot 197
 80-Column Display Signals 197

APPENDIX A

The 65C02 Microprocessor 205
Differences Between 6502 and 65C02 206
 Different Cycle Times 206
 Different Instruction Results 207
Data Sheet 207

APPENDIX B

Directory of Built-in Subroutines 217

APPENDIX C

Apple II Family Differences	225
Keyboard	226
Apple Keys	226
Character Sets	226
80-Column Display	227
Escape Codes and Control Characters	227
Built-in Language Card	227
Auxiliary Memory	228
Auxiliary Slot	228
Back Panel and Connectors	228
Soft Switches	228
Built-in Self-Test	229
Forced Reset	229
Interrupt Handling	229
Vertical Sync for Animators	229
Signature Byte	230
Hardware Implementation	230

APPENDIX D

Operating Systems and Languages	231
Operating Systems	232
ProDOS	232
DOS 3.3	232
Pascal Operatiing System	232
CP/M	233



Languages 233
 Assembly Language 233
 Applesoft BASIC 233
 Interger BASIC 233
 Pascal Language 234
 FORTRAN 234

APPENDIX E

Conversion Tables 255
 Bits and Bytes 236
 Hexadecimal and Decimal 238
 Hexadecimal and Negative Decimal 240
 Graphics Bits and Pieces 242
 Eight-Bit Code Conversions 244

APPENDIX F

Frequently Used Tables 253

APPENDIX G

Using an 80-Column Text Card 267
 Starting Up With Pascal or CP/M 268
 Starting Up With ProDOS or DOS 3.3 269
 Using the GET Command 269
 When to Switch Modes Versus When to Deactivate 270
 Display Features With the Text Card 270
 INVERSE, FLASH, NORMAL, HOME 270

Tabbing With the Original Apple IIe	271
Comma Tabbing With the Original Apple IIe	271
HTAB and POKE 1403	272
Using Control-Characters With the Card	272
Control Characters and Their Functions	273
How to Use Control-Character Codes in Programs	274
A Word of Caution to Pascal Programmers	275

APPENDIX H

Programming With the Super Serial Card	277
Locating the Card	278
Operating Modes	278
Operating Commands	279
The Command Character	280
Baud Rate, nB	280
Data Format, nD	281
Parity, nP	281
Set Time Delay, nC, nL, and nF	282
Echo Characters to the Screen, E_E/D	283
Automatic Carriage Return, C	283
Automatic Line Feed, LE/D	284
Mask Line Feed In, M_E/D	284
Reset Card, R	284
Specify Screen Slot, S	284
Translate Lowercase Characters, nT	284
Suppress Control Characters, Z	285
Find Keyboard, F_E/D	285
XOFF Recognition, X_E/D	286
Tab in BASIC, T_E/D	286

Terminal Mode	286
Entering Terminal Mode, T	286
Transmitting a Break, B	287
Special Characters, S_E/D	287
Quitting Terminal Mode, Q	287
SSC Error Codes	287
The ACIA	289
SSC Firmware Memory Use	289
Zero-Page Location	290
Peripheral Card I/O Space	290
Scratchpad RAM Location	292

APPENDIX I

Monitor ROM Listing	293
Glossary	377
Bibliography	399
Index	401
Tell Apple Card	

Figures and Tables

CHAPTER 1

Introduction

1

- | | | |
|------------|----------------------------------|---|
| Figure 1-1 | Removing the Cover | 2 |
| Figure 1-2 | The Apple IIe With the Cover Off | 2 |
| Figure 1-3 | The Apple IIe Keyboard | 3 |
| Figure 1-4 | The Circuit Board | 5 |
| Figure 1-5 | The Expansion Slots | 7 |
| Figure 1-6 | The Auxiliary Slot | 7 |
| Figure 1-7 | The Back Panel Connectors | 8 |

CHAPTER 2

Built-in I/O Devices

9

- | | | |
|------------|--|----|
| Figure 2-1 | The Keyboard | 11 |
| Table 2-1 | Apple IIe Keyboard Specifications | 11 |
| Table 2-2 | Keyboard Memory Locations | 12 |
| Table 2-3 | Keys and ASCII Codes | 14 |
| Table 2-4 | Video Display Specifications | 17 |
| Table 2-5 | Display Character Sets | 20 |
| Figure 2-2 | 40-Column Text Display | 21 |
| Figure 2-3 | 80-Column Text Display | 21 |
| Table 2-6 | Low-Resolution Graphics Colors | 23 |
| Figure 2-4 | High-Resolution Display Bits | 24 |
| Table 2-7 | High-Resolution Graphics Colors | 25 |
| Table 2-8 | Double-High-Resolution Graphics Colors | 26 |
| Table 2-9 | Video Display Page Locations | 28 |
| Table 2-10 | Display Soft Switches | 29 |
| Figure 2-5 | Map of 40-Column Text Display | 32 |
| Figure 2-6 | Map of 80-Column Text Display | 33 |
| Figure 2-7 | Map of Low-Resolution Graphics Display | 34 |



CHAPTER 3

Figure 2-8	Map of High-Resolution Graphics Display	35
Figure 2-9	Map of Double-High-Resolution Graphics Display	36
Table 2-11	Annunciator Memory Locations	40
Table 2-12	Secondary I/O Memory Location	43

45

CHAPTER 4

Memory Organization		
Figure 4-1	System Memory Map	73
Figure 4-2	RAM Allocation Map	74
Table 4-1	Monitor Zero-Page Use	77
Table 4-2	Applesoft Zero-Page Use	77
Table 4-3	Integer BASIC Zero-Page Use	78
Table 4-4	DOS 3.3 Zero-Page Use	78
Table 4-5	ProDOS MLI and Disk-Driver Zero-Page Use	79

71

CHAPTER 5	Using the Monitor	97
	Table 5-1 Mini-Assembler Address Formats	124
CHAPTER 6	Programming for Peripheral Cards	129
	Table 6-1 Peripheral-Card I/O Memory Locations Enabled by DEVICE SELECT' 131	
	Table 6-2 Peripheral-Card ROM Memory Locations Enabled by I/O SELECT' 132	
	Figure 6-1 Expansion ROM Enable Circuit 133	
	Figure 6-2 ROM Disable Address Decoding 133	
	Table 6-3 Peripheral-Card RAM Memory Locations 134	
	Table 6-4 Peripheral-Card I/O Base Addresses 137	
	Figure 6-3 I/O Memory Map 141	
	Table 6-5 I/O Memory Switches 142	
	Table 6-6 Peripheral-Card Device-Class Assignments 144	
	Table 6-7 I/O Routine Offsets and Registers Under Pascal 1.1 Protocol 146	



CHAPTER 7

Figure 6-4	Interrupt Handling Sequence
Table 6-8	BRK Handler Information
Table 6-9	Memory Configuration Information

Hardware Implementation

157

Table 7-1	Summary of Environmental Specifications
Table 7-2	Power Supply Specifications
Table 7-3	Power Connector Signal Specifications
Table 7-4	65C02 Microprocessor Specifications
Table 7-5	65C02 Timing Signal Descriptions
Figure 7-1	65C02 Timing Signals
Figure 7-2	The MMU Pinouts
Table 7-6	The MMU Signal Descriptions
Figure 7-3	The IOU Pinouts
Table 7-7	The IOU Signal Descriptions
Figure 7-4	The PAL Pinouts
Table 7-8	The PAL Signal Descriptions
Figure 7-5	The 2364 ROM Pinouts
Figure 7-6	The 2316 ROM Pinouts
Figure 7-7	The 2333 ROM Pinouts
Figure 7-8	The 64K RAM Pinouts
Table 7-9	RAM Address Multiplexing
Figure 7-9	RAM Timing Signals
Table 7-10	RAM Timing Signal Descriptions
Table 7-11	Display Address Transformation
Figure 7-10	40-Column Text Display Memory
Table 7-12	Display Memory Addressing

Table 7-13	Memory Address Bits for Display Modes	178
Figure 7-11a	7 MHz Video Timing Signals	180
Figure 7-11b	14 MHz Video Timing Signals	181
Table 7-14	Character-Generator Control Signals	182
Table 7-15	Internal Video Connector Signals	186
Table 7-16	Keyboard Connector Signals	188
Table 7-17	Keypad Connector Signals	188
Table 7-18	Speaker Connector Signals	189
Table 7-19	Game I/O Connector Signals	191
Figure 7-12	Peripheral-Signal Timing	194
Table 7-20	Expansion Slot Signals	195
Table 7-21	Auxiliary Slot Signals	198
Figure 7-13	Schematic Diagram	200

APPENDIX A

The 65C02 Microprocessor	205	
Table A-1	Cycle Time Differences	206

APPENDIX E

Conversion Tables	235	
Table E-1	What a Bit Can Represent	236
Figure E-1	Bits, Nibbles, and Bytes	237
Table E-2	Hexadecimal/Decimal Conversion	238
Table E-3	Hexadecimal to Negative Decimal Conversion	240
Table E-4	Hexadecimal Values for High-Resolution Dot Patterns	242
Table E-5	Control Characters, High Bit Off	245
Table E-6	Special Characters, High Bit Off	246

Table E-7	Uppercase Characters, High Bit Off	247
Table E-8	Lowercase Characters, High Bit Off	248
Table E-9	Control Characters, High Bit On	249
Table E-10	Special Characters, High Bit On	250
Table E-11	Uppercase Characters, High Bit On	251
Table E-12	Lowercase Characters, High Bit On	252

APPENDIX F**Frequently Used Tables****253**

Table 2-3	Keys and ASCII Codes	254
Table 2-2	Keyboard Memory Location	255
Table 2-4	Video Display Specifications	256
Table 2-8	Double-High-Resolution Graphics Colors	257
Table 2-9	Video Display Page Locations	257
Table 2-10	Display Soft Switches	258
Table 3-1	Monitor Firmware Routines	259
Table 3-3a	Control Characters With 80-Column Firmware Off	260
Table 3-3b	Control Characters With 80-Column Firmware On	260
Table 3-5	Text Format Control Values	261
Table 3-6	Escape Codes	262
Table 3-10	Pascal Video Control Functions	263
Table 4-6	Bank Select Switches	264
Table 4-7	Auxiliary-Memory Select Switches	265
Table 4-8	48K RAM Transfer Routines	265
Table 6-5	I/O Memory Switches	266
Table 6-6	I/O Routine Offsets and Registers Under Pascal 1.1 Protocol	266

APPENDIX G**Using an 80-Column Text Card****267**

Table G-1 Control Characters With 80-Column Firmware On 273

APPENDIX H**Programming With the Super Serial Card****277**

- | | | |
|------------|--|-----|
| Table H-1 | Baud Rate Selections | 280 |
| Table H-2 | Data Format Selections | 281 |
| Table H-3 | Parity Selections | 281 |
| Table H-4 | Time Delay Selections | 282 |
| Table H-5 | Lowercase Character Display Options | 285 |
| Table H-6 | STSBYTE Bit Definitions | 287 |
| Table H-7 | Error Codes and Bits | 288 |
| Table H-8 | Memory Use Map | 289 |
| Table H-9 | Zero-Page Locations Used by the SSC | 290 |
| Table H-10 | Address Register Bits Interpretation | 291 |
| Table H-11 | Scratchpad RAM Locations Used by the SSC | 292 |



Radio and Television Interference

The equipment described in this manual generates and uses radio-frequency energy. If it is not installed and used properly—that is, in strict accordance with our instructions—it may cause interference with radio and television reception.

This equipment has been tested and complies with the limits for a Class B computing device in accordance with the specifications in Subpart J, Part 15, of FCC rules. These rules are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that the interference will not occur in a particular installation, especially if a "rabbit ear" television antenna is used. (A "rabbit ear" antenna is the telescoping-rod type usually contained on television receivers.)

You can determine whether your computer is causing interference by turning it off. If the interference stops, it was probably caused by the computer or its peripherals. To further isolate the problem, disconnect the peripheral devices and their input/output cables one at a time. If the interference stops, it was caused by either the peripheral device or the I/O cable. These devices usually require shielded I/O cables. For Apple peripherals, you can obtain the proper **shielded cable** from your dealer. For non-Apple peripheral devices, contact the manufacturer or dealer for assistance.

If your computer does cause interference to radio or television reception, you can try to correct the interference by using one or more of the following measures:

- Turn the television or radio antenna until the interference stops.
- Move the computer to one side or the other of the television or radio.
- Move the computer farther away from the television or radio.

A **shielded cable** is a cable that uses a metallic wrap around the wires to reduce the potential effects of radio frequency interference.

- Plug the computer into an outlet that is on a different circuit than the television or radio. (That is, make certain the computer and the radio or television set are on circuits controlled by different circuit breakers or fuses.)
- Consider installing a rooftop television antenna with coaxial cable lead-in between the antenna and television.

If necessary, you should consult your Apple-authorized dealer or an experienced radio/television technician for additional suggestions.

About This Manual

This is the reference manual for the Apple IIe personal computer. It contains detailed descriptions of all of the hardware and firmware that make up the Apple IIe and provides the technical information that peripheral-card designers and programmers need.

This manual contains a lot of information about the way the Apple IIe works, but it doesn't tell you how to use the Apple IIe. For this, you should read the other Apple IIe manuals, especially the following:

- Apple IIe Owner's Manual*
- The Applesoft Tutorial*

Contents of This Manual

The material in this manual is presented roughly in order of increasing intimacy with the hardware; the farther you go in the manual, the more technical the material becomes. The main subject areas are

- introduction: Preface and Chapter 1
- use of built-in features: Chapters 2 and 3
- how the memory is organized: Chapter 4
- information for programmers: Chapters 5 and 6
- hardware implementation: Chapter 7
- additional information: appendixes, glossary, and bibliography.

Chapter 1 identifies the main parts of the Apple IIe and tells where in the manual each part is described.

The next two chapters describe the built-in input and output features of the Apple IIe. This part of the manual includes information you need for low-level programming on the Apple IIe. Chapter 2 describes the built-in I/O features and Chapter 3 tells you how to use the firmware that supports them.

Chapter 4 describes the way the Apple IIe's memory space is organized, including the allocation of programmable memory for the video display buffers.

Chapter 5 is a user manual for the Monitor that is included in the built-in firmware. The Monitor is a system program that you can use for program debugging at the machine level.

Chapter 6 describes the programmable features of the peripheral-card connectors and gives guidelines for their use. It also describes interrupt programming on the Apple IIe.

Chapter 7 is a description of the hardware that implements the features described in the earlier chapters. This information is included primarily for programmers and peripheral-card designers, but it will also help you if you just want to understand more about the way the Apple IIe works.

Additional reference information appears in the appendixes. Appendix A is the manufacturer's description of the Apple IIe's microprocessor.

Appendix B is a directory of the built-in I/O subroutines, including their functions and starting addresses.

Appendix C describes differences among Apple II family members.

Appendix D describes some of the operating systems and languages supported by Apple Computer for the Apple IIe.

Appendix E contains conversion tables of interest to programmers.

Appendix F contains additional copies of some of the tables that appear in the body of the manual. The ones you will need to refer to often are duplicated here for easy reference.

Appendix G contains information about using Apple IIe 80-column text cards with the Apple IIe and high level languages.

Appendix H discusses programming on the Apple IIe with the Apple Super Serial Card.

Appendix I contains the source listing of the Monitor firmware. You can refer to it to find out more about the operation of the Monitor subroutines listed in Appendix B.

Following Appendix I is a glossary defining many of the technical terms used in this manual. Some terms that describe the use of the Apple IIe are defined in the glossaries of the other manuals listed earlier.

Following the glossary, there is a selected bibliography of sources of additional information.

The Enhanced Apple IIe

Changes have been made in the Apple IIe since the original version was introduced. The new version is called the enhanced Apple IIe and is described in this manual. Where there are differences in the original Apple IIe compared with the enhanced Apple IIe, they will be called out in the manual. Otherwise, the two machines operate identically.

You can tell whether you have an original or enhanced Apple IIe when you start up the system. An original Apple IIe will display **Apple II** at the top of the monitor screen, while an enhanced Apple IIe will display **Apple //e**.

The changes embodied in the enhanced Apple IIe are described in the following sections of this preface.

Physical Changes

The enhanced Apple IIe includes the following changes from the original Apple IIe:

- The 65C02 microprocessor, which is a new version of the 6502 microprocessor found in the original Apple IIe. The 65C02 uses less power, has 27 new **opcodes**, and runs at the same speed as the 6502. (See Chapter 7 and Appendix A.)
- A new video ROM containing the same MouseText characters found in the Apple IIc. (See Chapter 2.)
- New Monitor ROMs (the CD and EF ROMs) containing the enhanced Apple IIe firmware. (See Chapter 5.)
- The identification byte at \$FBC0 has been changed. In the original Apple IIe it was \$EA (decimal 234), in the enhanced Apple IIe it is \$E0 (decimal 224).

Opcode is short for *operation code* and is used to describe the basic instructions performed by the central processing unit of a computer.

Startup Drives

You can use startup (boot) devices other than a Disk II to start up ProDOS on the enhanced Apple IIe.

Apple II Pascal versions 1.3 and later may start up from slots 4, 5, or 6 on a Disk II, ProFile, or other Apple II disk drive. Apple II Pascal versions 1.0 through 1.2 must start up from a Disk II in slot 6.

DOS 3.3 may be started from a Disk II in any slot.

When you turn on your Apple IIe, it searches for a disk drive controller to start up from, beginning with slot 7 and working down toward slot 1. As soon as a disk controller card is found, the Apple IIe will try to load and execute the operating system found on the disk. If the drive is not a Disk II, then the operating system of the startup volume must be either ProDOS or Apple II Pascal (version 1.3 or later). If it is a Disk II, then the startup volume may be any Apple II operating system.

Video Firmware

The enhanced Apple IIe has improved 80-column firmware:

- The enhanced Apple IIe now supports lowercase input.
- **[ESC] [CONTROL]-[E]** passes most control characters to the screen.
- **[ESC] [CONTROL]-[D]** traps most control characters before they get to the screen.
- **[ESC] [R]** was removed because uppercase characters are no longer required by Applesoft.

Video Enhancements

Both 80-column Pascal and 80-column mode Applesoft output are faster than before and scrolling is smoother. 40-column Pascal performance is unchanged.

In the original Apple IIe, characters echoed to COUT1 during 80-column operation were printed in every other column; the enhanced Apple IIe firmware now prints the characters in each column.

Applesoft 80-Column Support

The following Applesoft routines now work in 80-column mode:

- HTAB
- TAB
- SPC
- Comma tabbing in PRINT statements

Applesoft Lowercase Support

Applesoft now lets you do all your programming in lowercase. When you list your programs, all Applesoft keywords and variable names automatically are in uppercase characters; literal strings and the contents of DATA and REM statements are unchanged.

Apple II Pascal

Apple II Pascal¹ (version 1.2 and later) can now use a ProFile hard disk through the Pascal ProFile Manager.

To find out more, see the *Pascal ProFile Manager Manual*.

The Pascal 1.1 firmware no longer supports the control character that switches from 80-column to 40-column operation. This control character is no longer supported because it can put Pascal in a condition where the exact memory configuration is not known.

System Monitor Enhancements

Enhancements to the Apple IIe's built-in Monitor (described in Chapter 5 in this manual) include the following:

- lowercase input
- ASCII input mode
- Monitor Search command
- the Mini-Assembler

Interrupt Handling

Interrupt handler support in the enhanced Apple IIe firmware now handles any Apple IIe memory configuration.

Symbols Used in This Manual

Special text in this manual is set off in several different ways, as shown in these examples.

▲Warning

Important warnings appear in red like this. These flag potential danger to the Apple IIe, its software, or you.

Important!

The information here is important, but non-threatening. The ways in which the original Apple IIe differs from the enhanced Apple IIe are called out this way with the tag **Original IIe** in the margin.

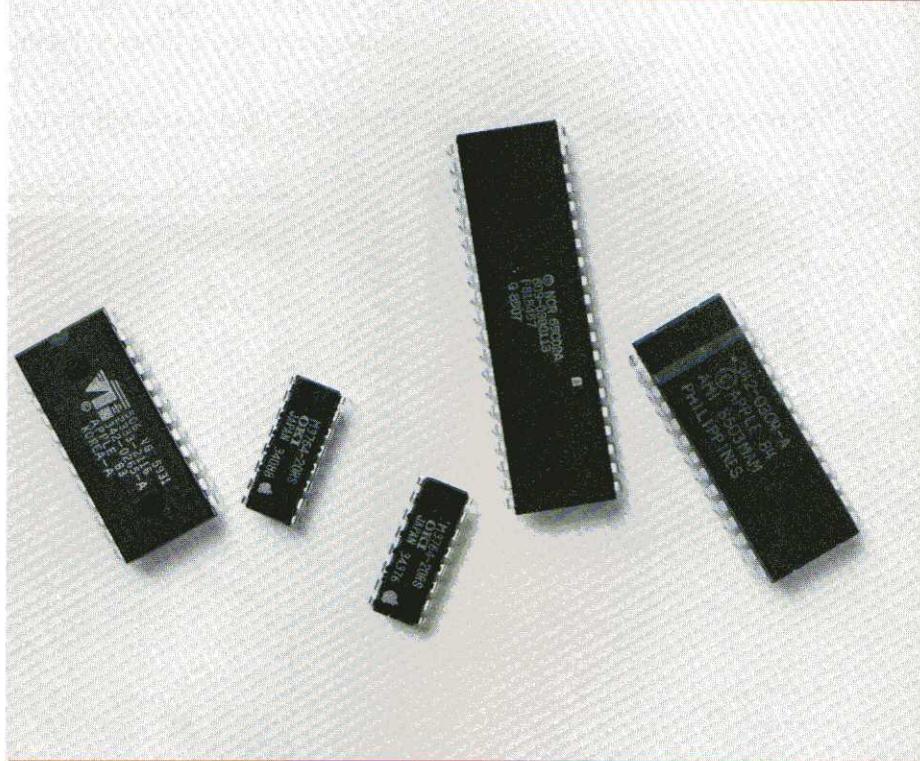
Definitions, cross-references, and other short items appear in marginal glosses like this.

By the Way: Information that is useful but is incidental to the text is set off like this. You may want to skip over such information and return to it later.

Terms that are defined in a marginal gloss or in the glossary appear in **boldface**.

Chapter 1

Introduction



This first chapter introduces you to the Apple IIe itself. It shows you what the inside looks like, identifies the major components that make up the machine, and tells you where to find information about each one.

Removing the Cover

Remove the cover of the Apple IIe by pulling up on the back edge until the fasteners on either side pop loose, then move the cover an inch or so toward the rear of the machine to free the front of the cover, as shown in Figure 1-1. What you will see is shown in Figure 1-2.

Figure 1-1. Removing the Cover

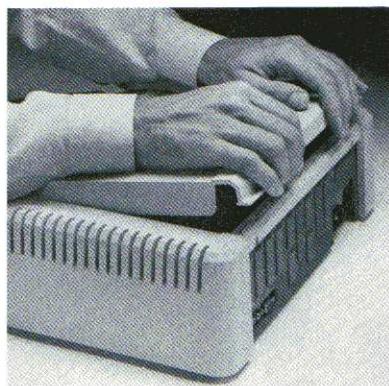
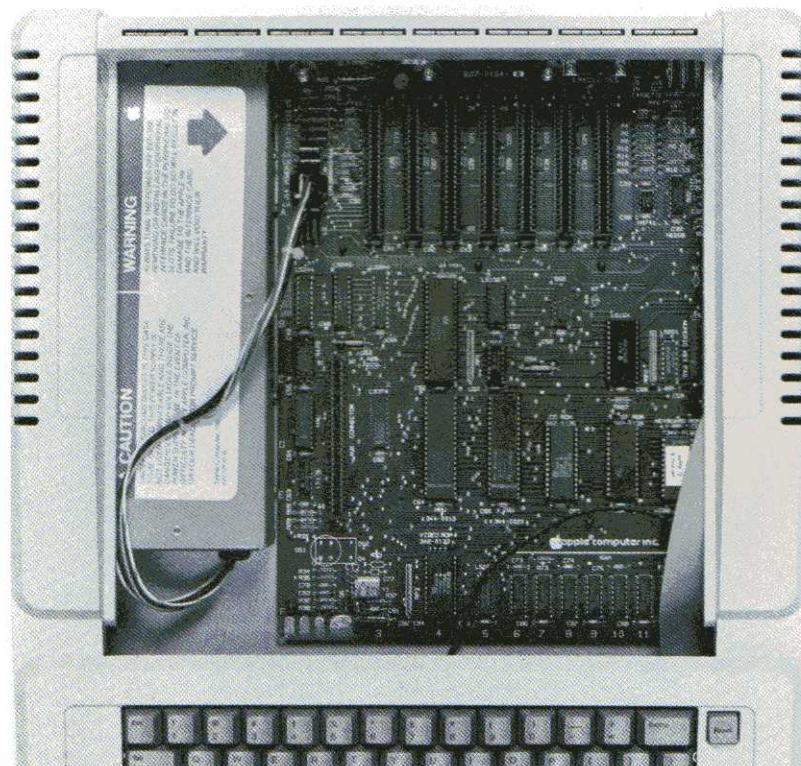


Figure 1-2. The Apple IIe With the Cover Off



▲Warning

There is a red LED (light-emitting diode) inside the Apple IIe, in the left rear corner of the circuit board. If the LED is on, it means that the power is on and you must turn it off before you insert or remove anything. To avoid damaging the Apple IIe, don't even *think* of changing anything inside it without first turning off the power.

The Keyboard

The keyboard is the Apple IIe's primary input device. As shown in Figure 1-3, it has a normal typewriter layout, uppercase and lowercase, with all of the special characters in the **ASCII** character set. The keyboard is fully integrated into the machine; its operation is described in the first part of Chapter 2. Firmware subroutines for reading the keyboard are described in Chapter 3.

Figure 1-3. The Apple IIe Keyboard



The Speaker

The Apple IIe has a small loudspeaker in the bottom of the case. The speaker enables Apple IIe programs to produce a variety of sounds that make the programs more useful and interesting. The way programs control the speaker is described in Chapter 2.

The Power Supply

The power supply is inside the flat metal box along the left side of the interior of the Apple IIe. It provides power for the main board and for any peripheral cards installed in the Apple IIe.

The power supply produces four different voltages: +5V, -5V, +12V, and -12V. It is a high-efficiency switching supply; it includes special circuits that protect it and the rest of the Apple IIe against short circuits and other mishaps. Complete specifications of the Apple IIe power supply appear in Chapter 7.

▲Warning

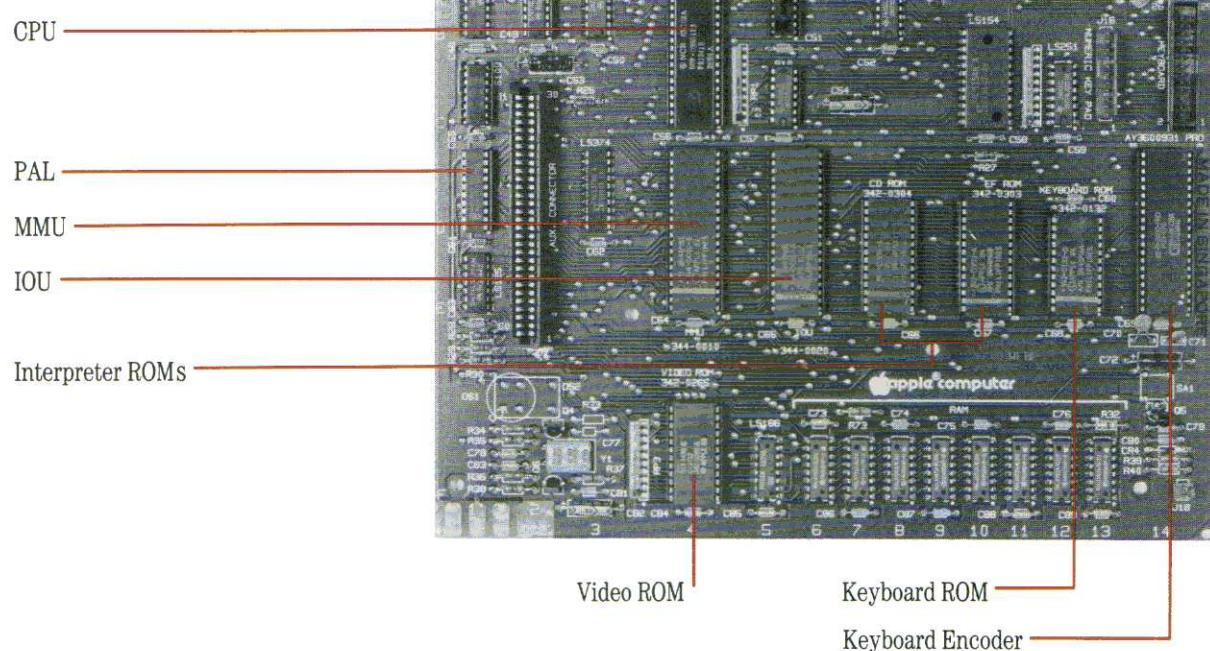
The power switch and the socket for the power cord are mounted directly on the back of the power supply's metal case. This mounting ensures that all the circuits that carry dangerous voltages are inside the power supply. Do not defeat this design feature by attempting to open the power supply.

The Circuit Board

All of the electronic parts of the Apple IIe are attached to the circuit board, which is mounted flat in the bottom of the case.

Figure 1-4 shows the main integrated circuits (ICs) in the Apple IIe. They are the central processing unit (CPU), the keyboard encoder, the keyboard read-only memory (ROM), the two interpreter ROMs, the video ROM, and the custom integrated circuits: the Input/Output Unit (IOU), the Memory Management Unit (MMU), and the Programmed Array Logic (PAL) device.

Figure 1-4. The Circuit Board



The CPU is a 65C02 microprocessor. The 65C02 is an enhanced version of the 6502, which is an eight-bit microprocessor with a sixteen-bit address bus. It uses instruction pipelining for faster processing than comparable microprocessors. In the Apple IIe, the 65C02 runs at 1.02 MHz and performs up to 500,000 eight-bit operations per second. The specifications of the 65C02 and its instruction set are given in Appendix A.

The original version of the Apple IIe uses the 6502 microprocessor. You can tell which version of Apple IIe that you have by starting up your machine. An original Apple IIe displays **Apple II** at the top of the screen during startup, while an enhanced Apple IIe displays **Apple //e**. This manual will call out specific areas where the two versions of the Apple IIe are different.

Original IIe

The 6502 is very similar to the 65C02, but lacks 10 instructions and 2 addressing modes found on the 65C02. The 6502 is an NMOS device and so uses more power than the CMOS 65C02. Except for the differences listed above, and some minor differences in the number of clock cycles used by some instructions, the two microprocessors are identical.

The keyboard is decoded by an AY-3600-type integrated circuit and a read-only memory (ROM). These devices are described in Chapter 7.

The interpreter ROMs are integrated circuits that contain the Applesoft BASIC interpreter. The ROMs are described in Chapter 7. The Applesoft language is described in the *Applesoft Tutorial* and the *Applesoft BASIC Programmer's Reference Manual*.

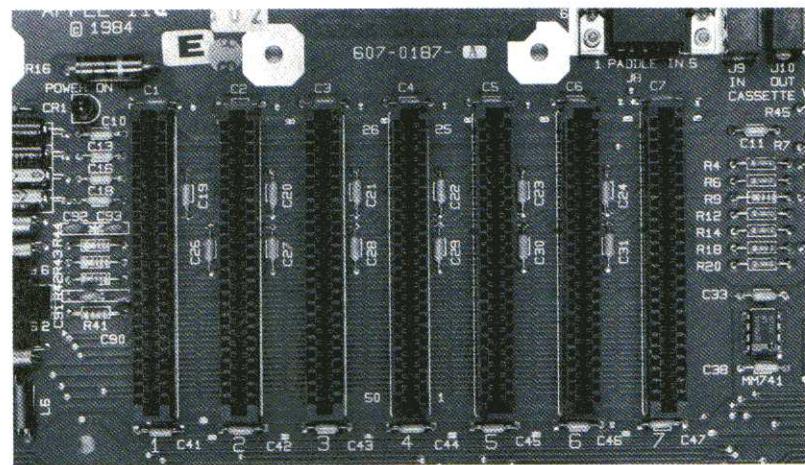
Two of the large ICs are custom-made for the Apple IIe: the MMU and the IOU. The MMU IC contains most of the logic that controls memory addressing in the Apple IIe. The organization of the memory is described in Chapter 4; the circuitry in the MMU itself is described in Chapter 7.

The IOU IC contains most of the logic that controls the built-in input/output features of the Apple IIe. These features are described in Chapter 2 and Chapter 3; the IOU circuits are described in Chapter 7.

Connectors on the Circuit Board

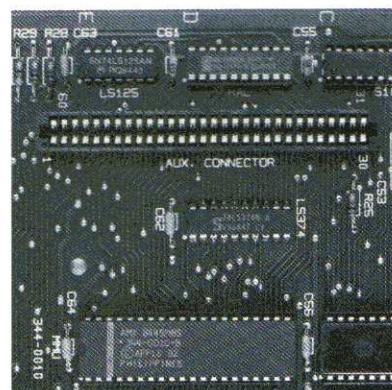
The seven slots lined up along the back of the Apple IIe circuit board are the expansion slots, sometimes called peripheral slots. (See Figure 1-5.) These slots make it possible to attach additional hardware to the Apple IIe. Chapter 6 tells you how your programs deal with the devices that plug into these slots; Chapter 7 describes the circuitry for the slots themselves.

Figure 1-5. The Expansion Slots



The large slot next to the left-hand side of the circuit board is the auxiliary slot (Figure 1-6). If your Apple IIe has an Apple IIe 80-column text card, it will be installed in this slot. The 80-column display option is fully integrated into the Apple IIe; it is described along with the other display features in Chapter 2. The hardware and firmware interfaces to this card are described in Chapter 7.

Figure 1-6. The Auxiliary Slot



There are also smaller connectors for game I/O and for an internal RF (radio frequency) modulator. These connectors are described in Chapter 7.

Connectors on the Back Panel

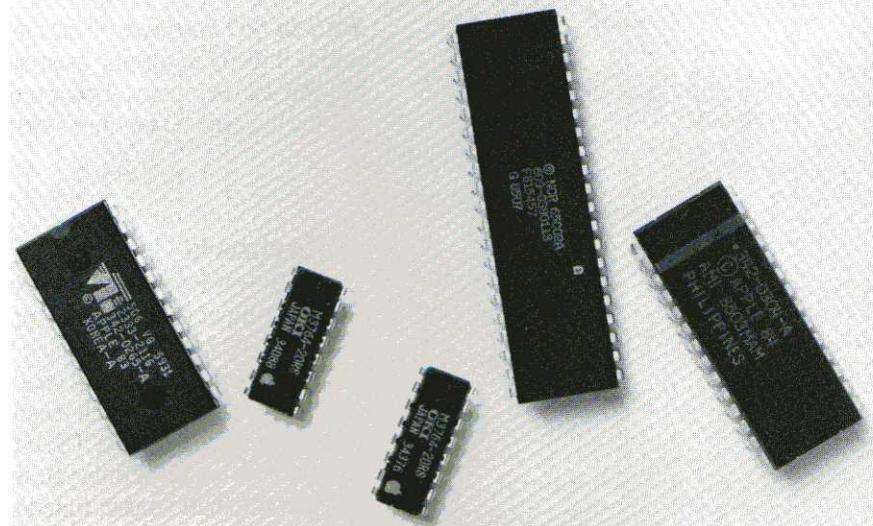
The back of the Apple IIe has two miniature phone jacks for connecting a cassette recorder, an RCA-type jack for a video monitor, and a 9-pin D-type miniature connector for the hand controls, as shown in Figure 1-7. In addition to these, there are spaces for additional connectors used with the peripheral cards installed in the Apple IIe. The installation manuals for the peripheral cards contain instructions for installing the peripheral connectors.

Figure 1-7. The Back Panel Connectors



Chapter 2

Built-in I/O Devices



This chapter describes the input and output (I/O) devices built into the Apple IIe in terms of their functions and the way they are used by programs. The built-in I/O devices are

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

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

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

For descriptions of the built-in I/O hardware refer to Chapter 7.

Built-in I/O firmware routines are described in Chapter 3.

The Keyboard

The primary input device of the Apple IIe is its built-in keyboard. The keyboard has 63 keys and is similar to a typewriter keyboard. The Apple IIe keyboard has automatic repeat on all keys: hold the key down to repeat. It also has N-key rollover, which means that you can hold down any number of keys while typing another. Of course, if you hold the keys down much longer than the length of time you would hold them down during normal typing, the automatic-repeat function will start repeating the last key you pressed.

The keyboard arrangement shown in Figure 2-1 is the standard one used in the United States. The specifications for the keyboard are given in Table 2-1. Apple IIe's manufactured for sale outside the United States have a slightly different standard keyboard arrangement and include provisions for switching between two different arrangements.

Figure 2-1. The Keyboard



Table 2-1. Apple IIe Keyboard Specifications

Number of keys:	63
Character encoding:	ASCII
Number of codes:	128
Features:	Automatic repeat, two-key rollover
Special function keys:	[RESET], [C], [apple]
Cursor movement keys:	[←], [→], [↓], [↑], [RETURN], [DELETE], [TAB]
Modifier keys:	[CONTROL], [SHIFT], [CAPS LOCK], [ESC]
Electrical Interface:	AY-5-3600 keyboard encoder

In addition to the keys normally used for typing characters, there are four cursor-control keys with arrows: left, right, down, and up. The cursor-control keys can be read the same as other keys; their codes are \$08, \$15, \$0A, and \$0B. (See Table 2-3.)

Three special keys, [CONTROL], [SHIFT], and [CAPS LOCK], change the codes generated by the other keys. The [CONTROL] key is similar to the ASCII CTRL key.

Three other keys have special functions: the [RESET] key, and two keys marked with apples, one outlined, or open (□), and one solid, or closed (■). Pressing the [RESET] key with the [CONTROL] key depressed resets the Apple IIe, as described in Chapter 4. The Apple keys are connected to the one-bit game inputs, described later in this chapter.

See Chapter 7 for a complete description of the electrical interface to the keyboard.

The electrical interface between the Apple IIe and the keyboard is a ribbon cable with a 26-pin connector. This cable carries the keyboard signals to the encoding circuitry on the main board.

Reading the Keyboard

The keyboard encoder and ROM generate all 128 ASCII codes, so all of the special character codes in the ASCII character set are available from the keyboard. Machine-language programs obtain character codes from the keyboard by reading a byte from the keyboard-data location shown in Table 2-2.

Table 2-2. Keyboard Memory Locations

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

Hexadecimal refers to the base-16 number system, which uses the digits 0 through 9 and the six letters A through F to represent values from 0 to 15.

Your programs can get the code for the last key pressed by reading the keyboard-data location. Table 2-2 gives this location in three different forms: the **hexadecimal** value used in assembly language, indicated by a preceding dollar sign (\$); the decimal value used in Applesoft BASIC, and the complementary decimal value used in Apple Integer BASIC. (Integer BASIC requires that values greater than 32767 be written as the number obtained by subtracting 65536 from the value. These are the decimal numbers shown as negative in tables in this manual; refer to the *Apple II BASIC Programming Manual*.) The low-order seven bits of the byte at the keyboard location contain the character code; the high-order bit of this byte is the strobe bit, described later in this section.

Your program can find out whether any key is down, except the **RESET**, **CONTROL**, **SHIFT**, **CAPS LOCK**, **OPTION**, and **APPLE** keys by reading from location 49168 (hexadecimal \$C010 or complementary decimal -16368). The high-order bit (bit 7) of the byte you read at this location is called any-key-down; it is 1 if a key is down, and 0 if no key is down. The value of this bit is 128; if a BASIC program gets this information with a PEEK, the value is 128 or greater if any key is down, and less than 128 if no key is down.

The [A] and [B] keys are connected to switches 0 and 1 of the game I/O connector inputs. If [A] is pressed, switch 0 is “pressed,” and if [B] is pressed, switch 1 is “pressed.”

The strobe bit is the high-order bit of the keyboard-data byte. After any key has been pressed, the strobe bit is high. It remains high until you reset it by reading or writing at the clear-strobe location. This location is a combination flag and switch; the flag tells whether any key is down, and the switch clears the strobe bit. The switch function of this memory location is called a soft switch because it is controlled by software. In this case, it doesn’t matter whether the program reads or writes, and it doesn’t matter what data the program writes: the only action that occurs is the resetting of the keyboard strobe. Similar soft switches, described later, are used for controlling other functions in the Apple IIe.

Important!

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

After the keyboard strobe has been cleared, it remains low until another key is pressed. Even after you have cleared the strobe, you can still read the character code at the keyboard location. The data byte has a different value, because the high-order bit is no longer set, but the ASCII code in the seven low-order bits is the same until another key is pressed. Table 2-3 shows the ASCII codes for most of the keys on the keyboard of the Apple IIe.

There are several special-function keys that do not generate ASCII codes. For example, you cannot read the [CONTROL], [SHIFT], and [CAPS LOCK] keys directly, but pressing one of these keys alters the character codes produced by the other keys.

Another key that doesn’t generate a code is [RESET], located at the upper-right corner of the keyboard; it is connected directly to the Apple IIe’s circuits. Pressing [RESET] with [CONTROL] depressed normally causes the system to stop whatever program it’s running and restart itself. This restarting process is called the reset routine.

Two more special keys are the Apple keys, [A] and [B], located on either side of the [SPACE] bar. These keys are connected to the one-bit game inputs, which are described later in this chapter in the section “Switch Inputs.” Pressing them in combination with the [CONTROL] and [RESET] keys causes the built-in firmware to perform special reset and self-test cycles, described with the reset routine in Chapter 4.

The reset routine is described in Chapter 4.

Table 2-3. Keys and ASCII Codes

Note: Codes are shown here in hexadecimal; to find the decimal equivalents, refer to Table E-2.

Key	Normal		Control		Shift		Both	
	Code	Char	Code	Char	Code	Char	Code	Char
DELETE	7F	DEL	7F	DEL	7F	DEL	7F	DEL
←	08	BS	08	BS	08	BS	08	BS
TAB	09	HT	09	HT	09	HT	09	HT
↓	0A	LF	0A	LF	0A	LF	0A	LF
↑	0B	VT	0B	VT	0B	VT	0B	VT
RETURN	0D	CR	0D	CR	0D	CR	0D	CR
→	15	NAK	15	NAK	15	NAK	15	NAK
ESC	1B	ESC	1B	ESC	1B	ESC	1B	ESC
SPACE	20	SP	20	SP	20	SP	20	SP
,	27	,	27	,	22	,	22	,
<	2C	,	2C	,	3C	<	3C	<
-	2D	-	1F	US	5F	-	1F	US
. >	2E	.	2E	.	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0)	30	0	30	0	29)	29)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 \$	34	4	34	4	24	\$	24	\$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	*	2A	*
9 (39	9	39	9	28	(28	(
;;	3B	;	3B	;	3A	:	3A	:
= +	3D	=	3D	=	2B	+	2B	+
[{	5B	[1B	ESC	7B	{	1B	ESC
\ \}	5C	\	1C	FS	7C		1C	FS
] }	5D]	1D	GS	7D	}	1D	GS
	60		60		7E		7E	

Table 2-3—Continued. Keys and ASCII Codes

Note: Codes are shown here in hexadecimal; to find the decimal equivalents, refer to Table E-2.

Key	Normal		Control		Shift		Both	
	Code	Char	Code	Char	Code	Char	Code	Char
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

The Video Display Generator

The primary output device of the Apple IIe is the video display. You can use any ordinary video monitor, either color or black-and-white, to display video information from the Apple IIe. An ordinary monitor is one that accepts composite video compatible with the standard set by the NTSC (National Television Standards Committee). If you use Apple IIe color graphics with a black-and-white monitor, the display will appear as black and white (or green or amber or...) and various patterns of these two shades mixed together.

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

Important!

With the 80-column text card installed, the Apple IIe can produce an 80-column text display. However, if you use an ordinary color or black-and-white television set, 80-column text will be too blurry to read. For a clear 80-column display, you must use a high-resolution video monitor with a bandwidth of 14 MHz or greater.

The specifications for the video display are summarized in Table 2-4.

Original IIe

Note that MouseText characters are not included in the original version of the Apple IIe.

The video signal produced by the Apple IIe is NTSC-compatible composite color video. It is available at three places: the RCA-type phono jack on the back of the Apple IIe, the single Molex-type pin on the main circuit board near the back on the right side, and one of the group of four Molex-type pins in the same area on the main board. Use the RCA-type phono jack to connect a video monitor or an external video modulator; use the Molex pins to connect the type of video modulator that fits inside the Apple IIe case.

For a full description of the video signal and the connections to the Molex-type pins, refer to the section "Video Output Signals" in Chapter 7.

Table 2-4. Video Display Specifications

Display modes:	40-column text; map: Figure 2-5 80-column text; map: Figure 2-6
	Low-resolution color graphics; map: Figure 2-7
	High-resolution color graphics; map: Figure 2-8
	Double-high-res. color graphics; map: Figure 2-9
Text capacity:	24 lines by 80 columns (character positions)
Character set:	96 ASCII characters (uppercase and lowercase)
Display formats:	Normal, inverse, flashing, MouseText (Table 2-5)
Low-resolution graphics:	16 colors (Table 2-6) 40 horizontal by 48 vertical; map: Figure 2-7
High-resolution graphics:	6 colors (Table 2-7) 140 horizontal by 192 vertical (restricted) Black-and-white: 280 horizontal by 192 vertical; map: Figure 2-8
Double-high-resolution graphics:	16 colors (Table 2-8) 140 horizontal by 192 vertical (no restrictions) Black-and-white: 560 horizontal by 192 vertical; map: Figure 2-9

The Apple IIe can produce seven different kinds of video display:

- text, 24 lines of 40 characters
- text, 24 lines of 80 characters (with optional text card)
- low-resolution graphics, 40 by 48, in 16 colors
- high-resolution graphics, 140 by 192, in 6 colors
- high-resolution graphics, 280 by 192, in black and white
- double high-resolution graphics, 140 by 192, in 16 colors (with optional 64K text card)
- double high-resolution graphics, 560 by 192, in black and white (with optional 64K text card)

The two text modes can display all 96 ASCII characters: the uppercase and lowercase letters, numbers, and symbols. The enhanced Apple IIe can also display MouseText characters.

Any of the graphics displays can have 4 lines of text at the bottom of the screen. The text may be either 40-column or 80-column, except that double-high-resolution graphics may only have 80-column text at the bottom of the screen. Graphics displays with text at the bottom are called mixed-mode displays.

The low-resolution graphics display is an array of colored blocks, 40 wide by 48 high, in any of 16 colors. In mixed mode, the 4 lines of text replace the bottom 8 rows of blocks, leaving 40 rows of 40 blocks each.

The high-resolution graphics display is an array of dots, 280 wide by 192 high. There are 6 colors available in high-resolution displays, but a given dot can use only 4 of the 6 colors. In mixed mode, the 4 lines of text replace the bottom 32 rows of dots, leaving 160 rows of 280 dots each.

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

Text Modes

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

The normal display has white (or other single color) dots on a black background. Characters can also be displayed as black dots on a white background; this is called inverse format.

Text Character Sets

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

- normal, with white dots on a black screen
- inverse, with black dots on a white screen
- flashing, alternating between normal and inverse.

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

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

- uppercase letters
- lowercase letters
- numbers
- special characters.

In inverse format, you can get

- MouseText characters (on the enhanced Apple IIe)
- uppercase letters
- lowercase letters
- numbers
- special characters.

The MouseText characters that replace the alternate uppercase inverse characters in the range of \$40-\$5F in the original Apple IIe are inverse characters, but they don't look like it because of the way that they have been constructed.

You select the character set by means of the alternate-text soft switch, ALTCHAR, described later in the section "Display Mode Switching." Table 2-5 shows the character codes in hexadecimal for the Apple IIe primary and alternate character sets in normal, inverse, and flashing formats.

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

Table 2-5. Display Character Sets

Note: To identify particular characters and values, refer to Table 2-3.

Hex Values	Primary Character Set Character Type	Format	Alternate Character Set Character Type	Format
\$00-\$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
\$20-\$3F	Special characters	Inverse	Special characters	Inverse
\$40-\$5F	Uppercase letters	Flashing	MouseText	
\$60-\$7F	Special characters	Flashing	Lowercase letters	Inverse
\$80-\$9F	Uppercase letters	Normal	Uppercase letters	Normal
\$A0-\$BF	Special characters	Normal	Special characters	Normal
\$C0-\$DF	Uppercase letters	Normal	Uppercase letters	Normal
\$E0-\$FF	Lowercase letters	Normal	Lowercase letters	Normal

Original IIe

In the alternate character set of the original Apple IIe, characters in the range \$40-\$5F are uppercase inverse.

40-Column Versus 80-Column Text

The Apple IIe has two modes of text display: 40-column and 80-column. (The 80-column display mode described in this manual is the one you get with the Apple IIe 80-Column Text Card or other auxiliary-memory card installed in the auxiliary slot.) The number of dots in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare Figure 2-2 and Figure 2-3. On an ordinary color or black-and-white television set, the narrow characters in the 80-column display blur together; you must use the 40-column mode to display text on a television set.

Figure 2-2. 40-Column Text Display

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

Figure 2-3. 80-Column Text Display

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

Graphics Modes

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

Low-Resolution Graphics

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

Data for the low-resolution graphics display is stored in the same part of memory as the data for the 40-column text display. Each byte contains data for two low-resolution graphics blocks. The two blocks are displayed one atop the other in a display space the same size as a 40-column text character, seven dots wide by eight dots high.

Half a byte—four bits, or one nibble—is assigned to each graphics block. Each nibble can have a value from 0 to 15, and this value determines which one of sixteen colors appears on the screen. The colors and their corresponding nibble values are shown in Table 2-6. In each byte, the low-order nibble sets the color for the top block of the pair, and the high-order nibble sets the color for the bottom block. Thus, a byte containing the hexadecimal value \$D8 produces a brown block atop a yellow block on the screen.

Table 2-6. Low-Resolution Graphics Colors

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

Nibble Value			Nibble Value		
Dec	Hex	Color	Dec	Hex	Color
0	\$00	Black	8	\$08	Brown
1	\$01	Magenta	9	\$09	Orange
2	\$02	Dark Blue	10	\$0A	Gray 2
3	\$03	Purple	11	\$0B	Pink
4	\$04	Dark Green	12	\$0C	Light Green
5	\$05	Gray 1	13	\$0D	Yellow
6	\$06	Medium Blue	14	\$0E	Aquamarine
7	\$07	Light Blue	15	\$0F	White

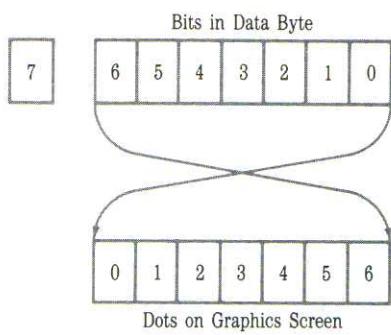
As explained later in the section "Display Pages," the text display and the low-resolution graphics display use the same area in memory. Most programs that generate text and graphics clear this part of memory when they change display modes, but it is possible to store data as text and display it as graphics,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

High-Resolution Graphics

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

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

Figure 2-4. High-Resolution Display Bits



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

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

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

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

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

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

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

For more details about the way the Apple IIe produces color on a TV set, see the section "Video Display Modes" in Chapter 7.

These rules are summarized in Table 2-7. The blacks and whites are numbered to remind you that the high-order bit is different.

Table 2-7. High-Resolution Graphics Colors

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

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

For information about the way NTSC color television works, see the magazine articles listed in the bibliography.

The peculiar behavior of the high-resolution colors reflects the way NTSC color television works. The dots that make up the Apple IIe video signal are spaced to coincide with the frequency of the color subcarrier used in the NTSC system. Alternating black and white dots at this spacing cause a color monitor or TV set to produce color, but two or more white dots together do not.

Double-High-Resolution Graphics

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

Unlike high-resolution color, double-high-resolution color has no restrictions on which colors can be adjacent. Color is determined by any four adjacent dots along a line. Think of a 4-dot-wide window moving across the screen: at any given time, the color displayed will correspond to the 4-bit value from Table 2-8 that corresponds to the window's position (Figure 2-9). Effective horizontal resolution with color is 140 (560 divided by four) dots per line.

To use Table 2-8, divide the display column number by 4, and use the remainder to find the correct column in the table: $ab0$ is a byte residing in auxiliary memory corresponding to a remainder of 0 (byte 0, 4, 8, and so on); $mb1$ is a byte residing in main memory corresponding to a remainder of 1 (byte 1, 5, 9 and so on), and similarly for $ab3$ and $mb4$.

Table 2-8. Double-High-Resolution Graphics Colors

Color	ab0	mb1	ab2	mb3	Repeated Bit Pattern
Black	\$00	\$00	\$00	\$00	0000
Magenta	\$08	\$11	\$22	\$44	0001
Brown	\$44	\$08	\$11	\$22	0010
Orange	\$4C	\$19	\$33	\$66	0011
Dark Green	\$22	\$44	\$08	\$11	0100
Gray 1	\$2A	\$55	\$2A	\$55	0101
Green	\$66	\$4C	\$19	\$33	0110
Yellow	\$6E	\$5D	\$3B	\$77	0111
Dark Blue	\$11	\$22	\$44	\$08	1000
Purple	\$19	\$33	\$66	\$4C	1001
Gray 2	\$55	\$2A	\$55	\$2A	1010
Pink	\$5D	\$3B	\$77	\$6E	1011
Medium Blue	\$33	\$66	\$4C	\$19	1100
Light Blue	\$3B	\$77	\$6E	\$5D	1101
Aqua	\$77	\$6E	\$5D	\$3B	1110
White	\$7F	\$7F	\$7F	\$7F	1111

Video Display Pages

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



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

The 80-column text mode displays twice as much data as the 40-column mode—1920 bytes—but it cannot switch pages. The 80-column text display uses a combination page made up of text Page 1 in main memory plus another page in auxiliary memory located on the 80-column text card. This additional memory is *not* the same as text Page 2—in fact, it occupies the same address space as text Page 1, and there is a special soft switch that enables you to store data into it. (See the next section “Display Mode Switching.”) The built-in firmware I/O routines described in Chapter 3 take care of this extra addressing automatically; that is one reason to use those routines for all your normal text output.

The high-resolution graphics mode also has two display pages, but each page is 8192 bytes long. In the 40-column text and low-resolution graphics modes each byte controls a display area seven dots wide by eight dots high. In high-resolution graphics mode each byte controls an area seven dots wide by one dot high. Thus, a high-resolution display requires eight times as much data storage, as shown in Table 2-9.

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

Table 2-9. Video Display Page Locations

Display Mode	Display Page	Lowest Address		Highest Address	
		Hex	Dec	Hex	Dec
40-column text, low-resolution graphics	1	\$0400	1024	\$07FF	2047
	2 *	\$0800	2048	\$0BFF	3071
80-column text	1	\$0400	1024	\$07FF	2047
	2 *	\$0800	2048	\$0BFF	3071
High-resolution graphics	1	\$2000	8192	\$3FFF	16383
	2	\$4000	16384	\$5FFF	24575
Double-high- resolution graphics	1 †	\$2000	8192	\$3FFF	16383
	2 †	\$4000	16384	\$5FFF	24575

* This is not supported by firmware; for instructions on how to switch pages, refer to the next section "Display Mode Switching."

† See the section "Double-High-Resolution Graphics," earlier in this chapter.

Display Mode Switching

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

Table 2-10 shows the reserved locations for the soft switches that control the display modes. For example, to switch from mixed-mode to full-screen graphics in an assembly-language program, you could use the instruction

STA \$C052

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

Poke 49234, 0

Some of the soft switches in Table 2-10 must be read, some must be written to, and for some you can use either action. When writing to a soft switch, it doesn't matter what value you write; the action occurs when you address the location, and the value is ignored.

Table 2-10. Display Soft Switches

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

Name	Action	Hex	Function
ALTCHAR	W	\$C00E	Off: display text using primary character set
ALTCHAR	W	\$C00F	On: display text using alternate character set
RDALTCHAR	R7	\$C01E	Read ALTCHAR switch (1 = on)
80COL	W	\$C00C	Off: display 40 columns
80COL	W	\$C00D	On: display 80 columns
RD80COL	R7	\$C01F	Read 80COL switch (1 = on)
80STORE	W	\$C000	Off: cause PAGE2 on to select auxiliary RAM
80STORE	W	\$C001	On: allow PAGE2 to switch main RAM areas
RD80STORE	R7	\$C018	Read 80STORE switch (1 = on)
PAGE2	R/W	\$C054	Off: select Page 1
PAGE2	R/W	\$C055	On: select Page 2 or, if 80STORE on, Page 1 in auxiliary memory
RDPAGE2	R7	\$C01C	Read PAGE2 switch (1 = on)
TEXT	R/W	\$C050	Off: display graphics or, if MIXED on, mixed
TEXT	R/W	\$C051	On: display text
RDTEXT	R7	\$C01A	Read TEXT switch (1 = on)
MIXED	R/W	\$C052	Off: display only text or only graphics
MIXED	R/W	\$C053	On: if TEXT off, display text and graphics
RDMIXED	R7	\$C01B	Read MIXED switch (1 = on)
HIRES	R/W	\$C056	Off: if TEXT off, display low-resolution graphics
HIRES	R/W	\$C057	On: if TEXT off, display high-resolution or, if DHIRES on, double-high-resolution graphics
RDHIRES	R7	\$C01D	Read HIRES switch (1 = on)
IOUDIS	W	\$C07E	On: disable IOU access for addresses \$C058 to \$C05F; enable access to DHIRES switch *
IOUDIS	W	\$C07F	Off: enable IOU access for addresses \$C058 to \$C05F; disable access to DHIRES switch *
RDIUDIS	R7	\$C07E	Read IOUDIS switch (1 = off) †
DHIRES	R/W	\$C05E	On: if IOUDIS on, turn on double-high-res.
DHIRES	R/W	\$C05F	Off: if IOUDIS on, turn off double-high-res.
RDDHIRES	R7	\$C07F	Read DHIRES switch (1 = on) †

* The firmware normally leaves IOUDIS on. See also †.

† Reading or writing any address in the range \$C070-\$C07F also triggers the paddle timer and resets VBLINT (Chapter 7).

By the Way: You may not need to deal with these functions by reading and writing directly to the memory locations in Table 2-10. Many of the functions shown here are selected automatically if you use the display routines in the various high-level languages on the Apple IIe.

Any time you read a soft switch, you get a byte of data. However, the only information the byte contains is the state of the switch, and this occupies only one bit—bit 7, the high-order bit. The other bits in the byte are unpredictable. If you are programming in machine language, the switch setting is the sign bit; as soon as you read the byte, you can do a Branch Plus if the switch is off, or Branch Minus if the switch is on.

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

Addressing Display Pages Directly

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

For a full description of the way the Apple IIe handles its display memory, refer to the section "Display Memory Addressing" in Chapter 7.

The display memory maps are shown in Figures 2-5, 2-6, 2-7, 2-8, and 2-9. All of the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes (\$80 hexadecimal). By folding the display data into memory this way, the Apple IIe, like the Apple II, stores all 960 characters of displayed text within 1K bytes of memory.



The high-resolution graphics display is stored in much the same way as text, but there are eight times as many bytes to store, because eight rows of dots occupy the same space on the display as one row of characters. The subset consisting of all the first rows from the groups of eight is stored in the first 1024 bytes of the high-resolution display page. The subset consisting of all the second rows from the groups of eight is stored in the second 1024 bytes, and so on for a total of 8 times 1024, or 8192 bytes. In other words, each block of 1024 bytes in the high-resolution display page contains one row of dots out of every group of eight rows. The individual rows are stored in sets of three 40-byte rows, the same way as the text display.

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

The 80-column display and double-high-resolution graphics mode work a little differently. Half of the data is stored in the normal text Page-1 memory, and the other half is stored in memory on the 80-column text card using the same addresses. The display circuitry fetches bytes from these two memory areas simultaneously and displays them sequentially: first the byte from the 80-column text card memory, then the byte from the main memory. The main memory stores the characters in the odd columns of the display, and the 80-column text card memory stores the characters in the even columns.

For more details about the way the displays are generated, see Chapter 7.

To store display data on the 80-column text card, first turn on the 80STORE soft switch by writing to location 49153 (hexadecimal \$C001 or complementary -16383). With 80STORE on, the page-select switch, PAGE2, selects between the portion of the 80-column display stored in Page 1 of main memory and the portion stored in the 80-column text card memory. To select the 80-column text card, turn the PAGE2 soft switch on by reading or writing at location 49237.

Figure 2-5. Map of 40-Column Text Display

Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39				
0	\$400	1024																																										
1	\$480	1152																																										
2	\$500	1280																																										
3	\$580	1408																																										
4	\$600	1536																																										
5	\$680	1664																																										
6	\$700	1792																																										
7	\$780	1920																																										
8	\$428	1064																																										
9	\$4A8	1192																																										
10	\$528	1320																																										
11	\$5A8	1448																																										
12	\$628	1576																																										
13	\$6A8	1704																																										
14	\$728	1832																																										
15	\$7A8	1960																																										
16	\$450	1104																																										
17	\$4D0	1232																																										
18	\$550	1360																																										
19	\$5D0	1488																																										
20	\$650	1616																																										
21	\$6D0	1744																																										
22	\$750	1872																																										
23	\$7D0	2000																																										

Figure 2-6. Map of 80-Column Text Display

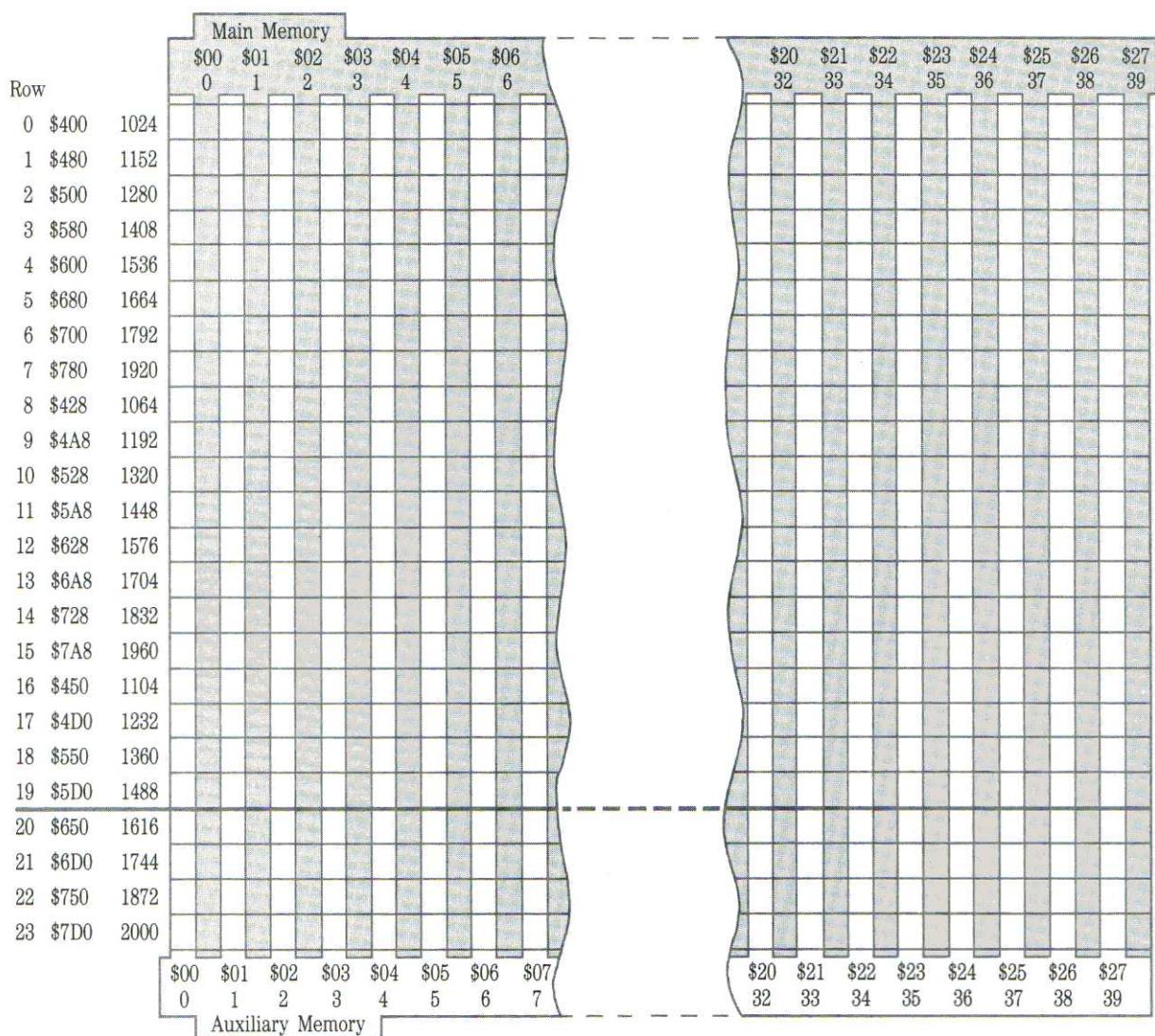


Figure 2-7. Map of Low-Resolution Graphics Display

Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39				
0	\$400	1024																																										
2	\$480	1152																																										
4	\$500	1280																																										
6	\$580	1408																																										
8	\$600	1536																																										
10	\$680	1664																																										
12	\$700	1792																																										
14	\$780	1920																																										
16	\$428	1064																																										
18	\$4A8	1192																																										
20	\$528	1320																																										
22	\$5A8	1448																																										
24	\$628	1576																																										
26	\$6A8	1704																																										
28	\$728	1832																																										
30	\$7A8	1960																																										
32	\$450	1104																																										
34	\$4D0	1232																																										
36	\$550	1360																																										
38	\$5D0	1488																																										
40	\$650	1616																																										
42	\$6D0	1744																																										
44	\$750	1872																																										
46	\$7D0	2000																																										

Figure 2-8. Map of High-Resolution Graphics Display

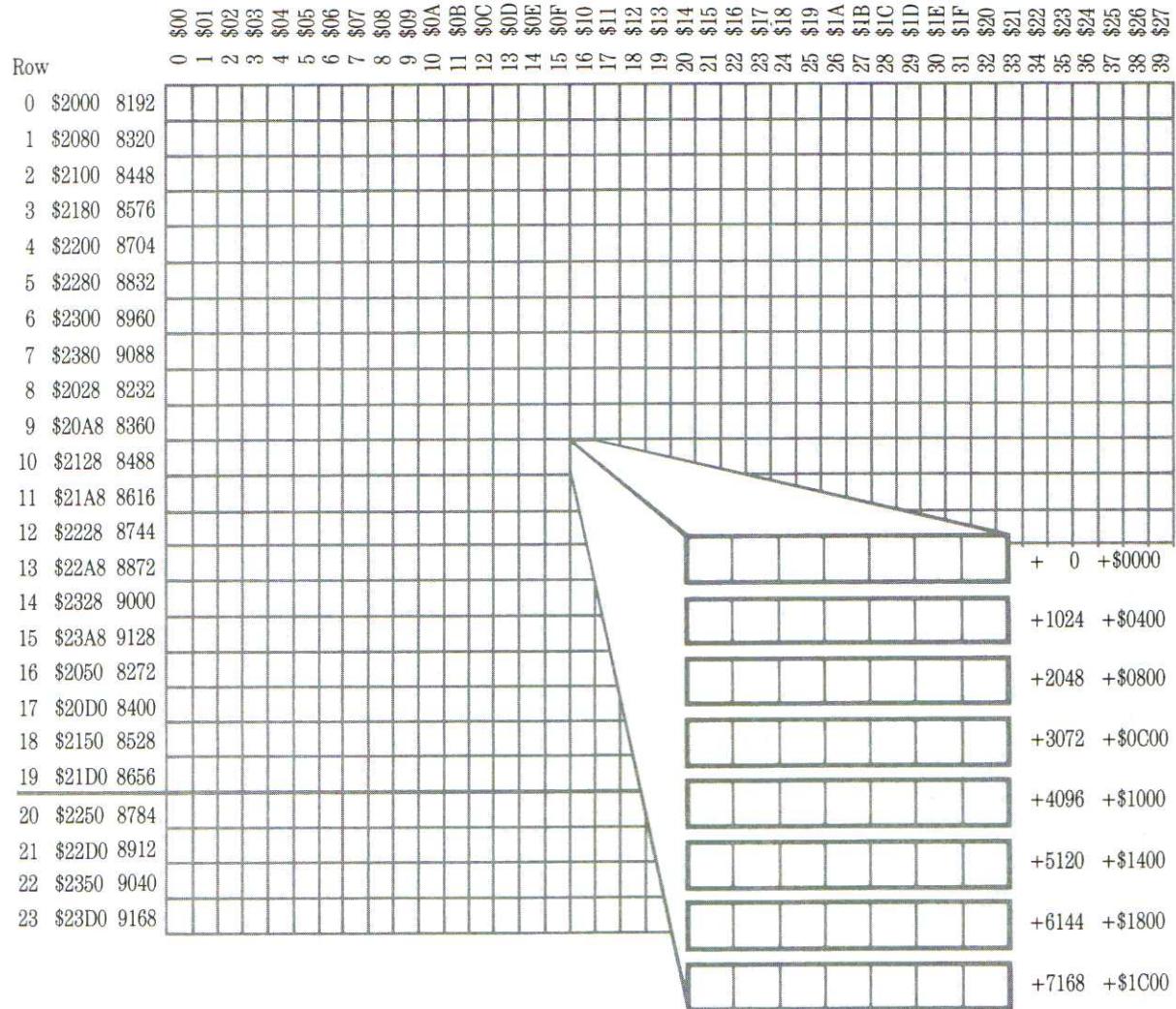


Figure 2-9. Map of Double-High-Resolution Graphics Display



Secondary Inputs and Outputs

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

- the speaker (output)
- cassette input and output
- annunciator outputs
- strobe output
- switch inputs
- analog (hand control) inputs.

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

Important!

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

The Speaker

Electrical specifications of the speaker circuit appear in Chapter 7.

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

If you switch the speaker once, it emits a click; to make longer sounds, you access the speaker repeatedly. You should always use a read operation to toggle the speaker. If you write to this soft switch, it switches twice in rapid succession. The resulting pulse is so short that the speaker doesn't have time to respond; it doesn't make a sound.

BELL1 is described in Appendix B.

The soft switch for the speaker uses memory location 49200 (hexadecimal \$C030). From Integer BASIC, use the complementary address -16336. You can make various tones and buzzes with the speaker by using combinations of timing loops in your program. There is also a routine in the built-in firmware to make a beep through the speaker. This routine is called BELL1.

Detailed electrical specifications for the cassette input and output are given in Chapter 7.

WRITE is described in Appendix B.

Cassette Input and Output

There are two miniature phone jacks on the back panel of the Apple IIe. You can use a pair of standard cables with miniature phone plugs to connect an ordinary cassette tape recorder to the Apple IIe and save programs and data on audio cassettes.

The phone jack marked with a picture of an arrow pointing towards a cassette is the output jack. It is connected to a toggled soft switch, like the speaker switch described above. The signal at the phone jack switches from zero to 25 millivolts or from 25 millivolts to zero each time you access the soft switch.

If you connect a cable from this jack to the microphone input of a cassette tape recorder and switch the recorder to record mode, the signal changes you produce by accessing this soft switch will be recorded on the tape. The cassette output switch uses memory location 49184 (hexadecimal \$C020; complementary value -16352). Like the speaker, this output will toggle twice if you write to it, so you should only use read operations to control the cassette output.

The standard method for writing computer data on audio tapes uses tones with two different pitches to represent the binary states zero and one. To store data, you convert the data into a stream of bits and convert the bits into the appropriate tones. To save you the trouble of actually programming the tones, and to ensure consistency among all Apple II cassette tapes, there is a built-in routine called WRITE for producing cassette data output.

READ is described in Appendix B.

The phone jack marked with a picture of an arrow coming from a cassette is the input jack. It accepts a cable from the cassette recorder's earphone jack. The signal from the cassette is 1 volt (peak-to-peak) audio. Each time the instantaneous value of this audio signal changes from positive to negative, or vice-versa, the state of the cassette input circuit changes from zero to one or vice-versa. You can read the state of this circuit at memory location 49248 (hexadecimal \$C060, or complementary decimal -16288).

When you read this location, you get a byte, but only the high-order bit (bit 7) is valid. If you are programming in machine language, this is the sign bit, so you can perform a Branch Plus or Branch Minus immediately after reading this byte. BASIC is too slow to keep up with the audio tones used for data recording on tape, but you don't need to write the program: there is a built-in routine called READ for reading data from a cassette.

The Hand Control Connector Signals

Several inputs and outputs are available on a 9-pin D-type miniature connector on the back of the Apple IIe: three one-bit inputs, or switches, and four analog inputs. These signals are also available on the 16-pin IC connector on the main circuit board, along with four one-bit outputs and a data strobe. You can access all of these signals from your programs.

Ordinarily, you connect a pair of hand controls to the 9-pin connector. The rotary controls use two analog inputs, and the push-buttons use two one-bit inputs. However, you can also use these inputs and outputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick. Table 7-19 shows the connector pin numbers.

For electrical specifications of the annunciator outputs, refer to Chapter 7.

Annunciator Outputs

The four one-bit outputs are called annunciators. Each annunciator can be used to turn a lamp, a relay, or some similar electronic device on and off.

Each annunciator is controlled by a soft switch, and each switch uses a pair of memory locations. These memory locations are shown in Table 2-11. Any reference to the first location of a pair turns the corresponding annunciator off; a reference to the second location turns the annunciator on. There is no way to read the state of an annunciator.

Table 2-11. Annunciator Memory Locations

No.	Annunciator Pin*	State	Address		Hex
			Decimal	Hex	
0	15	off	49240	-16296	\$C058
		on	49241	-16295	\$C059
1	14	off	49242	-16294	\$C05A
		on	49243	-16293	\$C05B
2	13	off	49244	-16292	\$C05C
		on	49245	-16291	\$C05D
3	12	off	49246	-16290	\$C05E
		on	49247	-16289	\$C05F

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

Strobe Output

The strobe output is normally at +5 volts, but it drops to zero for about half a microsecond any time its dedicated memory location is accessed. You can use this signal to control functions such as data latching in external devices. If you use this signal, remember that memory is addressed twice by a write; if you need only a single pulse, use a read operation to activate the strobe. The memory location for the strobe signal is 49216 (hexadecimal \$C040 or complementary -16320).

Switch Inputs

The three one-bit inputs can be connected to the output of another electronic device or to a pushbutton. When you read a byte from one of these locations, only the high-order bit—bit 7—is valid information; the rest of the byte is undefined. From machine language, you can do a Branch Plus or Branch Minus on the state of bit 7. From BASIC, you read the switch with a PEEK and compare the value with 128. If the value is 128 or greater, the switch is on.

The memory locations for these switches are 49249 through 49251 (hexadecimal \$C061 through \$C063, or complementary -16287 through -16285), as shown in Table 2-12. Switch 0 and switch 1 are permanently connected to the **C** and **B** keys on the keyboard; these are the ones normally connected to the buttons on the hand controls. Some software for the older models of the Apple II uses the third switch, switch 2, as a way of detecting the shift key. This technique requires a hardware modification known as the single-wire shift-key mod.

You should be sure that you really need the shift-key mod before you go ahead and do it. It probably is not worth it unless you have a program that requires the shift-key mod that you cannot either replace or modify to work without it.

▲Warning

If you make the shift-key modification and connect a joystick or other hand control that uses switch 2, you must be careful never to close the switch and press **SHIFT** at the same time: doing so produces a short circuit that causes the power supply to turn off. When this happens, any programs or data in the computer's internal memory are lost.

Shift-Key Mod: To perform this modification on your Apple IIe, all you have to do is solder across the broken diamond labelled X6 on the main circuit board. Remember to turn off the power before changing anything inside the Apple IIe. Also remember that changes such as this are at your own risk and may void your warranty.

Refer to the section "Game I/O Signals" in Chapter 7 for details.

PREAD is described in Appendix B.

Analog Inputs

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

Before a program can read the analog inputs, it must first reset the timing circuits. Accessing memory location 49264 (hexadecimal \$C070 or complementary -16272) does this. As soon as you reset the timing circuits, the high bits of the bytes at locations 49252 through 49255 (hexadecimal \$C064 through \$C067 or complementary -16284 through -16281) are set to 1. If you PEEK at them from BASIC, the values will be 128 or greater. Within about 3 milliseconds, these bits will change back to 0—byte values less than 128—and remain there until you reset the timing circuits again. The exact time each of the four bits remains high is directly proportional to the resistance connected to the corresponding input. If these inputs are open—no resistances are connected—the corresponding bits may remain high indefinitely.

To read the analog inputs from machine language, you can use a program loop that resets the timers and then increments a counter until the bit at the appropriate memory location changes to 0, or you can use the built-in routine called PREAD. High-level languages, such as BASIC, also include convenient means of reading the analog inputs: refer to your language manuals.

Summary of Secondary I/O Locations

Table 2-12 shows the memory locations for all of the built-in I/O devices except the keyboard and display. As explained earlier, some soft switches should only be accessed by means of read operations; those switches are marked.

Table 2-12. Secondary I/O Memory Locations

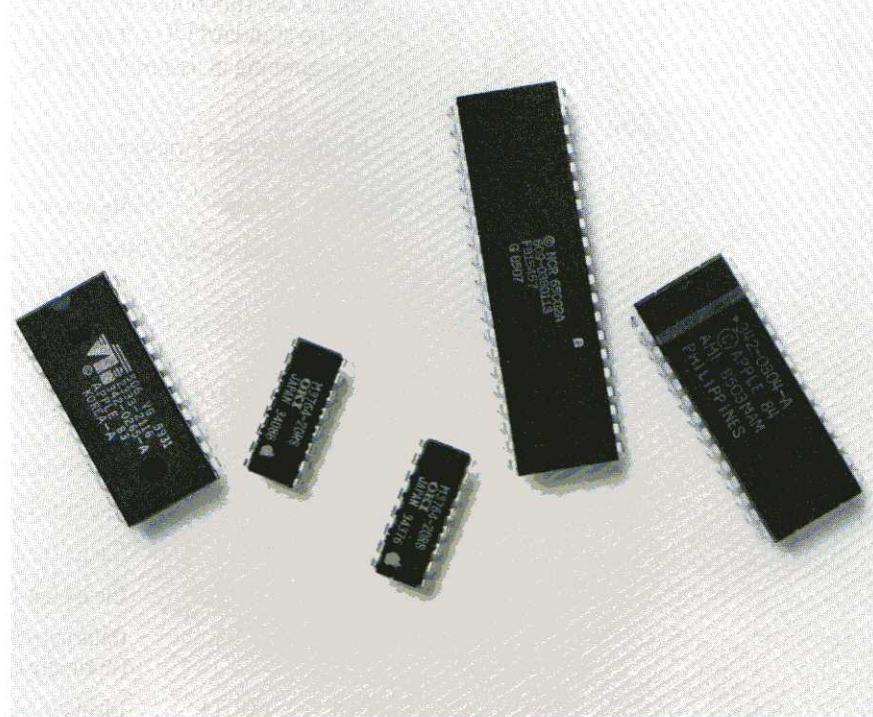
For connector identification and pin numbers, refer to Tables 7-18 and 7-19.

Function	Address			
	Decimal	Hex	Access	
Speaker	49200	-16336	\$C030	Read only
Cassette out	49184	-16352	\$C020	Read only
Cassette in	49248	-16288	\$C060	Read only
Annunciator 0 on	49241	-16295	\$C059	
Annunciator 0 off	49240	-16296	\$C058	
Annunciator 1 on	49243	-16293	\$C05B	
Annunciator 1 off	49242	-16294	\$C05A	
Annunciator 2 on	49245	-16291	\$C05D	
Annunciator 2 off	49244	-16292	\$C05C	
Annunciator 3 on	49247	-16289	\$C05F	
Annunciator 3 off	49246	-16290	\$C05E	
Strobe output	49216	-16320	\$C040	Read only
Switch input 0 (▲)	49249	-16287	\$C061	Read only
Switch input 1 (▼)	49250	-16286	\$C062	Read only
Switch input 2	49251	-16285	\$C063	Read only
Analog input reset	49264	-16272	\$C070	
Analog input 0	49252	-16284	\$C064	Read only
Analog input 1	49253	-16283	\$C065	Read only
Analog input 2	49254	-16282	\$C066	Read only
Analog input 3	49255	-16281	\$C067	Read only



Chapter 3

Built-in I/O Firmware



The **Monitor**, or System Monitor, is a computer program that is used to operate the computer at the machine language level.

Important!

Almost every program on the Apple IIe takes input from the keyboard and sends output to the display. The **Monitor** and the Applesoft and Integer BASICs do this by means of standard I/O subroutines that are built into the Apple IIe's firmware. Many application programs also use the standard I/O subroutines, but Pascal programs do not; Pascal has its own I/O subroutines.

This chapter describes the features of these subroutines as they are used by the Monitor and by the BASIC interpreters, and tells you how to use the standard subroutines in your assembly-language programs.

High-level languages already include convenient methods for handling most of the functions described in this chapter. You should not need to use the standard I/O subroutines in your programs unless you are programming in assembly language.

Table 3-1. Monitor Firmware Routines

Location	Name	Description
\$C305	BASICIN	With 80-column firmware active, displays solid, blinking cursor. Accepts character from keyboard.
\$C307	BASICOUT	Displays a character on the screen; used when the 80-column firmware is active (Chapter 3).
\$FC9C	CLREOL	Clears to end of line from current cursor position.
\$FC9E	CLEOLZ	Clears to end of line using contents of Y register as cursor position.
\$FC42	CLREOP	Clears to bottom of window.
\$F832	CLRSCR	Clears the low-resolution screen.
\$F836	CLRTOP	Clears top 40 lines of low-resolution screen.
\$FDED	COUT	Calls output routine whose address is stored in CSW (normally COUT1, Chapter 3).
\$FDF0	COUT1	Displays a character on the screen (Chapter 3).
\$FD8E	CROUT	Generates a carriage return character.
\$FD8B	CROUT1	Clears to end of line, then generates a carriage return character.
\$FD6A	GETLN	Displays the prompt character; accepts a string of characters by means of RDKEY.
\$F819	HLINE	Draws a horizontal line of blocks.
\$FC58	HOME	Clears the window and puts cursor in upper-left corner of window.

Table 3-1—Continued. Monitor Firmware Routines

Location	Name	Description
\$FD1B	KEYIN	With 80-column firmware inactive, displays checkerboard cursor. Accepts character from keyboard.
\$F800	PLOT	Plots a single low-resolution block on the screen.
\$F94A	PRBL2	Sends 1 to 256 blank spaces to the output device.
\$FDDA	PRBYTE	Prints a hexadecimal byte.
\$FF2D	PRERR	Sends ERR and Control-G to the output device.
\$FDE3	PRHEX	Prints 4 bits as a hexadecimal number.
\$F941	PRNTAX	Prints contents of A and X in hexadecimal.
\$FD0C	RDKEY	Displays blinking cursor; goes to standard input routine, normally KEYIN or BASICIN.
\$F871	SCRN	Reads color value of a low-resolution block.
\$F864	SETCOL	Sets the color for plotting in low-resolution.
\$FC24	VTABZ	Sets cursor vertical position.
\$F828	VLINE	Draws a vertical line of low-resolution blocks.

AUXMOVE and XFER are described in the section "Auxiliary-Memory Subroutines" in Chapter 4.

The standard I/O subroutines listed in Table 3-1 are fully described in this chapter. The Apple IIe firmware also contains many other subroutines that you might find useful. Those subroutines are described in Appendix B. Two of the built-in subroutines, AUXMOVE and XFER, can help you use the optional auxiliary memory.

Using the I/O Subroutines

Before you use the standard I/O subroutines, you should understand a little about the way they are used. The Apple IIe firmware operates differently when an option such as an 80-column text card is used. This section describes general situations that affect the operation of the standard I/O subroutines. Specific instances are described in the sections devoted to the individual subroutines.

Apple II Compatibility

Compared to older Apple II models, the Apple IIe has some additional keyboard and display features. To run programs that were written for the older models, you can make the Apple IIe resemble an Apple II Plus by turning those features off. The features that you can turn off and on to put the Apple IIe into and out of Apple II mode are listed in Table 3-2.

Table 3-2. Apple II Mode

	Apple IIe	Apple II Mode
Keyboard	Uppercase and lowercase	Uppercase only
Display characters	Inverse and normal only	Flashing, inverse, and normal
Display size	40-column; also 80-column with optional card	40-column only

If the Apple IIe does not have an 80-column text card installed in the auxiliary slot, it is almost in Apple II mode as soon as you turn it on or reset it. One exception is the keyboard, which is both uppercase and lowercase.

Original IIe

On an original Apple IIe, DOS 3.3 commands and statements in Integer BASIC and Applesoft must be typed in uppercase letters. To be compatible with older software, you should switch the Apple IIe keyboard to uppercase by pressing **CAPS LOCK**.

Another feature that is different on the Apple IIe as compared to the Apple II is the displayed character set. An Apple II displays only uppercase characters, but it displays them three ways: normal, inverse, and flashing. The Apple IIe can display uppercase characters all three ways, and it can display lowercase characters in the normal way. This combination is called the *primary character set*. When the Apple IIe is first turned on or reset, it displays the primary character set.

The Apple IIe has another character set, called the *alternate character set*, that displays a full set of normal and inverse characters, with the inverse uppercase characters between \$40 and \$5F replaced on enhanced Apple IIe's with MouseText characters.

Original IIe

In the original Apple IIe, uppercase inverse characters appear in place of the MouseText characters of the enhanced Apple IIe and the Apple IIC.

You can switch character sets at any time by means of the ALTCHAR soft switch.

The primary and alternate character sets are described in Chapter 2 in the section "Text Character Sets."

The ALTCHAR soft switch is described in Chapter 2.

The 80-Column Firmware

There are a few features that are normally available only with the optional 80-column display. These features are identified in Table 3-3b and Table 3-6. The firmware that supports these features is built into the Apple IIe, but it is normally active only if an 80-column text card is installed in the auxiliary slot.

When you turn on power or reset the Apple IIe, the 80-column firmware is inactive and the Apple IIe displays the primary character set, even if an 80-column text card is installed. When you activate the 80-column firmware, it switches to the alternate character set.

The built-in 80-column firmware is implemented as if it were installed in expansion slot 3. Programs written for an Apple II or Apple II Plus with an 80-column text card installed in slot 3 usually will run properly on a Apple IIe with an 80-column text card in the auxiliary slot.

If the Apple IIe has an 80-column text card and you want to use the 80-column display, you can activate the built-in firmware from BASIC by typing

PR#3

To activate the 80-column firmware from the Monitor, press **[3]**, then **CONTROL-[P]**. Notice that this is the same procedure you use to activate a card in expansion slot 3. Any card installed in the auxiliary slot takes precedence over a card installed in expansion slot 3:

Important!

Even though you activated the 80-column firmware by typing PR#3, you should never deactivate it by typing PR#0, because that just disconnects the firmware, leaving several soft switches still set for 80-column operation. Instead, type the sequence **ESC CONTROL-[Q]**. (See Table 3-6.)

If there is no 80-column text card or other auxiliary memory card in your Apple IIe, you can still activate the 80-column firmware and use it with a 40-column display. First, set the SLOTC3ROM soft switch located at \$C00A (49162). Then type PR#3 to transfer control to the firmware.

When the 80-column firmware is active without a card in the auxiliary slot, it does not work quite the same as it does with a card. The functions that clear the display (CLREOL, CLEOLZ, CLREOP, and HOME) work as if the firmware were inactive: they always clear to the current color. Also, interrupts are supported only with a card installed in the auxiliary slot.

See the section "Switching I/O Memory" in Chapter 6 for details.

SLOTC3ROM is described in Chapter 6 in the section "Switching I/O Memory."

For more information about interrupts, see Chapter 6.

▲Warning

If you do not have an interface card in either the auxiliary slot or slot 3, don't try to activate the firmware with PR#3. Typing PR#3 with no card installed transfers control to the empty connector, with unpredictable results.

Programs activate the 80-column firmware by transferring control to address \$C300. If there is no card in the auxiliary slot, you must set the SLOT C3 ROM soft switch first. To deactivate the 80-column firmware from a program, write a Control-U character via subroutine COUT.

The Old Monitor

Apple II's and Apple II Pluses used a version of the System Monitor different from the one the Apple IIe uses. It had the same standard I/O subroutines, but a few of the features were different; for example, there were no arrow keys for cursor motion. If you start the Apple IIe with a DOS or BASIC disk that loads Integer BASIC into the bank-switched area in RAM, the old Monitor (sometimes called the Autostart Monitor) is also loaded with it. When you type INT from Applesoft to activate Integer BASIC, you also activate this copy of the old Monitor, which remains active until you either type FP to switch back to Applesoft, which uses the new Monitor in ROM, or type

PR#3

to activate the 80-column firmware. Part of the firmware's initialization procedure checks to see which version of the Monitor is in RAM. If it finds the old Monitor, it replaces it with a copy of the new Monitor from ROM. After the firmware has copied the new Monitor into RAM, it remains there until the next time you start up the system.

The Standard I/O Links

When you call one of the character I/O subroutines (COUT and RDKEY), the first thing that happens is an indirect jump to an address stored in programmable memory. Memory locations used for transferring control to other subroutines are sometimes called vectors; in this manual, the locations used for transferring control to the I/O subroutines are called **I/O links**. In an Apple IIe running without a disk operating system, each I/O link is normally the address of the body of the subroutine (COUT1 or KEYIN). If a disk operating system is running, one or both of these links hold the addresses of the corresponding DOS or ProDOS I/O routines instead. (DOS and ProDOS maintain their own links to the standard I/O subroutines.)

By calling the I/O subroutines that jump to the link addresses instead of calling the standard subroutines directly, you ensure that your program will work properly in conjunction with other software, such as DOS or a printer driver, that changes one or both of the I/O links.

For more information about the I/O links, see the section "Changing the Standard I/O Links" in Chapter 6.

For the purposes of this chapter, we shall assume that the I/O links contain the addresses of the standard I/O subroutines—COUT1 and KEYIN if the 80-column firmware is off, and BASICOUT and BASICIN if it is on.

Standard Output Features

The standard output routine is named COUT, pronounced C-out, which stands for *character out*. COUT normally calls COUT1, which sends one character to the display, advances the cursor position, and scrolls the display when necessary. COUT1 restricts its use of the display to an active area called the text window, described below.

COUT Output Subroutine

Your program makes a subroutine call to COUT at memory location \$FDED with a character in the accumulator. COUT then passes control via the output link CSW to the current output subroutine, normally COUT1 (or BASICOUT), which takes the character in the accumulator and writes it out. If the accumulator contains an uppercase or lowercase letter, a number, or a special character, COUT1 displays it; if the accumulator contains a control character, COUT1 either performs one of the special functions described below or ignores the character.

Each time you send a character to COUT1, it displays the character at the current cursor position, replacing whatever was there, and then advances the cursor position one space to the right. If the cursor position is already at the right-hand edge of the window, COUT1 moves it to the left-most position on the next line down. If this would move the cursor position past the end of the last line in the window, COUT1 scrolls the display up one line and sets the cursor position at the left end of the new bottom line.

The cursor position is controlled by the values in memory locations 36 and 37 (hexadecimal \$24 and \$25). These locations are named CH, for cursor horizontal, and CV, for cursor vertical. COUT1 does not display a cursor, but the input routines described below do, and they use this cursor position. If some other routine displays a cursor, it will not necessarily put it in the cursor position used by COUT1.

Control Characters With COUT1 and BASICOUT

COUT1 and BASICOUT do not display control characters. Instead, the control characters listed in Tables 3-3a and 3-3b are used to initiate some action by the firmware. Other control characters are ignored. Most of the functions listed here can also be invoked from the keyboard, either by typing the control character listed or by using the appropriate escape code, as described in the section “Escape Codes With KEYIN” later in this chapter. The stop-list function, described separately, can only be invoked from the keyboard.

Table 3-3a. Control Characters With 80-Column Firmware Off

Control Character	ASCII Name	Apple IIe Name	Action Taken by COUT1
Control-G	BEL	bell	Produces a 1000 Hz tone for 0.1 second.
Control-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.
Control-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed.
Control-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed.

Table 3-3b. Control Characters With 80-Column Firmware On

Control Character	ASCII Name	Apple IIe Name	Action Taken by BASICOUT
Control-G	BEL	bell	Produces a 1000 Hz tone for 0.1 second.
Control-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.
Control-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed.
Control-K†	VT	clear EOS	Clears from cursor position to the end of the screen.
Control-L†	FF	home and clear	Moves cursor position to upper-left corner of window and clears window.

Table 3-3b—Continued. Control Characters With 80-Column Firmware On

Control Character	ASCII Name	Apple IIe Name	Action Taken by BASICOUT
Control-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed.
Control-N†	SO	normal	Sets display format normal.
Control-O†	SI	inverse	Sets display format inverse.
Control-Q†	DC1	40-column	Sets display to 40-column.
Control-R†	DC2	80-column	Sets display to 80-column.
Control-S*	DC3	stop-list	Stops listing characters on the display until another key is pressed.
Control-U †	NAK	quit	Deactivates 80-column video firmware.
Control-V †	SYN	scroll	Scrolls the display down one line, leaving the cursor in the current position.
Control-W †	ETB	scroll-up	Scrolls the display up one line, leaving the cursor in the current position.
Control-X	CAN	disable MouseText	Disable MouseText character display; use inverse uppercase.
Control-Y †	EM	home	Moves cursor position to upper-left corner of window (but doesn't clear).
Control-Z †	SUB	clear line	Clears the line the cursor position is on.
Control-[ESC	enable MouseText	Map inverse uppercase characters to MouseText characters.
Control-\†	FS	forward space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below.
Control-]†	GS	clear EOL	Clears from the current cursor position to the end of the line (that is, to the right edge of the window).
Control-_	US	up	Moves cursor up a line, no scroll.

* Only works from the keyboard.

† Doesn't work from the keyboard.

The Stop-List Feature

When you are using any program that displays text via COUT1 (or BASICOUT), you can make it stop updating the display by holding down [CONTROL] and pressing [S]. Whenever COUT1 gets a carriage return from the program, it checks to see if you have pressed [CONTROL]-[S]. If you have, COUT1 stops and waits for you to press another key. When you want COUT1 to resume, press another key; COUT1 will send the carriage return it got earlier to the display, then continue normally. The character code of the key you pressed to resume displaying is ignored unless you pressed [CONTROL]-[C]. COUT1 passes Control-C back to the program; if it is a BASIC program, this enables you to terminate the program while in stop-list mode.

The Text Window

After starting up the computer or after a reset, the firmware uses the entire display. However, you can restrict video activity to any rectangular portion of the display you wish. The active portion of the display is called the text window. COUT1 or BASICOUT puts characters into the window only; when it reaches the end of the last line in the window, it scrolls only the contents of the window.

You can set the top, bottom, left side, and width of the text window by storing the appropriate values into four locations in memory. This enables your programs to control the placement of text in the display and to protect other portions of the screen from being written over by new text.

Memory location 32 (hexadecimal \$20) contains the number of the leftmost column in the text window. This number is normally 0, the number of the leftmost column in the display. In a 40-column display, the maximum value for this number is 39 (hexadecimal \$27); in an 80-column display, the maximum value is 79 (hexadecimal \$4F).

Memory location 33 (hexadecimal \$21) holds the width of the text window. For a 40-column display, it is normally 40 (hexadecimal \$28); for an 80-column display, it is normally 80 (hexadecimal \$50).

Original IIe | COUT1 truncates the column width to an even value on the original Apple IIe.

▲Warning

On an original Apple IIe, be careful not to let the sum of the window width and the leftmost position in the window exceed the width of the display you are using (40 or 80). If this happens, it is possible for COUT1 to put characters into memory locations outside the display page, possibly into your current program or data space.

Memory location 34 (hexadecimal \$22) contains the number of the top line of the text window. This is normally 0, the topmost line in the display. Its maximum value is 23 (hexadecimal \$17).

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

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

▲Warning

Any time you change the boundaries of the text window, you should make sure that the current cursor position (stored at CH and CV) is inside the new window. If it is outside, it is possible for COUT1 to put characters into memory locations outside the display page, possibly destroying programs or data.

Table 3-4 summarizes the memory locations and the possible values for the window parameters.

Table 3-4. Text Window Memory Locations

Window Parameter	Location		Minimum Value		Normal Values				Maximum Values			
	Dec	Hex	Dec	Hex	40 col.	80 col.	Dec	Hex	40 col.	Dec	Hex	80 col.
Left Edge	32	\$20	00	\$00	00	\$00	00	\$00	39	\$27	79	\$4F
Width	33	\$21	00	\$00	40	\$28	80	\$50	40	\$28	80	\$50
Top Edge	34	\$22	00	\$00	00	\$00	00	\$00	23	\$17	23	\$17
Bottom Edge	35	\$23	01	\$01	24	\$18	24	\$18	24	\$18	24	\$18

Inverse and Flashing Text

Subroutine COUT1 can display text in normal format, inverse format, or, with some restrictions, flashing format. The display format for any character in the display depends on two things: the character set being used at the moment, and the setting of the two high-order bits of the character's byte in the display memory.

As it sends your text characters to the display, COUT1 sets the high-order bits according to the value stored at memory location 50 (hexadecimal \$32). If that value is 255 (hexadecimal \$FF), COUT1 sets the characters to display in normal format; if the value is 63 (hexadecimal \$3F), COUT1 sets the characters to inverse format. If the value is 127 (hexadecimal \$7F) and if you have selected the primary character set, the characters will be displayed in flashing format. Note that flashing format is not available in the alternate character set.

Table 3-5. Text Format Control Values

Note: These mask values apply only to the primary character set (see text).

Mask Value		
Dec	Hex	Display Format
255	\$FF	Normal, uppercase, and lowercase
127	\$7F	Flashing, uppercase, and symbols
63	\$3F	Inverse, uppercase, and lowercase

To control the display format of the characters, routine COUT1 uses the value at location 50 as a logical mask to force the setting of the two high-order bits of each character byte it puts into the display page. It does this by performing the logical AND function on the data byte and the mask byte. The result byte contains a 0 in any bit that was 0 in the mask. BASICOUT, used when the 80-column firmware is active, changes only the high-order bit of the data.

Important!

If the 80-column firmware is inactive and you store a mask value at location 50 with zeros in its low-order bits, COUT1 will mask out those bits in your text. As a result, some characters will be transformed into other characters. You should set the mask to the values given in Table 3-5 only.

Switching between character sets is described in the section "Display Mode Switching" in Chapter 2.

Original IIe

If you set the mask value at location 50 to 127 (hexadecimal \$7F), the high-order bit of each result byte will be 0, and the characters will be displayed either as lowercase or as flashing, depending on which character set you have selected. Refer to the tables of display character sets in Chapter 2. In the primary character set, the next-highest bit, bit 6, selects flashing format with uppercase characters. With the primary character set you can display lowercase characters in normal format and uppercase characters in normal, inverse, and flashing formats. In the alternate character set, bit 6 selects lowercase or special characters. With the alternate character set you can display uppercase and lowercase characters in normal and inverse formats.

On the original Apple IIe, the MouseText characters are replaced by uppercase inverse characters.

Standard Input Features

The Apple IIe's firmware includes two different subroutines for reading from the keyboard. One subroutine is named RDKEY, which stands for *read key*. It calls the standard character input subroutine KEYIN (or BASICIN when the 80-column firmware is active) which accepts one character at a time from the keyboard.

The other subroutine is named GETLN, which stands for *get line*. By making repeated calls to RDKEY, GETLN accepts a sequence of characters terminated with a carriage return. GETLN also provides on-screen editing features.

RDKEY Input Subroutine

A program gets a character from the keyboard by making a subroutine call to RDKEY at memory location \$FD0C. RDKEY sets the character at the cursor position to flash, then passes control via the input link KSW to the current input subroutine, which is normally KEYIN or BASICIN.

RDKEY displays a cursor at the current cursor position, which is immediately to the right of whatever character you last sent to the display (normally by using the COUT routine, described earlier). The cursor displayed by RDKEY is a flashing version of whatever character happens to be at that position on the screen. It is usually a space, so the cursor appears as a blinking rectangle.

KEYIN Input Subroutine

KEYIN is the standard input subroutine when the 80-column firmware is inactive; BASICIN is used when the 80-column firmware is active. When called, the subroutine waits until the user presses a key, then returns with the key code in the accumulator.

If the 80-column firmware is inactive, KEYIN displays a cursor by alternately storing a checkerboard block in the cursor location, then storing the original character, then the checkerboard again. If the firmware is active, BASICIN displays a steady inverse space (rectangle), unless you are in escape mode, when it displays a plus sign (+) in inverse format.

KEYIN also generates a random number. While it is waiting for the user to press a key, KEYIN repeatedly increments the 16-bit number in memory locations 78 and 79 (hexadecimal \$4E and \$4F). This number keeps increasing from 0 to 65535, then starts over again at 0. The value of this number changes so rapidly that there is no way to predict what it will be after a key is pressed. A program that reads from the keyboard can use this value as a random number or as a seed for a random number routine.

When the user presses a key, KEYIN accepts the character, stops displaying the cursor, and returns to the calling program with the character in the accumulator.

Escape Codes

KEYIN has special functions that you invoke by typing escape codes on the keyboard. An escape code is obtained by pressing [ESC], releasing it, and then pressing some other key. See Table 3-6; the notation in the table means press [ESC], release it, then press the key that follows.

Table 3-6 includes three sets of cursor-control keys. The first set consists of [ESC] followed by A, B, C, or D. The letter keys can be either uppercase or lowercase. These keys are the standard cursor-motion keys on older Apple II models; they are present on the Apple IIe primarily for compatibility with programs written for old machines.

Cursor Motion in Escape Mode

The second and third set of cursor-control keys are listed together because they activate escape mode. In escape mode, you can keep using the cursor-motion keys without pressing [ESC] again. This enables you to perform repeated cursor moves by holding down the appropriate key.

Escape mode is described in the next section, "Escape Codes."

When the 80-column firmware is active, you can tell when BASICIN is in escape mode: it displays a plus sign in inverse format as the cursor. You leave escape mode by typing any key other than a cursor-motion key.

The escape codes with the directional arrow keys are the standard cursor-motion keys on the Apple IIe. The escape codes with the I, J, K, and M keys are the standard cursor-motion keys on the Apple II Plus, and are present on the Apple IIe for compatibility with the Apple II Plus. On the Apple IIe, the escape codes with the I, J, K, and M keys function with either uppercase or lowercase letters.

Table 3-6. Escape Codes

Escape Code	Function
<code>ESC @</code>	Clears window and homes cursor (places it in upper-left corner of screen), then exits from escape mode.
<code>ESC A</code> or <code>a</code>	Moves cursor right one line; exits from escape mode.
<code>ESC B</code> or <code>b</code>	Moves cursor left one line; exits from escape mode.
<code>ESC C</code> or <code>c</code>	Moves cursor down one line; exits from escape mode.
<code>ESC D</code> or <code>d</code>	Moves cursor up one line; exits from escape mode.
<code>ESC E</code> or <code>e</code>	Clears to end of line; exits from escape mode.
<code>ESC F</code> or <code>f</code>	Clears to bottom of window; exits from escape mode.
<code>ESC I</code> or <code>i</code> or <code>ESC +</code>	Moves the cursor up one line; remains in escape mode. See text.
<code>ESC J</code> or <code>j</code> or <code>ESC ←</code>	Moves the cursor left one space; remains in escape mode. See text.
<code>ESC K</code> or <code>k</code> or <code>ESC →</code>	Moves the cursor right one space; remains in escape mode. See text.
<code>ESC M</code> or <code>m</code> or <code>ESC ↓</code>	Moves the cursor down one line; remains in escape mode. See text.
<code>ESC 4</code>	If 80-column firmware is active, switches to 40-column mode; sets links to BASICIN and BASICOUT; restores normal window size; exits from escape mode.
<code>ESC 8</code>	If 80-column firmware is active, switches to 80-column mode; sets links to BASICIN and BASICOUT; restores normal window size; exits from escape mode.
<code>ESC CONTROL-D</code>	Disables control characters; only carriage return, line feed, BELL, and backspace have an effect when printed.
<code>ESC CONTROL-E</code>	Reactivates control characters.
<code>ESC CONTROL-Q</code>	If 80-column firmware is active, deactivates 80-column firmware; sets links to KEYIN and COUT1; restores normal window size; exits from escape mode.

GETLN Input Subroutine

Programs often need strings of characters as input. While it is possible to call RDKEY repeatedly to get several characters from the keyboard, there is a more powerful subroutine you can use. This routine is named GETLN, which stands for *get line*, and starts at location \$FD6A. Using repeated calls to RDKEY, GETLN accepts characters from the standard input subroutine—usually KEYIN—and puts them into the input buffer located in the memory page from \$200 to \$2FF. GETLN also provides the user with on-screen editing and control features, described in the next section “Editing With GETLN.”

The first thing GETLN does when you call it is display a prompting character, called simply a **prompt**. The prompt indicates to the user that the program is waiting for input. Different programs use different prompt characters, helping to remind the user which program is requesting the input. For example, an INPUT statement in a BASIC program displays a question mark (?) as a prompt. The prompt characters used by the different programs on the Apple IIe are shown in Table 3-7.

GETLN uses the character stored at memory location 51 (hexadecimal \$33) as the prompt character. In an assembly-language program, you can change the prompt to any character you wish. In BASIC, changing the prompt character has no effect, because both BASIC interpreters and the Monitor restore it each time they request input from the user.

Table 3-7. Prompt Characters

Prompt Character	Program Requesting Input
?	User's BASIC program (INPUT statement)
]	Applesoft BASIC (Appendix D)
>	Integer BASIC (Appendix D)
*	Firmware Monitor (Chapter 5)

As you type the character string, GETLN sends each character to the standard output routine—normally COUT1—which displays it at the previous cursor position and puts the cursor at the next available position on the display, usually immediately to the right. As the cursor travels across the display, it indicates the position where the next character will be displayed.

GETLN stores the characters in its buffer, starting at memory location \$200 and using the X register to index the buffer. GETLN continues to accept and display characters until you press [RETURN]; then it clears the remainder of the line the cursor is on, stores the carriage-return code in the buffer, sends the carriage-return code to the display, and returns to the calling program.

The maximum line-length that GETLN can handle is 255 characters. If the user types more than this, GETLN sends a backslash (\) and a carriage return to the display, cancels the line it has accepted so far, and starts over. To warn the user that the line is getting full, GETLN sounds a bell (tone) at every keypress after the 248th.

Important!

In the Apple II and the Apple II Plus, the GETLN routine converts all input to uppercase. GETLN in the Apple IIe does not do this, even in Apple II mode. To get uppercase input for BASIC, use [CAPS LOCK].

Editing With GETLN

Subroutine GETLN provides the standard on-screen editing features used by the BASIC interpreters and the Monitor. For an introduction to editing with these features, refer to the *Applesoft Tutorial*. Any program that uses GETLN for reading the keyboard has these features.

Cancel Line

Any time you are typing a line, pressing [CONTROL]-[X] causes GETLN to cancel the line. GETLN displays a backslash (\) and issues a carriage return, then displays the prompt and waits for you to type a new line. GETLN takes the same action when you type more than 255 characters, as described earlier.

Backspace

When you press [←], GETLN moves its buffer pointer back one space, effectively deleting the last character in its buffer. It also sends a backspace character to routine COUT, which moves the display position and the cursor back one space. If you type another character now, it will replace the character you backspaced over, both on the display and in the line buffer. Each time you press [←], it moves the cursor left and deletes another character, until you reach the beginning of the line. If you then press [←] one more time, you have cancelled the line, and GETLN issues a carriage return and displays the prompt.

Retype

→ has a function complementary to the backspace function. When you press →, GETLN picks up the character at the display position just as if it had been typed on the keyboard. You can use this procedure to pick up characters that you have just deleted by backspacing across them. You can use the backspace and retype functions with the cursor-motion functions to edit data on the display. (See the earlier section "Cursor Motion in Escape Mode.")

Monitor Firmware Support

Table 3-8 summarizes the addresses and functions of the video display support routines the Monitor provides. These routines are described in the subsections that follow.

Table 3-8. Video Firmware Routines

Location	Name	Description
\$C307	BASICOUT	Displays a character on the screen when 80-column firmware is active.
\$FC9C	CLREOL	Clears to end of line from current cursor position.
\$FC9E	CLEOLZ	Clears to end of line using contents of Y register as cursor position.
\$FC42	CLREOP	Clears to bottom of window.
\$F832	CLRSCR	Clears the low-resolution screen.
\$F836	CLRTOP	Clears top 40 lines of low-resolution screen.
\$FD6D	COUT	Calls output routine whose address is stored in CSW (normally COUT1, Chapter 3).
\$FDF0	COUT1	Displays a character on the screen (Chapter 3).
\$FD8E	CROUT	Generates a carriage return character.
\$FD8B	CROUT1	Clears to end of line, then generates a carriage return character.
\$F819	HLINE	Draws a horizontal line of blocks.

Table 3-8—Continued. Video Firmware Routines

Location	Name	Description
\$FC58	HOME	Clears the window and puts cursor in upper-left corner of window.
\$F800	PLOT	Plots a single low-resolution block on the screen.
\$F94A	PRBL2	Sends 1 to 256 blank spaces to the output device whose address is in CSW.
\$FDAA	PRBYTE	Prints a hexadecimal byte.
\$FF2D	PRERR	Sends ERR and Control-G to the output device whose output routine address is in CSW.
\$FDE3	PRHEX	Prints 4 bits as a hexadecimal number.
\$F941	PRNTAX	Prints contents of A and X in hexadecimal.
\$F871	SCRN	Reads color value of a low-resolution block on the screen.
\$F864	SETCOL	Sets the color for plotting in low-resolution.
\$FC24	VTABZ	Sets cursor vertical position. (Setting CV at location \$25 does not change vertical position until a carriage return.)
\$F828	VLINE	Draws a vertical line of low-resolution blocks.

BASICOUT, \$C307

BASICOUT is essentially the same as COUT1—BASICOUT is used instead of COUT1 when the 80-column firmware is active. BASICOUT displays the character in the accumulator on the display screen at the current cursor position and advances the cursor. It places the character using the setting of the inverse mask (location \$32). BASICOUT handles control characters; see Table 3-3b. When it returns control to the calling program, all registers are intact.

CLREOL, \$FC9C

CLREOL clears a text line from the cursor position to the right edge of the window. This routine destroys the contents of A and Y.



CLEOLZ, \$FC9E

CLEOLZ clears a text line to the right edge of the window, starting at the location given by base address BASL, which is indexed by the contents of the Y register. This routine destroys the contents of A and Y.

CLREOP, \$FC42

CLREOP clears the text window from the cursor position to the bottom of the window. This routine destroys the contents of A and Y.

CLRSCR, \$F832

CLRSCR clears the low-resolution graphics display to black. If you call this routine while the video display is in text mode, it fills the screen with inverse-mode at-sign (@) characters. This routine destroys the contents of A and Y.

CLRTOP, \$F836

CLRTOP is the same as CLRSCR, except that it clears only the top 40 rows of the low-resolution display.

COUT, \$FDDE

COUT calls the current character output subroutine. (See the section “COUT Output Subroutine” earlier in this chapter.) The character to be sent to the output device should be in the accumulator. COUT calls the subroutine whose address is stored in CSW (locations \$36 and \$37), which is usually the standard character output subroutine COUT1 (or BASICOUT).

COUT1, \$FDFO

COUT1 displays the character in the accumulator on the display screen at the current cursor position and advances the cursor. It places the character using the setting of the inverse mask (location \$32). It handles these control characters: carriage return, line feed, backspace, and bell. When it returns control to the calling program, all registers are intact.

CROUT, \$FD8E

CROUT sends a carriage return to the current output device.

See the section “Control Characters With COUT1 and BASICOUT,” earlier in this chapter for more information on COUT1.

CROUT1, \$FD8B

CROUT1 clears the screen from the current cursor position to the edge of the text window, then calls CROUT.

HLINE, \$F819

HLINE draws a horizontal line of blocks of the color set by SETCOL on the low-resolution graphics display. Call HLINE with the vertical coordinate of the line in the accumulator, the leftmost horizontal coordinate in the Y register, and the rightmost horizontal coordinate in location \$2C. HLINE returns with A and Y scrambled and X intact.

HOME, \$FC58

HOME clears the display and puts the cursor in the upper-left corner of the screen.

PLOT, \$F800

PLOT puts a single block of the color value set by SETCOL on the low-resolution display screen. Call PLOT with the vertical coordinate of the line in the accumulator, and its horizontal position in the Y register. PLOT returns with the accumulator scrambled, but X and Y intact.

PRBL2, \$F94A

PRBL2 sends from 1 to 256 blanks to the standard output device. Upon entry, the X register should contain the number of blanks to send. If X = \$00, then PRBLANK will send 256 blanks.

PRBYTE, \$FDDA

PRBYTE sends the contents of the accumulator in hexadecimal to the current output device. The contents of the accumulator are scrambled.

PRERR, \$FF2D

PRERR sends the word **ERR**, followed by a bell character, to the standard output device. On return, the accumulator is scrambled.

PRHEX, \$FDE3

PRHEX prints the lower nibble of the byte in the accumulator as a single hexadecimal digit. On return, the contents of the accumulator are scrambled.

PRNTAX, \$F941

PRTAX prints the contents of the A and X registers as a four-digit hexadecimal value. The accumulator contains the first byte printed, and the X register contains the second. On return, the contents of the accumulator are scrambled.

SCRN, \$F871

SCRN returns the color value of a single block on the low-resolution display. Call it with the vertical position of the block in the accumulator and the horizontal position in the Y register. The block's color is returned in the accumulator. No other registers are changed.

SETCOL, \$F864

SETCOL sets the color used for plotting in low-resolution graphics to the value passed in the accumulator. The colors and their values are listed in Table 2-6.

VTABZ, \$FC24

VTABZ sets the cursor vertical position. Unlike setting the position at location \$25, change of cursor position doesn't wait until a carriage return character has been sent.

VLINE, \$F828

VLINE draws a vertical line of blocks of the color set by SETCOL on the low-resolution display. Call VLINE with the horizontal coordinate of the line in the Y register, the top vertical coordinate in the accumulator, and the bottom vertical coordinate in location \$2D. VLINE returns with the accumulator scrambled.

I/O Firmware Support

Apple IIe video firmware conforms to the I/O firmware protocol of Apple II Pascal 1.1. However, it does not support windows other than the full 80-by-24 window in 80-column mode, and the full 40-by-24 window in 40-column mode. The video protocol table is shown in Table 3-9.

Table 3-9. Slot 3 Firmware Protocol Table

Address	Value	Description
\$C30B	\$01	Generic signature byte of firmware cards
\$C30C	\$88	80-column card device signature
\$C30D	\$ii	\$C3ii is entry point of initialization routine (PINIT).
\$C30E	\$rr	\$C3rr is entry point of read routine (PREAD).
\$C30F	\$ww	\$C3ww is entry point of write routine (PWRITE).
\$C310	\$ss	\$C3ss is entry point of the status routine (PSTATUS).

PINIT, \$C30D

PINIT does the following:

- Sets a full 80-column window.
- Sets 80STORE (\$C001).
- Sets 80COL (\$C00D).
- Switches on ALTCHAR (\$C00F).
- Clears the screen; places cursor in upper-left corner.
- Displays the cursor.

PREAD, \$C30E

PREAD reads a character from the keyboard and places it in the accumulator with the high bit cleared. It also puts a zero in the X register to indicate IORESULT = GOOD.

PWRITE, \$C30F

PWRITE should be called after placing a character in the accumulator with its high bit cleared. PWRITE does the following:

- Turns the cursor off.
- If the character in the accumulator is not a control character, turns the high bit on for normal display or off for inverse display, displays it at the current cursor position, and advances the cursor; if at the end of a line, does carriage return but not line feed. (See Table 3-10 for control character functions.)

When PWRITE has completed this, it

- turns the cursor back on (if it was not intentionally turned off)
- puts a zero in the X register (IRESULT = GOOD) and returns to the calling program.

Table 3-10. Pascal Video Control Functions

Control	Hex	Function Performed
E or e	\$05	Turns cursor on (enables cursor display).
F or f	\$06	Turns cursor off (disables cursor display).
G or g	\$07	Sounds bell (beeps).
H or h	\$08	Moves cursor left one column. If cursor was at beginning of line, moves it to end of previous line.
J or j	\$0A	Moves cursor down one row; scrolls if needed.
K or k	\$0B	Clears to end of screen.
L or l	\$0C	Clears screen; moves cursor to upper-left of screen.
M or m	\$0D	Moves cursor to column 0.
N or n	\$0E	Displays subsequent characters in normal video. (Characters already on display are unaffected.)

Table 3-10—Continued. Pascal Video Control Functions

Control-	Hex	Function Performed
O or o	\$0F	Displays subsequent characters in inverse video. (Characters already on display are unaffected.)
V or v	\$16	Scrolls screen up one line; clears bottom line.
W or w	\$17	Scrolls screen down one line; clears top line.
Y or y	\$19	Moves cursor to upper-left (home) position on screen.
Z or z	\$1A	Clears entire line that cursor is on.
\ or \	\$1C	Moves cursor right one column; if at end of line, does Control-M.
} or]	\$1D	Clears to end of the line the cursor is on, including current cursor position; does not move cursor.
^ or 6	\$1E	GOTOxy: initiates a GOTOxy sequence; interprets the next two characters as x+32 and y+32, respectively.
—	\$1F	If not at top of screen, moves cursor up one line.

PSTATUS, \$C310

A program that calls PSTATUS must first put a request code in the accumulator: either a 0, meaning “Ready for output?” or a 1, meaning “Is there any input?” PSTATUS returns with the reply in the carry bit: 0 (No) or 1 (Yes).

PSTATUS returns with a 0 in the X register (IORESULT = GOOD), unless the request was not 0 or 1; then PSTATUS returns with a 3 in the X register (IORESULT = ILLEGAL OPERATION).

