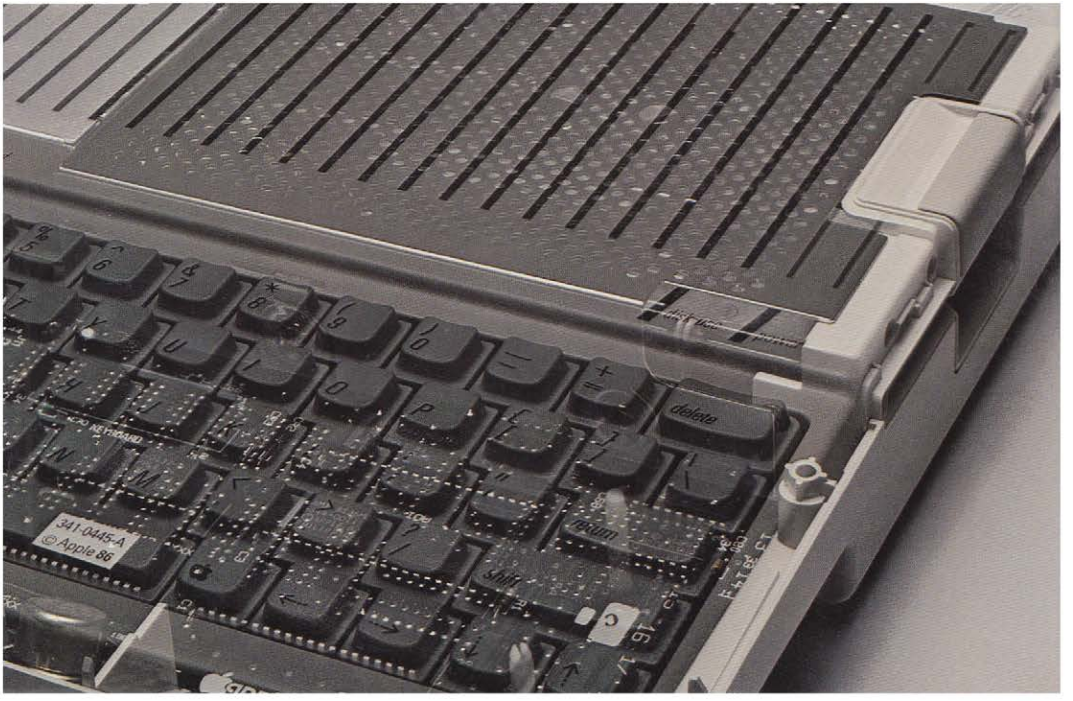
Apple® II

Apple IIc Technical Reference Manual



Includes ROM Listings for Memory Expandable IIc

> $24.95 FPT
USA

Apple® Technical Library Titles for the Apple IIe and IIc

The Official Publications from Apple Computer, Inc.

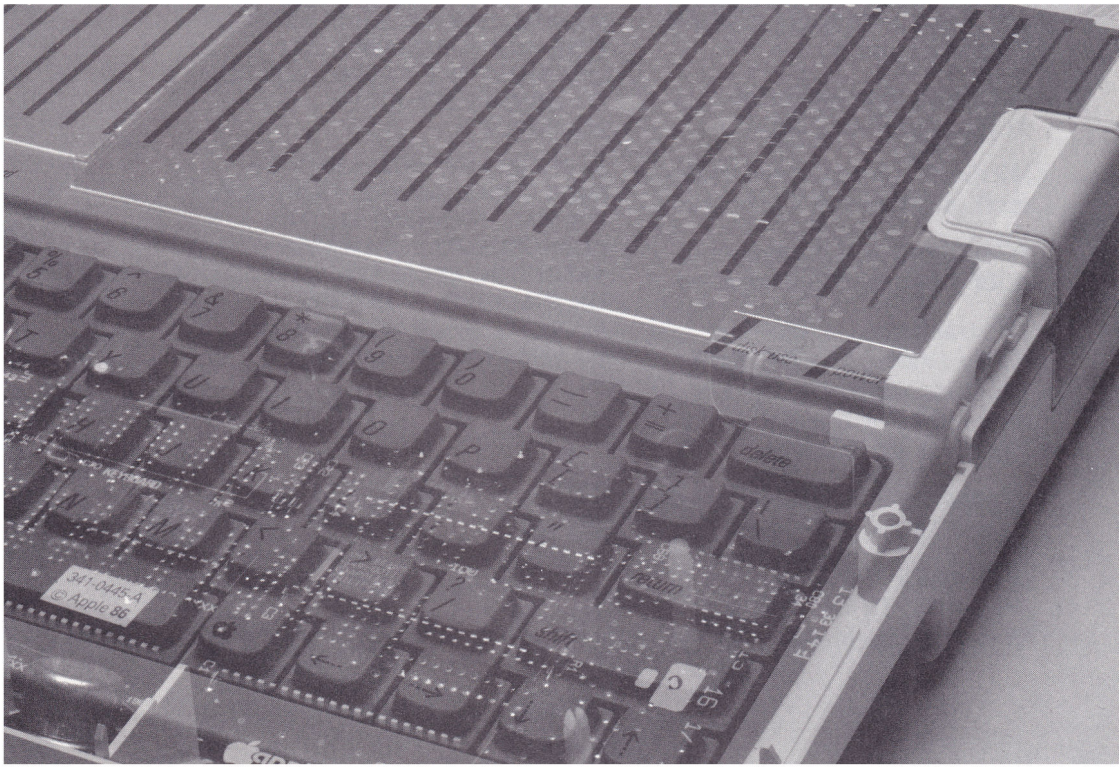
Apple IIe and Apple IIc programmers, developers, and enthusiasts will find a wealth of information in the Apple Technical Library, an ongoing series of comprehensive reference manuals. The first volumes in the Library contained detailed information about the Apple IIe and Apple IIc computers. They describe the hardware, firmware, the ProDOS 8 operating system, and the Applesoft BASIC programming language found in Apple IIe and IIc computers.

These books, written and produced by Apple Computer, Inc., provide definitive references for those interested in getting the most out of their Apple IIe or IIc.

Apple Technical Library Titles for the Apple IIe and IIc include:

- Apple IIe Technical Reference
- Apple IIc Technical Reference
- Applesoft Tutorial
- Applesoft BASIC Programmer's Reference Manual
- ProDOS 8 Technical Reference
- BASIC Programming with ProDOS
- Apple Numerics Manual
- ImageWriter II Technical Reference Manual

Apple® II Apple IIc Technical Reference Manual



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

APPLE COMPUTER, INC.

Copyright © 1984, 1986 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 prior written permission of Apple Computer, Inc. Printed in the United States of America.

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

Macintosh is a trademark of Apple Computer, Inc.

Microsoft is a registered trademark of Microsoft Corporation.

POSTSCRIPT is a trademark of Adobe Systems Incorporated.

ITC Garamond, ITC Avant Garde Gothic, and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation.

Simultaneously published in the United States and Canada.

ISBN 0-201-17752-8
ABCDEFGHIJ-DO-89876
First printing, March 1987

WARRANTY INFORMATION

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

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

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

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

Some states do no allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures and tables xiv

Preface **About This Manual** xxi

Contents of this manual xxi
The Apple IIc family xxiii
Identifying your Apple IIc xxiii
The original Apple IIc xxiv
The UniDisk 3.5 Apple IIc xxiv
The memory expansion Apple IIc xxiv
Conventions used in this manual xxv

Chapter 1 **Introduction** 1

The outside of the machine 2
The keyboard 3
Features 3
Special function keys 4
Cursor movement keys 4
Modifier keys 5
The 80/40 switch 5
The keyboard switch 6
Disk-use and power lights 7
The speaker 8
The built-in disk drive 8
The back panel 9
The inside of the machine 11
The internal voltage converter 11
The main logic board 12
The other circuit boards 15

Chapter 2 Memory Organization and Control 17

The 65C02 microprocessor 18
Overview of the address space 20
Memory map and memory switching 20
Main RAM addresses ($0000–$BFFF and $D000–$FFFF) 22
Auxiliary RAM addresses ($0000–BFFF and $D000–$FFFF) 22
ROM addresses ($C100–$FFFF) 22
Hardware addresses ($C000–$C0FF) 23
Bank-switched memory 24
Page allocations 26
Page $00 (one-byte addresses) 26
Page $01 (the 65C02 stack) 26
Pages $D0–$FF (ROM and RAM) 26
Using bank selector switches 27
48K memory 36
Page allocations 36
Page $02 (the input buffer) 36
Page $03 (global storage and vectors) 36
Pages $04–$07 (text and low-resolution Page 1) 36
Pages $08–$0B (text and low-resolution Page 2) 38
Pages $08 (communication port buffers) 38
Pages $20–$3F (high-resolution Page 1) 38
Pages $40–$5F (high-resolution Page 2) 39
Using 48K memory switches 39
Transfers between main and auxiliary memory 42
Transferring data 42
Transferring control 43
Using display memory switches 44
The reset routine 49
The cold-start procedure (power on) 51
The warm-start procedure (Control-Reset) 51
Forced cold start (Open Apple-Control-Reset) 52
The reset vector 52

Chapter 3 Introduction to Apple IIc I/O 55

The standard I/O links 56
Standard input features 58
RdKey subroutine 58
KeyIn subroutine 58
GetLn subroutine 59
Escape codes with GetLn 60
Editing with GetLn 63
Cancel line 63
Backspace 63
Retype 63

Standard output features 64
COut subroutine 64
Control characters with COut1 65
Control characters with C3COut1 65
The stop-list feature 67
The text window 68
Normal, inverse, and flashing text 69
Primary character set display 70
Alternate character set display 70
Port I/O 71
Standard link entry points 71
Firmware protocol 72
Port I/O space 73
Port ROM space 73
Expansion ROM space 74
Port screen hole RAM space 74
Interrupts 75

Chapter 4 **Keyboard and Speaker 77**

Keyboard input 78
Reading the keyboard 78
Monitor firmware support for keyboard input 82
Speaker output 82
Using the speaker 83
Monitor firmware support for speaker output 84

Chapter 5 **Video Display Output 85**

Video display specifications 87
Text modes 88
Text character sets 88
MouseText 90
40-column versus 80-column text 91
Graphics modes 94
Low-resolution graphics 94
High-resolution graphics 95
Double high-resolution graphics 97
Mixed-mode displays 98
Display pages 99
Display mode switching 101
Display page maps 105
Monitor support for video display output 112
I/O firmware support for video display output 116

Chapter 6 **Block Device I/O 119**

Disk drive I/O 120
Startup 121
    Cold start 121
    Warm start 123
Memory expansion card I/O 123
The Smartport I/O interface 123
Locating the Smartport 124
Issuing a call to the Smartport 125
    Cautions 126
Descriptions of the Smartport calls 126
    STATUS 128
        Parameter descriptions 128
        Possible errors 132
    READ BLOCK 132
        Parameter descriptions 133
        Possible errors 133
    WRITE BLOCK 134
        Parameter descriptions 134
        Possible errors 135
    FORMAT 135
        Parameter descriptions 135
        Possible errors 136
    CONTROL 136
        Parameter descriptions 136
        Possible errors 139
    INIT 139
        Parameter descriptions 140
        Possible errors 140
    OPEN 140
        Parameter descriptions 140
        Possible errors 141
    CLOSE 141
        Parameter descriptions 141
        Possible errors 142
    READ 142
        Parameter descriptions 142
        Possible errors 143
    WRITE 143
        Parameter descriptions 144
        Possible errors 144
        An example: issuing a Smartport call 145
        Summary of commands and parameters 149
        Summary of error codes 150

Chapter 7 Serial I/O Port 1 153

Using serial port 1 155
Characteristics of port 1 at startup 159
Hardware page locations for port 1 159
I/O firmware support for port 1 160
Screen hole locations for port 1 160
Changing port 1 characteristics 161
- Data format and baud rate 163
- Carriage return and line feed 164
- Sending special characters 165
- Displaying output on the screen 165

Chapter 8 Serial I/O Port 2 167

Using serial port 2 169
Characteristics of port 2 at startup 173
Hardware page locations for port 2 173
I/O firmware support for port 2 174
Screen hole locations for port 2 174
Changing port 2 characteristics 176
- Data format and baud rate 177
- Carriage return and line feed 179
- Routing input and output 179
  - Half-duplex operation 180
  - Full-duplex operation 182
  - Terminal mode 184

Chapter 9 Mouse and Game Input 185

Mouse input 186
- Mouse connector signals 187
- Mouse operating modes 187
  - Transparent mode 187
  - Movement interrupt mode 187
  - Button interrupt mode 188
  - Movement/button interrupt mode 188
  - Vertical blanking active modes 188
- Mouse soft switches 189
- I/O firmware support for mouse input 191
  - Pascal support 195
  - BASIC and assembly-language support 195
- Screen holes 196
- Using the mouse as a hand controller 198

Game input 198
The hand controller connector signals 199
Switch inputs (Sw0 and Sw1) 200
Analog inputs (Pdl0 and Pdl1) 200
Monitor support for game input 201

Chapter 10 **Using the Monitor 203**

Invoking the Monitor 204
Syntax of Monitor commands 205
Monitor memory commands 205
Examining memory contents 206
Memory dump 206
Changing memory contents 208
Changing one byte 208
Changing consecutive locations 209
Moving data in memory 210
Comparing data in memory 211
Monitor register commands 212
Changing registers 213
Examining registers 213
Miscellaneous Monitor commands 213
Display inverse and normal 214
Back to BASIC 214
Redirecting input and output 215
Hexadecimal arithmetic 215
Advanced operations 216
Multiple-command lines 216
Filling memory 216
Repeating commands 217
Creating your own commands 218
Machine-language programs 219
Running a program 219
Disassembled programs 220
The STEP and TRACE commands 221
The Mini-Assembler 223
Starting the Mini-Assembler 223
Using the Mini-Assembler 224
Mini-Assembler instruction formats 226
Summary of Monitor commands 227
Examining memory 227
Changing the contents of memory 227
Moving and comparing 227
The Register command 228
Miscellaneous Monitor commands 228
Running and listing programs 229

Chapter 11 **Hardware Implementation 231**

Environmental specifications 232
Power requirements 233
The external power supply 233
The external power connector 234
The internal converter 234
Apple IIc overall block diagram 235
The 65C02 microprocessor 237
65C02 block diagram 237
65C02 timing 239
The custom integrated circuits 241
The memory management unit (MMU) 241
The input/output unit (IOU) 243
The timing generator (TMG) 245
The general logic unit (GLU) 245
The disk controller unit (IWM) 247
Memory addressing 248
ROM addressing 249
RAM addressing 251
Dynamic RAM refreshment 251
Dynamic RAM timing 252
The keyboard 254
The speaker 256
Volume control 256
Output jack 256
The video display 257
The video counters 257
Display memory addressing 258
Display address mapping 258
Video display modes 261
Text displays 263
Low-resolution display 266
High-resolution display 267
Double high-resolution display 269
Video output signals 270
Monitor output 270
Video expansion output 271
Disk I/O 273
Serial I/O 274
ACIA control register 278
ACIA command register 280
ACIA status register 281
ACIA transmit/receive register 282

Mouse input 282
Hand controller input 287
Memory expansion card 291
Schematic diagrams 291

Appendix A **The 65C02 Microprocessor 297**

Differences between 6502 and 65C02 297
Differing cycle times 297
Differing instruction results 298
Data sheet 298

Appendix B **Memory Map 308**

Page $00 308
Page $03 312
Screen holes 312
The hardware page 316

Appendix C **Important Firmware Locations 322**

The tables 322
Port addresses 323
Other video and I/O firmware addresses 326
Applesoft BASIC interpreter addresses 326
Monitor addresses 326

Appendix D **Operating Systems and Languages 328**

Operating systems 328
ProDOS 328
DOS 328
Pascal Operating System 329
Languages 329
Applesoft BASIC 329
Integer BASIC 330
Pascal 330
Fortran 330
Logo II 330

Appendix E **Interrupts 331**

Introduction 331
What is an interrupt? 331
Interrupts on Apple II computers 332
Interrupt handling on the 65C02 333
The interrupt vector at $FFFE 333

The built-in interrupt handler 334
Saving the memory configuration 335
Managing main and auxiliary stacks 336
User's interrupt handler at $03FE 336
Handling break instructions 337
Sources of interrupts 338
Firmware handling of interrupts 339
Firmware for mouse and VBL 339
Firmware for keyboard interrupts 340
Using keyboard buffering firmware 341
Using keyboard interrupts through firmware 342
Using external interrupts through firmware 342
Firmware for serial interrupts 343
Using serial buffering transparently 343
Using serial interrupts through firmware 344
Transmitting serial data 344
A loophole in the firmware 345
Bypassing the interrupt firmware 345
Using mouse interrupts without the firmware 345
Using ACIA interrupts without the firmware 347

Appendix F **Apple II Series Differences 348**

Overview 348
Type of processor 350
Machine indentification 350
Memory structure 351
Amount and address ranges of RAM 351
Amount and address ranges of ROM 351
Peripheral-card memory spaces 352
Hardware addresses 353
$C000–$C00F 353
$C010–$C01F 353
$C020–$C02F 354
$C030–$C03F 354
$C040–$C04F 354
$C050–$C05F 354
$C060–$C06F 355
$C070–$C07F 355
$C080–$C08F 356
$C090–$C0FF 356
Monitors 356

I/O in general 357
DMA transfers 357
Slots versus ports 357
Interrupts 357
The keyboard 357
Keys, switches, and lights 358
Character sets 358
The speaker 359
The video display 359
Character sets 359
MouseText 360
Vertical blanking 360
Display modes 360
Disk I/O 361
Serial I/O 361
Serial ports versus serial cards 361
Serial I/O buffers 362
Mouse and hand controllers 363
Mouse input 363
Hand controller input and output 363
Cassette I/O 364
Hardware 365
Power 365
Custom chips 365

Appendix G **USA and International Models 366**

Keyboard layouts and codes 366
USA standard (Sholes) keyboard 367
USA simplified (Dvorak) keyboard 370
ISO layout of USA keyboard 371
English keyboard 372
French keyboad 373
Canadian keyboard 375
German keyboard 376
Italian keyboard 378
Western Spanish keyboard 380
ASCII character sets 381
Certification 383
Product safety 383
Important safety instructions 383
Power supply specifications 383

Appendix H **Conversion Tables 384**

Bits and bytes 384
Hexadecimal and decimal 387
Hexadecimal and negative decimal 388
Peripheral identification numbers 389
Eight-bit code conversions 391

Appendix I **Firmware Listings 396**

Glossary 509
Bibliography 533
Index 535
Tell Apple Card

Figures and tables

Chapter 1 **Introduction 1**

Figure 1-1 Apple IIc external features, front 2
Figure 1-2 Apple IIc external features, back 2
Figure 1-3 Front of Apple IIc with standard USA keyboard 3
Figure 1-4 USA standard (or Sholes) keyboard, keyboard switch up 6
Figure 1-5 USA simplified (or Dvorak) keyboard, keyboard switch down 7
Figure 1-6 Speaker, volume control, and audio output jack 8
Figure 1-7 Built-in disk drive 9
Figure 1-8 Back panel connectors 10
Figure 1-9 Inside the machine 11
Figure 1-10 Power supply and voltage converter 12
Figure 1-11 Original and UniDisk 3.5 IIc main logic board 13
Figure 1-12 Memory expansion IIc main logic board 14
Table 1-1 Keyboard specifications

Chapter 2 **Memory Organization and Control 17**

Figure 2-1 Internal model of the 65C02 microprocessor 19
Figure 2-2 Apple IIc memory map 21
Figure 2-3 Bank-switched memory map 25
Figure 2-4 Read ROM 29
Figure 2-5 Read ROM, write RAM, and use first $D0 bank 30
Figure 2-6 Read ROM, write RAM, and use second $D0 bank 31
Figure 2-7 Read RAM and use first $D0 bank 32
Figure 2-8 Read RAM and use second $D0 bank 33
Figure 2-9 Read and write RAM and use first $D0 bank 34
Figure 2-10 Read and write RAM and use second $D0 bank 35
Figure 2-11 48K memory map 37
Figure 2-12 48K RAM selection, split pairs 40
Figure 2-13 48K RAM selection, one side only 41
Figure 2-14 Page2 selections, 80Store on and HiRes off 47
Figure 2-15 Page2 selections, 80Store on and HiRes on 48
Figure 2-16 Reset routine flowchart 49
Table 2-1 Bank selector switches 28

Table 2-2 48K memory switches 39
Table 2-3 48K RAM transfer routines 42
Table 2-4 Parameters for MoveAux routine 43
Table 2-5 Parameters for XFer routine 43
Table 2-6 Display memory switches 45
Table 2-7 Page $03 vectors 50

Chapter 3 **Introduction to Apple IIc I/O 55**

Table 3-1 Prompt characters 59
Table 3-2 Escape codes with GetLn 61
Table 3-3 Control characters with COut1 65
Table 3-4 Control characters with C3COut1 66
Table 3-5 Text window memory locations 69
Table 3-6 Port characteristics 71
Table 3-7 Firmware protocol locations 72
Table 3-8 Port I/O locations 73
Table 3-9 Port screen hole memory locations 74

Chapter 4 **Keyboard and Speaker 77**

Table 4-1 Keyboard input characteristics 79
Table 4-2 Keys and ASCII codes 80
Table 4-3 Speaker output characteristics 83

Chapter 5 **Video Display Output 85**

Figure 5-1 MouseText characters 91
Figure 5-2 40-column and 80-column text with alternate character set 92
Figure 5-3 Text mode characteristics and switching 93
Figure 5-4 High-resolution display bits 96
Figure 5-5 Map of 40-column text display 107
Figure 5-6 Map of 80-column text display 108
Figure 5-7 Map of low-resolution graphics display 109
Figure 5-8 Map of high-resolution graphics display 110
Figure 5-9 Map of double high-resolution graphics display 111
Table 5-1 Video output port characteristics 86
Table 5-2 Video display specifications 87
Table 5-3 Display character sets 89
Table 5-4 Low-resolution graphics colors 94
Table 5-5 High-resolution graphics colors 97
Table 5-6 Double high-resolution graphics colors 99
Table 5-7 Video display page locations 101
Table 5-8 Display soft switches 102

Table 5-9 Display modes supported by firmware, including Applesoft 104
Table 5-10 Other display modes 104
Table 5-11 Monitor firmware routines 112
Table 5-12 Port 3 firmware protocol table 116
Table 5-13 Pascal video control functions 117

Chapter 6 **Block Device I/O 119**

Figure 6-1 Summary of Smartport calls 149
Table 6-1 Disk I/O port characteristics 120

Chapter 7 **Serial I/O Port 1 153**

Figure 7-1 Diagram of port 1 characteristics storage 162
Figure 7-2 Data format 163
Table 7-1 Serial port 1 characteristics 154
Table 7-2 Printer port commands 155
Table 7-3 Port 1 hardware page locations 159
Table 7-4 Port 1 I/O firmware protocol 160
Table 7-5 Port 1 screen hole locations 160

Chapter 8 **Serial I/O Port 2 167**

Figure 8-1 Diagram of port 2 characteristics storage 177
Figure 8-2 Devices in a typical communication setup 178
Figure 8-3 Effect of IN#2 180
Figure 8-4 Effect of IN#2 and T command, half duplex 181
Figure 8-5 Effect of IN#2 and T command, full-duplex terminal 182
Figure 8-6 Effect of IN#2, PR#2, and T command, full-duplex host 183
Table 8-1 Serial port 2 characteristics 168
Table 8-2 Modem port commands 170
Table 8-3 Port 2 hardware page locations 174
Table 8-4 Port 2 I/O firmware protocol 174
Table 8-5 Port 2 screen hole locations 175

Chapter 9 **Mouse and Game Input 185**

Table 9-1 Mouse input port characteristics 186
Table 9-2 Mouse soft switches 189
Table 9-3 Mouse firmware routines 193
Table 9-4 Mouse port I/O firmware protocol 195
Table 9-5 Mouse port screen hole locations 197
Table 9-6 Game input characteristics 199

Chapter 10 **Using the Monitor 203**

Table 10-1 Mini-Assembler address formats 226

Chapter 11 **Hardware Implementation 231**

Figure 11-1 External power connector 234
Figure 11-2 Apple IIc block diagram 236
Figure 11-3 65C02 block diagram 238
Figure 11-4 65C02 timing signals 240
Figure 11-5 MMU pinouts 242
Figure 11-6 IOU pinouts 243
Figure 11-7 TMG pinouts 245
Figure 11-8 GLU pinouts 246
Figure 11-9 IWM pinouts 247
Figure 11-10 Memory bus organization 249
Figure 11-11 23128 ROM pinouts 249
Figure 11-12 2316 ROM pinouts 250
Figure 11-13 2364 pinouts 250
Figure 11-14 64K RAM pinouts 251
Figure 11-15 RAM timing signals 253
Figure 11-16 Keyboard circuit diagram 254
Figure 11-17 Keyboard signals 255
Figure 11-18 Speaker circuit diagram 256
Figure 11-19 Display address transformation 260
Figure 11-20 40-column text display memory 261
Figure 11-21 Video display circuits 262
Figure 11-22 7-MHz video timing signals: 40-column, low-resolution, and high-resolution display 264
Figure 11-23 14-MHz video timing signals: 80-column and double high-resolution display 265
Figure 11-24 Video output back panel connectors 270
Figure 11-25 Video expansion connector pinouts 272
Figure 11-26 Disk drive connector 274
Figure 11-27 Serial port circuits 275
Figure 11-28 6551 ACIA block diagram 276
Figure 11-29 6551 pinouts 277
Figure 11-30 Serial port connectors 278
Figure 11-31 ACIA control register 279
Figure 11-32 ACIA command register 280
Figure 11-33 ACIA status register 281
Figure 11-34 Sample mouse waveform 283
Figure 11-35 Mouse movement and direction waveforms 283
Figure 11-36 Mouse connector 284
Figure 11-37 Mouse circuits 285
Figure 11-38 Mouse button signals 286

Figure 11-39 Hand controller connector 287
Figure 11-40 How to connect switch inputs 288
Figure 11-41 Hand controller circuits 288
Figure 11-42 Hand controller signals 289
Figure 11-43 Memory expansion card connector pinout diagram 291
Figure 11-44 Apple IIc schematic diagram 292
Table 11-1 Environmental specifications 232
Table 11-2 Power supply specifications 233
Table 11-3 External power connector signals 234
Table 11-4 Internal converter specifications 234
Table 11-5 65C02 microprocessor specifications 239
Table 11-6 65C02 timing signal descriptions 240
Table 11-7 MMU signal descriptions 242
Table 11-8 IOU signal descriptions 243
Table 11-9 TMG signal descriptions 245
Table 11-10 GLU signal descriptions 246
Table 11-11 IWM signal descriptions 247
Table 11-12 RAM address multiplexing 252
Table 11-13 RAM timing signals 253
Table 11-14 Display memory addressing 260
Table 11-15 Memory address bits for display modes 260
Table 11-16 Character-generator control signals 266
Table 11-17 Video expansion connector signals 272
Table 11-18 Disk drive connector signals 274
Table 11-19 6551 signal descriptions 277
Table 11-20 Serial port connector signals 278
Table 11-21 Mouse connector signals 284
Table 11-22 Hand controller connector signals 287

Appendix A **The 65C02 Microprocessor 297**

Table A-1 Cycle time differences 298

Appendix B **Memory Map 308**

Table B-1 Page $00 use 309
Table B-2 Page $03 use 312
Table B-3 Main memory screen hole allocations 313
Table B-4 Auxiliary memory screen hole allocations 315
Table B-5 Addresses $C000–$C03F 316
Table B-6 Addresses $C040–$C05F 318
Table B-7 Addresses $C060–$C07F 319
Table B-8 Addresses $C080–$C0AF 320
Table B-9 Addresses $C0B0–$C0FF 321

Appendix C **Important Firmware Locations 322**

Table C-1 Serial port 1 addresses 323
Table C-2 Serial port 2 addresses 324
Table C-3 Video firmware addresses 324
Table C-4 Mouse port addresses 325
Table C-5 Apple IIc enhanced video and miscellaneous firmware 326
Table C-6 Apple IIc monitor entry points and vectors 326

Appendix E **Interrupts 331**

Table E-1 Interrupt-handling sequence 335
Table E-2 Activating mouse interrupts 346
Table E-3 Reading mouse interrupts 346

Appendix F **Apple II Series Differences 348**

Figure F-1 Apple II, II Plus, and IIe hand controller signals 364
Table F-1 Apple II series indentification bytes 350

Appendix G **USA and International Models 366**

Figure G-1 USA standard (or Sholes) keyboard, keyboard switch up 368
Figure G-2 USA simplified (or Dvorak) keyboard, keyboard switch down 370
Figure G-3 ISO version of USA standard keyboard, keyboard switch up 371
Figure G-4 English keyboard, keyboard switch up 372
Figure G-5 French keyboard, keyboard switch down 373
Figure G-6 Canadian keyboard, keyboard switch down 375
Figure G-7 German keyboard, keyboard switch down 376
Figure G-8 Italian keyboard, keyboard switch down 378
Figure G-9 Western Spanish keyboard, keyboard switch down 380
Table G-1 Keys and ASCII codes 368
Table G-2 English keyboard code differences from Table G-1 372
Table G-3 French keyboard code differences from Table G-1 374
Table G-4 Canadian keyboard code differences from Table G-1 375
Table G-5 German keyboard code differences from Table G-1 377

Table G-6 Italian keyboard code differences from Table G-1 379
Table G-7 Western Spanish keyboard code differences from Table G-1 381
Table G-8 ASCII code equivalents 381
Table G-9 50-Hz power supply specifications 383

Appendix H **Conversion Tables 384**

Figure H-1 Bits, nibbles, and bytes 386
Table H-1 What a bit can represent 385
Table H-2 Values represented by a nibble 386
Table H-3 Hexadecimal/decimal conversion 387
Table H-4 Hexadecimal to negative decimal conversion 388
Table H-5 PIN numbers 390
Table H-6 Control characters, high bit off 392
Table H-7 Special characters, high bit off 393
Table H-8 Uppercase characters, high bit off 394
Table H-9 Lowercase characters, high bit off 395

Appendix I **Firmware Listings 396**

Table I-1 Main side ROM map 397
Table I-2 Auxiliary side ROM map 398

Preface

About This Manual

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

The information in this manual is aimed at assembly-language programmers and hardware designers, but others interested in the internal operation of the Apple IIc can also benefit from reading it.

This manual tells you how the Apple IIc works, but not how to use it. If you need to know how to set up and use your Apple IIc, read the *Apple IIc Owner's Manual.*

This manual describes three versions of the Apple IIc:

- the original Apple IIc
- the Apple IIc that supports the UniDisk™ 3.5 drive
- the Apple IIc that supports the Memory Expansion Card

More information on the various versions of the Apple IIc is provided under "The Apple IIc Family," later in this Preface.

Contents of this manual

The Apple IIc is presented in this manual from the outside in.

Chapter 1 introduces the Apple IIc, including external controls, connectors, and the main internal components.

Chapter 2 introduces the 65C02 microprocessor and its directly addressable memory space.

Chapter 3 introduces the I/O characteristics of the Apple IIc. Chapters 4 and 9 cover specific areas of the I/O interface.

Chapter 4 describes the keyboard and speaker.

Chapter 5 describes the video display.

Chapter 6 describes block device I/O, including the Smartport firmware interface.

Chapter 7 describes serial port 1.

Chapter 8 describes serial port 2.

Chapter 9 describes the mouse/game paddle port.

Chapter 10 describes the Apple IIc's built-in Monitor firmware. The Monitor helps you write, disassemble, and debug machine-language programs, as well as providing you with a means to look at and manipulate the contents of main memory.

Chapter 11 describes the Apple IIc hardware in detail.

Appendix A describes the 65C02 microprocessor in detail, including the differences between it and the 6502 microprocessor used on early-model Apple II's. Most of this appendix is a reprint of the manufacturer's data sheet for the 65C02.

Appendix B contains a memory map of the Apple IIc main memory. Detailed maps are provided for memory pages $00 and $03, the screen holes, and the hardware page.

Appendix C lists the Apple IIc firmware entry points, including those for the I/O firmware and the Monitor firmware.

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

Appendix E describes the operation of the Apple IIc interrupt handler firmware and how to use it in your programs.

Appendix F outlines the differences and similarities between the diverse members of the Apple II family of computers.

Appendix G describes the various international versions of the Apple IIc keyboard and character set. Power and safety information for international versions of the Apple IIc is also included in this appendix.

Appendix H contains tables to aid you in code and number base conversions.

Appendix I contains the firmware listing for the new version of the Apple IIc and information on obtaining listings for the original and UniDisk 3.5 ROMs.

The Glossary defines many of the technical terms used in this manual.

The Bibliography lists articles and books with additional information about the Apple IIc.

Finally, after the index at the back of this manual, you'll find the Tell Apple Card; please take a minute to fill this card out and mail it back to us. Your experience with this and other Apple manuals can help us plan new reference materials.

The Apple IIc family

Changes have been made to the Apple IIc since the original version was introduced. The first change was made in order to support the UniDisk 3.5 external drive, and included a set of ROM-based machine-language routines called the **Protocol Converter.** The latest version incorporates all the UniDisk 3.5 upgrade features, a new version of the Protocol Converter called the **Smartport,** and support for an optional memory expansion card. All of these versions are described in this manual. Where there are differences between the various versions of the Apple IIc, they will be called out in the manual. For the sake of convenience, the various versions of the Apple IIc are identified by the features they support, such as *memory expansion* for the newest IIc and *UniDisk 3.5* for the version that introduced the UniDisk 3.5 drive support. Unless specified, all versions of the Apple IIc operate identically.

Important Smartport is merely a new name for the Protocol Converter; all the specifications for the Smartport apply to the Protocol Converter, and vice versa.

Identifying your Apple IIc

There are basically three versions of the Apple IIc:

- the *original* Apple IIc
- the *UniDisk 3.5* Apple IIc
- the *memory expansion* Apple IIc

You can tell which Apple IIc you have by checking the value of the ID byte at ROM location 64447 ($FBBF in hexadecimal). The value of this byte is 255 ($FF) in the original Apple IIc, 0 ($00) in the UniDisk 3.5 version, and 3 ($03) in the memory expansion version.

❖ *Checking the ID byte:* You can check the value of the ID byte from Applesoft by typing `PRINT PEEK (64447)`.

The original Apple IIc

The original Apple IIc is the oldest member of the IIc family. It has the following features:

- the 65C02 microprocessor
- 128K of RAM

The UniDisk 3.5 Apple IIc

The Apple IIc that introduced support for the UniDisk 3.5 drive is identified in this manual as the UniDisk 3.5 version. It includes the following changes from the original Apple IIc:

- the Protocol Converter, to support the UniDisk 3.5 external disk drive
- a 256K ROM IC to replace the 128K ROM
- some new serial port commands
- the Mini-Assembler
- two new Monitor commands (STEP and TRACE)
- built-in diagnostics

The UniDisk 3.5 Apple IIc also includes improved interrupt handler features and new external drive startup procedures.

The memory expansion Apple IIc

The Apple IIc that supports an optional memory expansion card supports all the features of the UniDisk 3.5 version. It includes the following changes from the UniDisk 3.5 IIc:

- an internal connector to support an optional memory expansion card
- 4 64Kx4 RAM ICs to replace the 16 64Kx1 ICs

The Apple IIc that supports the memory expansion option also reorganizes the I/O port ("slot") entry points in the firmware. The mouse, located at port 4 in the original and UniDisk 3.5 versions, is now at port 7. The memory expansion card uses port 4 in the new Apple IIc. What this means is that *all the mouse I/O entry point addresses have been changed from $C4XX to $C7XX.*

To avoid confusion and maintain compatibility with previous versions, the text and tables in this book still show the values used for the original and UniDisk 3.5 versions of the Apple IIc. However, a statement reminding you of the change appears near affected tables.

Remember that the Smartport and the Protocol Converter are the same thing.

Conventions used in this manual

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

Warning Important warnings look like this. These flag potential danger to the Apple IIc, its software, or you.

Important Text set off like this is less urgent or threatening than text in a Warning box, but still of a critical nature.

Original IIc Text set off like this applies only to the original version of the Apple IIc.

UniDisk 3.5 Text set off like this applies only to the UniDisk 3.5 version of the Apple IIc.

Memory expansion Text set off like this applies only to the memory expansion version of the Apple IIc.

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

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

`Computer voice` is used to indicate text that should be identical to your screen display or printout.

Chapter 1

Introduction

This chapter introduces you to the working parts of the Apple IIc by briefly describing the major components of the computer—both internal and external hardware and firmware—and telling you where in the manual to find out more about them.

The outside of the machine

This section briefly describes the Apple IIc's keyboard, controls, indicators, and expansion connectors.

The Apple IIc comes equipped with a keyboard, speaker (with audio output jack and volume control), built-in disk drive, external power supply, and internal voltage converter. It also has built-in interfaces with external connectors for a serial printer, video monitor, special video display adapters, modem, mouse, and game controllers. These external connectors allow you to plug in accessory equipment without having to go inside the machine to use expansion slots like those in the Apple IIe.

Figure 1-1 shows the front and right side of an Apple IIc, and Figure 1-2 shows the back and left side.

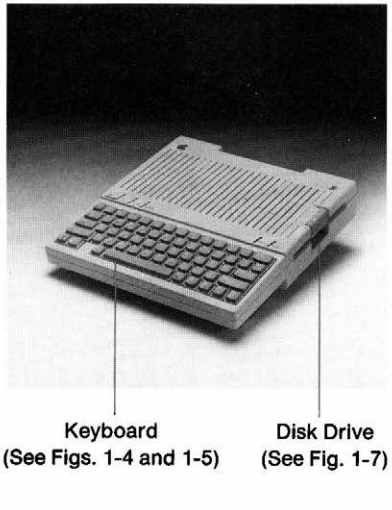


Figure 1-1
Apple IIc external features, front

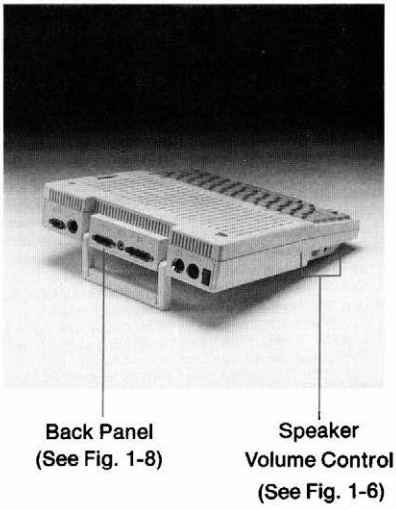


Figure 1-2
Apple IIc external features, back

The keyboard

ASCII stands for *American Standard Code for Information Interchange*. Table 4-2 lists the ASCII character encoding for the standard and simplified USA keyboards. Appendix G lists the encoding for international keyboards.

The Apple IIc's primary input device is the keyboard, shown in Figure 1-3. The keyboard has a 63-key typewriter layout with both uppercase and lowercase characters and can generate all 128 standard **ASCII** characters. A reset key, 80/40-column display selector switch, keyboard layout selector switch, disk-use light, and power light are also located on the front of the computer.

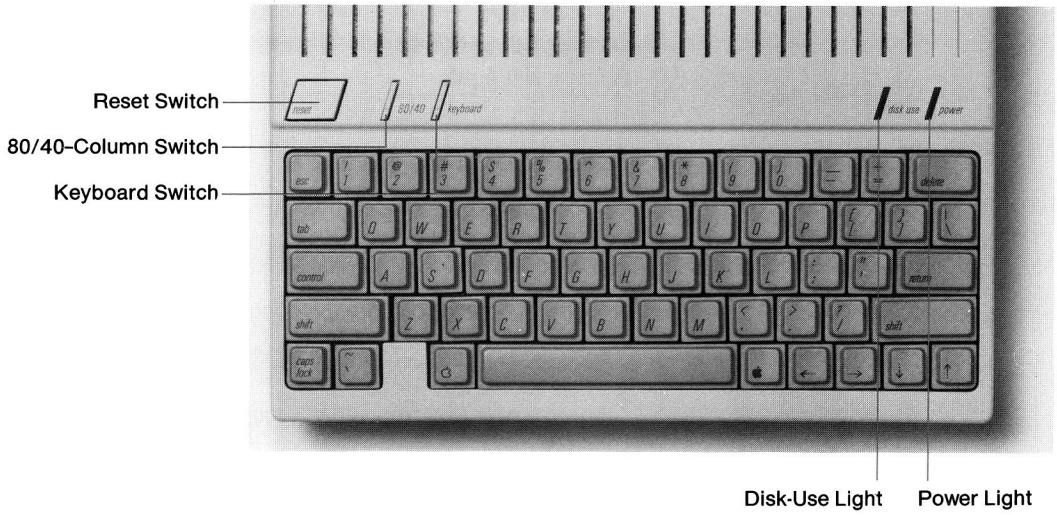


Figure 1-3
Front of Apple IIc with standard USA keyboard

Table 1-1 lists the characteristics of all Apple IIc keyboards and front panels.

Features

The Apple IIc keyboard has automatic repeat on all character keys. This means that if you hold the key down longer than about a second, the character it generates repeats until you let up the key. It also has two-key rollover, which means if you press a key before releasing the one you pressed before it, the second character enters the computer the same as though you had released the previous key first. (This is important for fast touch-typists.)

Table 1-1
Keyboard specifications

Number of keys	63
Character encoding	ASCII
Number of codes	128
Features	Automatic repeat, two-key rollover
Special function keys	Reset, Open Apple, Solid Apple,
Cursor movement keys	Left Arrow, Right Arrow, Down Arrow, Up Arrow, Return, Delete, Tab
Modifier keys	Control, Shift, Caps Lock, Escape
Front-panel switches	80/40 switch, keyboard switch
Front-panel lights	Power light, disk-use light

Special function keys

The Open Apple and Solid Apple keys are connected to 1-bit addresses in memory, described in Chapter 9.

The Apple IIc keyboard has three special function keys: Reset, and two keys marked with apples—one outlined (Open Apple) and one filled in (Solid Apple).

Reset has a direct line to the 65C02 microprocessor's RESET signal line (see Chapter 11): holding down Control while pressing Reset causes the Apple IIc to restart processing with an internal firmware program that puts the machine in a known state (see Chapter 2).

Chapter 2 describes the results of the various reset procedures.

You can restart the Apple IIc without turning the power off and back on again, by holding down both Control and Open Apple while pressing Reset. Restarting this way is less stressful to the Apple IIc's components than normal powerup.

Cursor movement keys

The Apple IIc keyboard has four cursor movement keys with arrows marked on them: left, right, down, and up. Three other keys can also cause cursor movements: Return, Delete, and Tab. All seven of these keys generate ASCII control characters (see Table 4-2). It is up to the operating system or application program to interpret and act on the control codes that these keys generate.

Modifier keys

Three special keys—Control, Shift, and Caps Lock—generate no codes when pressed by themselves, but change the codes generated by other keys they are pressed in combination with. A fourth key, Escape, generates a nonprinting control code that causes the **Monitor** to interpret certain subsequent keystrokes in a modified way.

The **Monitor** is a built-in program that performs some of the basic activities of the computer, such as retrieving and storing key codes as they come in, and clearing or updating the display screen.

- Control, when pressed in combination with letter keys or certain other keys, produces ASCII control characters. Most of the control characters are invisible most of the time.
- Shift works the same on the Apple IIc as on an ordinary typewriter: it selects uppercase letters and the upper characters on the keys.
- Caps Lock, in its down position, changes the letter keys to uppercase, but does not affect other keys.
- Escape is not a modifier key in the same sense as Control and Shift: you do not hold it down while pressing other keys. Rather, you press Escape and it generates the ASCII escape (ESC) control character (key code $1B—see Table 4-2). When the Escape key is pressed, many programs—including the built-in Monitor program—then interpret other specific keys as designating an escape sequence.

The 80/40 switch

The 80/40 switch lets you specify whether a program should display information in 40 or 80 columns per line. The switch indicates 40-column display when in its down position, and 80-column display when in its up position.

Important Not all programs check this switch. Even programs that do check the switch may do so only when the program first starts up. If that is the case, changing the switch position while the program is running will have no effect on the program's display. (See Table 4-1.)

The keyboard switch

You use the keyboard switch to select for use one of the two keyboard layouts and screen character sets built into your Apple IIc. On USA versions of the Apple IIc, you select the standard Sholes keyboard layout (Figure 1-4) with the switch in the up position, and the Dvorak simplified layout (Figure 1-5) with the switch in the down position.

If you normally use the Dvorak keyboard layout, you can *gently* pry up the keys from the keyboard and rearrange and replace them in their Dvorak positions.

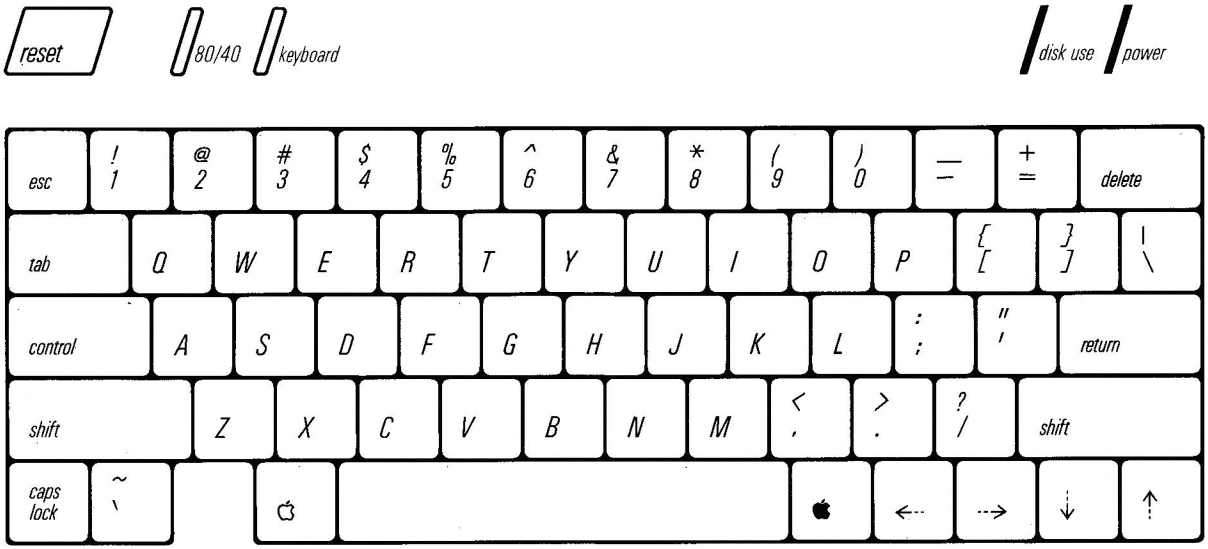


Figure 1-4
USA standard (or Sholes) keyboard, keyboard switch up

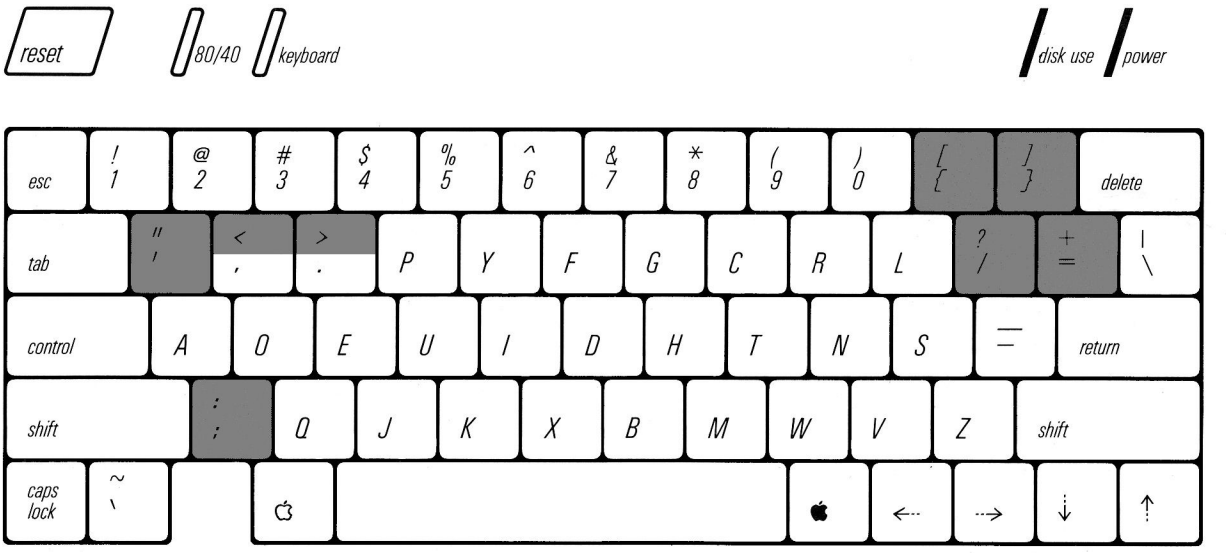


Figure 1-5
USA simplified (or Dvorak) keyboard, keyboard switch down (shaded characters may be in different positions on some models)

Appendix G illustrates the keyboard layouts for both keyboard switch positions on several international versions of the Apple IIc.

On international models, the keycaps indicate the character positions for the local keyboard layout, which is selected when the keyboard switch is down. When up, the keyboard switch selects the USA standard characters and key layout.

Disk-use and power lights

The red disk-use light glows whenever the built-in disk drive's motor is switched on.

The green power light glows when the Apple IIc is turned on and normal power is present at the Apple IIc's internal power supply.

Warning If the power light flashes on and off, turn off the computer **immediately.** Find out what caused the condition (such as a brownout or short circuit) and fix the problem before turning the computer on again. Above all, do not use the disk drive when the power light is flashing; this may damage the computer.

The speaker

The way programs control the speaker is described under "Speaker Output" in Chapter 4.

The Apple IIc has a speaker in the bottom of the case, as shown in Figure 1-6. The speaker lets Apple IIc programs produce a variety of sounds. There is also a volume control on the left side of the Apple IIc case, and a jack for connecting headphones or an external speaker. The jack accepts either one-channel (monaural) or two-channel (stereo) plugs, although speaker output is monaural only. Inserting a plug disconnects the built-in speaker

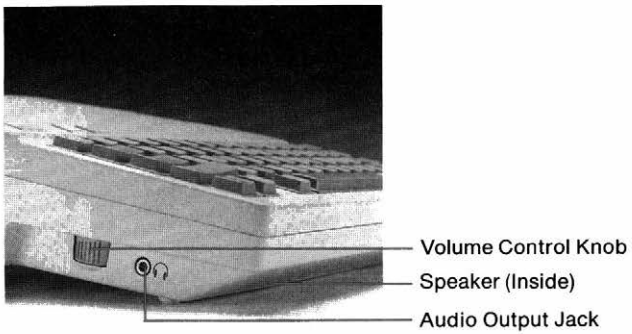


Figure 1-6
Speaker, volume control, and audio output jack

The built-in disk drive

The Apple IIc's built-in disk drive (Figure 1-7) is fully compatible with the Apple Disk IIc that reads and writes 5.25-inch single-sided 35-track disks. The drive door is on the right side of the Apple IIc case.

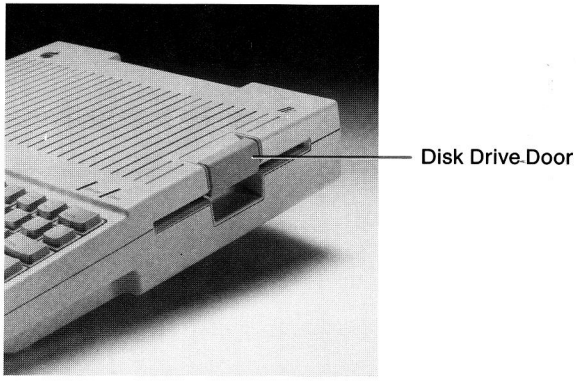


Figure 1-7
Built-in disk drive

The back panel

The back panel of the Apple IIc (Figure 1-8) has seven connectors and a main power switch. From left to right they are

- a 9-pin D-type miniature connector for connecting hand controllers, a mouse, a joystick, or some other device (see Chapters 9 and 11)
- a 5-pin DIN connector for serial input and output (port 2; normally for a modem) (see Chapters 7 and 11)
- a 15-pin D-type connector for video expansion (see Chapter 11)
- an RCA-type jack for a video monitor (see Chapter 11)
- a 19-pin D-type connector for connecting one or more external devices, such as intelligent disk drives (see Chapters 6 and 11)
- another 5-pin DIN connector for serial input and output (port 1; normally for a printer or plotter) (see Chapters 8 and 11)
- a special 7-pin DIN connector for power input (see Chapter 11)

Before attaching cables to the Apple IIc back panel connectors, be sure to move the handle until it clicks into position for propping up the computer. The handle should be down whenever the computer is running so that it can maintain proper cooling airflow.

The installation manuals for external devices contain instructions for connecting them to the Apple IIc.

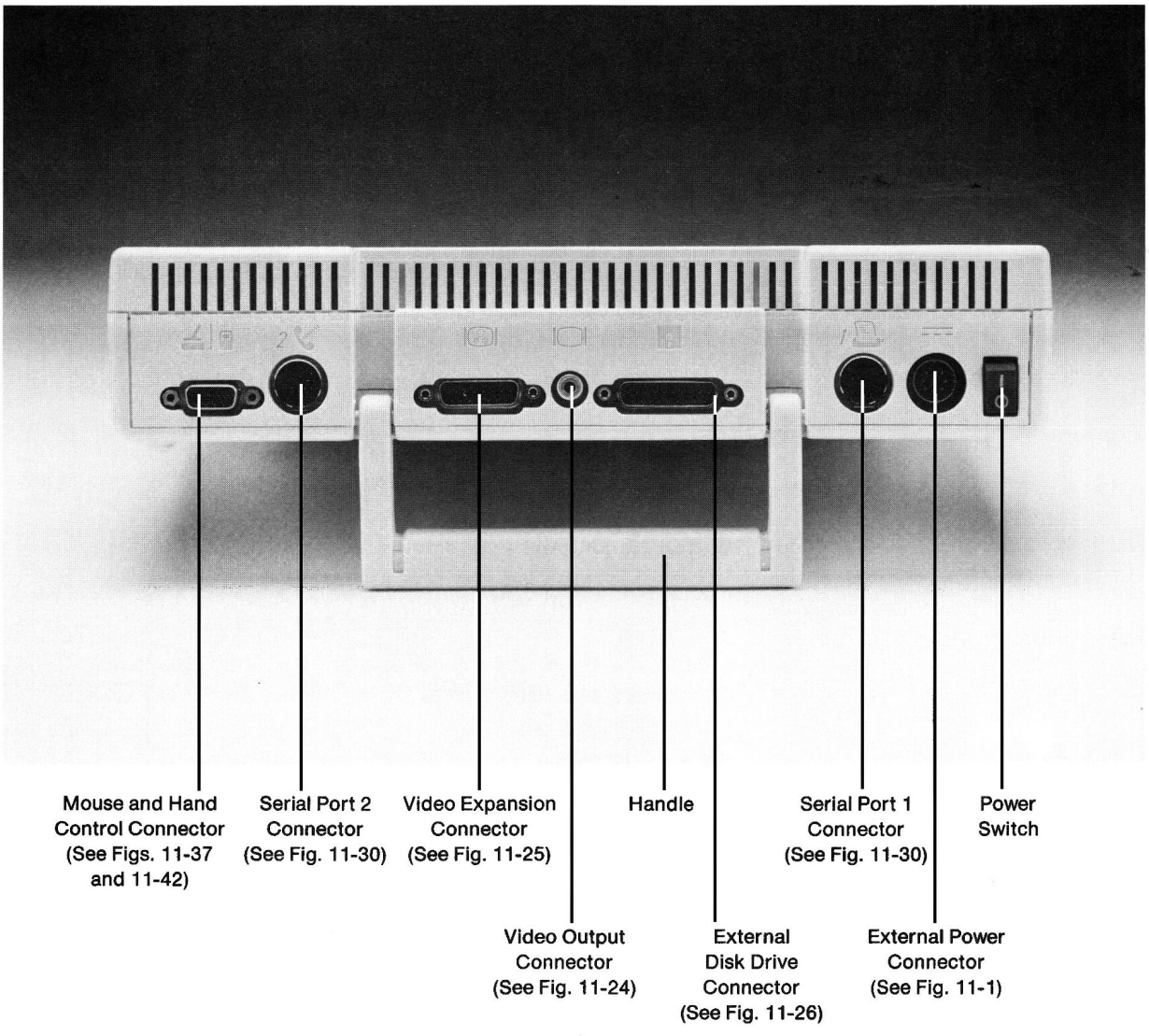


Figure 1-8
Back panel connectors

The inside of the machine

Figure 1-9 shows the main components inside the Apple IIc computer.

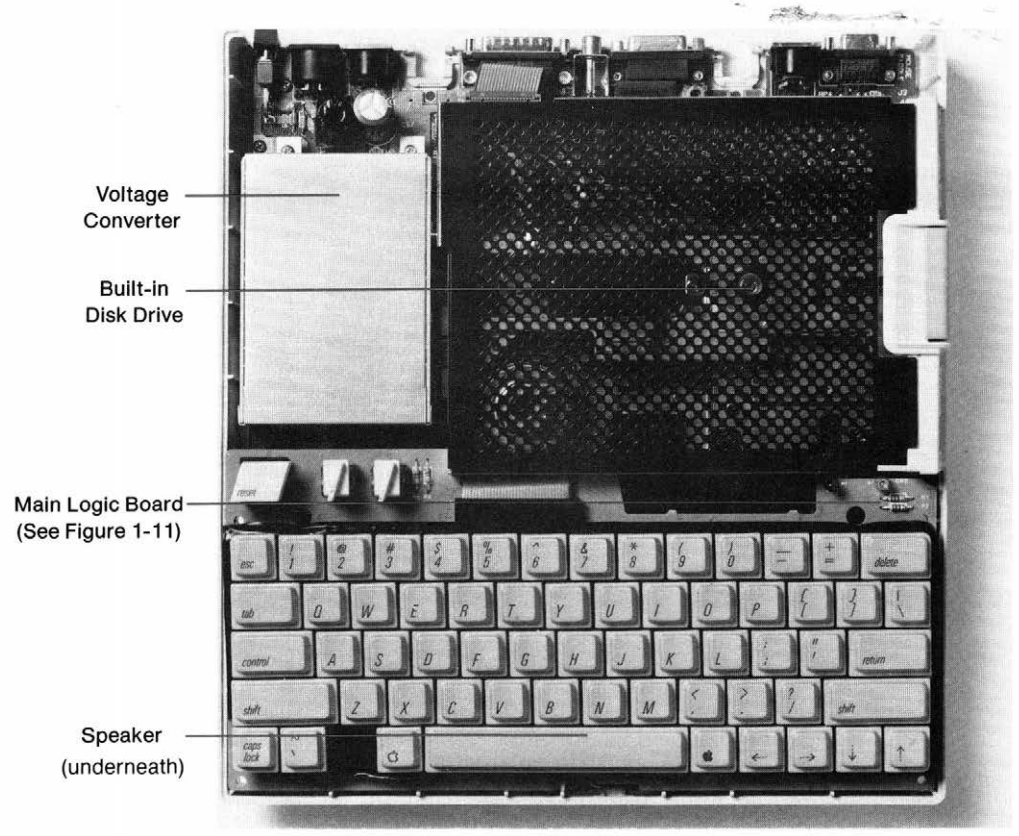


Figure 1-9
Inside the machine

The internal voltage converter

Complete specifications of the Apple IIc power supply and voltage converter appear in Chapter 11.

The built-in voltage converter operates from a 12 to 15 VDC input source, such as provided by the external power supply furnished with the Apple IIc (Figure 1-10). The voltage converter provides power for the logic board, built-in disk drive, one external disk drive, and the I/O signals available at the back panel.

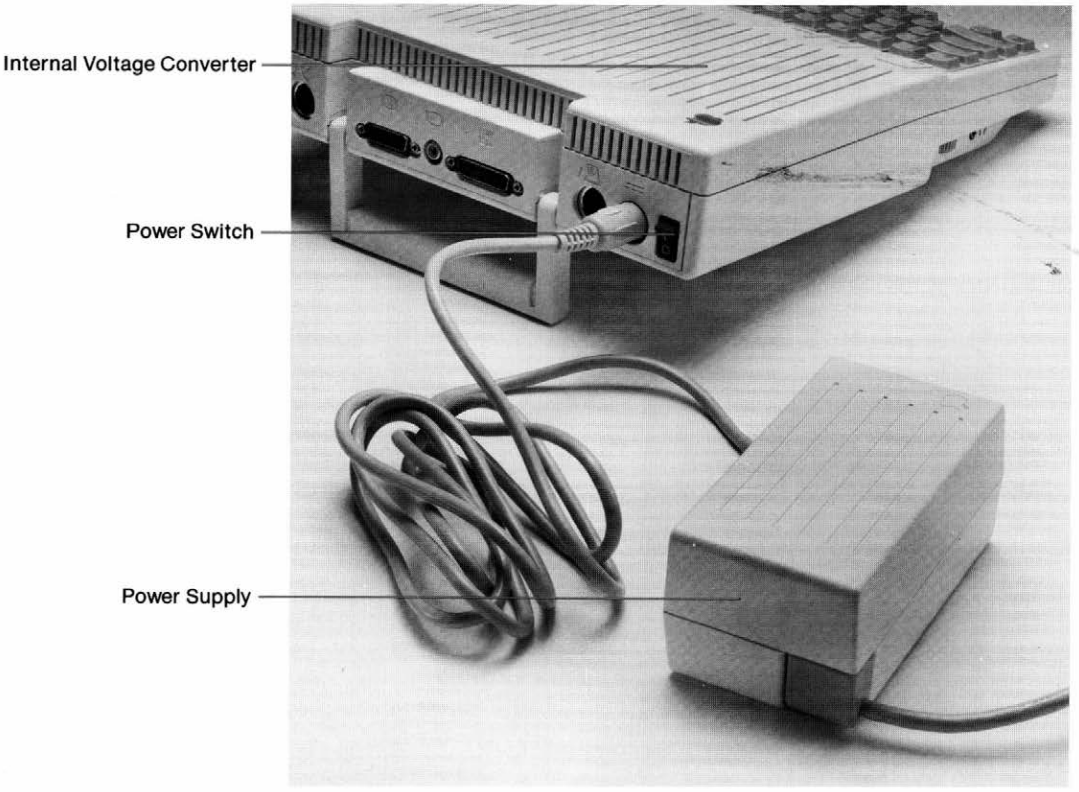


Figure 1-10
Power supply and voltage converter

The voltage converter produces three different voltages: +5V, +12V, and –12V. (Minus 5V, needed by some components in the Apple IIc, is derived from –12V on the main logic board.) It is a high-efficiency switching converter that protects itself and the rest of the Apple IIc against short circuits and other electrical mishaps.

The main logic board

The main logic board, which is mounted flat in the bottom of the Apple IIc's case, has almost all the electronic parts of the computer attached to it.

Figure 1-11 shows the main logic board and the most important integrated circuits (ICs) in the Apple IIc. They are the CPU (central processing unit), RAM (random-access memory), ROM (read-only memory) ICs for keyboard encoding, display character generation, and **firmware,** and the five custom ICs.

Firmware is program code that is stored in ROM. It can be read and executed, but not changed.

The processor is a 65C02 microprocessor. The 65C02 is a CMOS version of the 6502 used in other members of the Apple II family. It is an 8-bit microprocessor with a 16-bit address bus. In the Apple IIc, the 65C02 runs at 1 MHz and performs up to 500,000 8-bit operations per second.

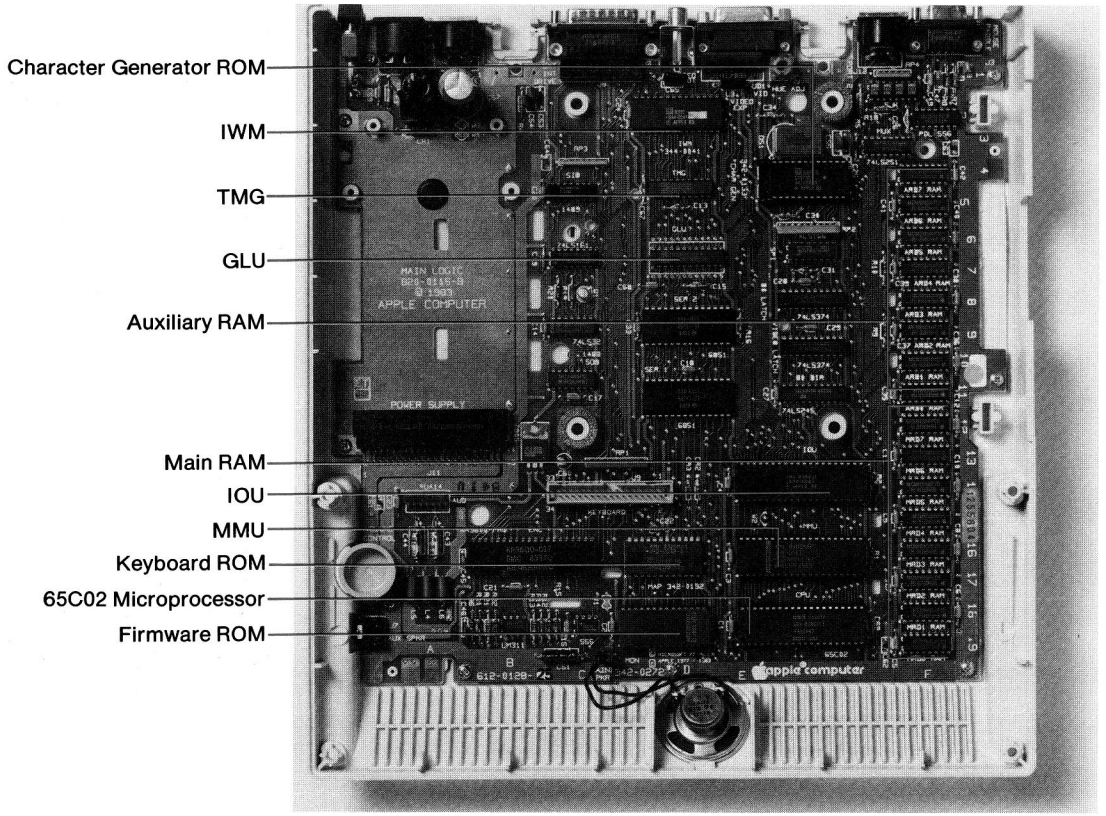


Figure 1-11
Original and UniDisk 3.5 IIc main logic board

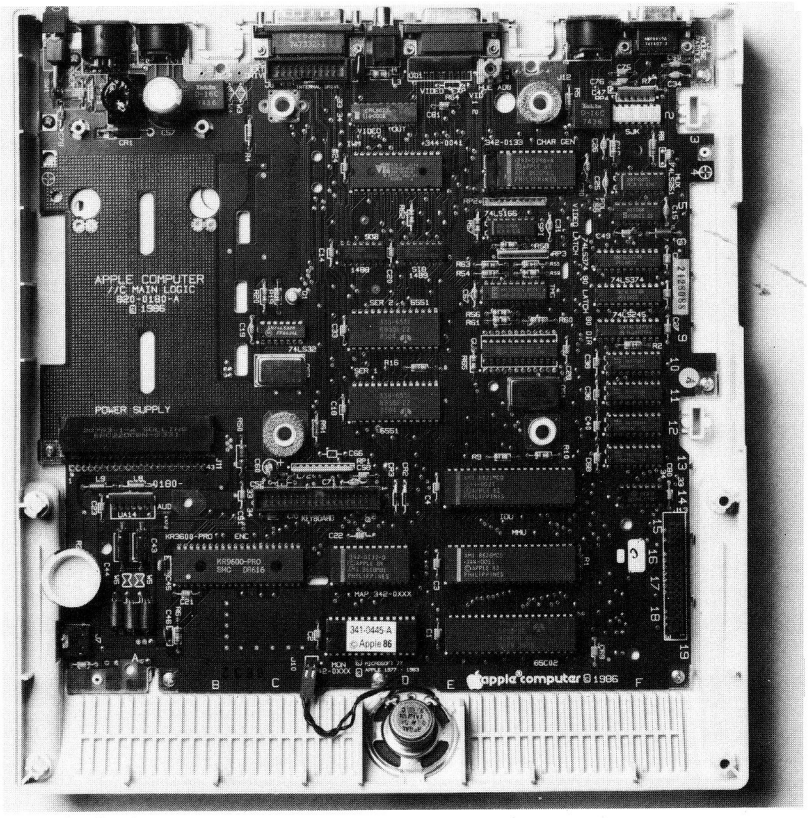


Figure 1-12
Memory expansion IIc main logic board

The keyboard is scanned by an IC that generates matrix values for a ROM. The value of the ASCII code supplied by the ROM is latched at a specified memory location and is readable by programs.

The character generator ROM converts ASCII character values to a form that the video display can use.

The Applesoft language interpreter is described in the *Applesoft Tutorial* and the *Applesoft BASIC Programmer's Reference Manual.*

The other ROM contains the Monitor, the Applesoft BASIC interpreter, enhanced video firmware, and other input/output firmware. The firmware that this ROM contains is described throughout this manual.

Five of the large ICs on the main logic board are custom-made for the Apple IIc:

For more on memory addressing, see Chapter 2.

- The *memory management unit* (MMU) contains most of the logic that controls memory addressing in the Apple IIc.

See Chapters 3 through 9.

- The *input/output unit* (IOU) contains most of the logic that controls the built-in input and output features of the Apple IIc.
- The *timing generator* (TMG) generates all the system and I/O clock and timing signals from a 14-MHz oscillator.
- The *general logic unit* (GLU) performs the remaining required logic functions.

Chapter 11 discusses the functions of these integrated circuits in some detail.

- The *disk controller unit*, also known as the Integrated Woz Machine (IWM), is a single-chip version of the Apple Disk II controller card. It controls the built-in and external disk drives connected to the Apple IIc.

The other circuit boards

The Apple IIc contains other circuit boards that serve special purposes: a motor-speed control and read/write logic board for the disk drive, and a matrix board for detecting the position of keys pressed. This manual does not discuss these circuit boards.

Warning Adjustment of disk drive speed must be done by an authorized Apple Service Center. Do not attempt to adjust the speed of your built-in disk drive. If you do, you may damage it and you will void your warranty.

Chapter 2

Memory Organization and Control

This chapter introduces the Apple IIc's processor, the 65C02, and the memory ranges and locations in the Apple IIc that have been set aside for special purposes. The last section of this chapter describes the reset routines, which restore the computer to a known state.

The 65C02 microprocessor

The 65C02 is a general-purpose 8-bit CMOS microprocessor similar in operation to the 6502 used in other members of the Apple II family of computers.

Figure 2-1 is a model of the 65C02 microprocessor's register organization. Registers are fast-acting built-in storage areas where the processor performs and keeps track of its work. The 65C02 has one 16-bit register and five 8-bit registers.

Each of the other registers holds eight bits (one byte), so the 65C02 is called an *8-bit processor.*

The 16-bit register is called the *program counter* (PC). It specifies the address in memory that contains the instruction the processor is currently carrying out. A 16-bit register can specify any one of 65,536 memory addresses, and so the 65C02 is said to have an address space of 65,536 locations.

Appendix A lists the instructions the 65C02 can carry out, their use, and their effects on the registers. For further information, consult the pertinent books listed in the Bibliography.

The five 8-bit registers in the 65C02 are the following:

- The *accumulator,* or A register. The accumulator is like a desk top where the processor performs mathematical and logical operations on information.
- The *index registers,* X and Y. The processor uses these registers to modify the address where information is to be found or placed, and to pass information from one program to another.
- A *stack pointer,* or S register. The processor uses a 256-byte region of memory—page $01—as an area to stack up bytes for future use. The stack is empty when the computer is turned on. Several 65C02 instructions either push (store) the contents of a register onto the stack, or pull (retrieve) a byte from the stack and place it in a register. The S register keeps track of the address of the byte in the stack that is currently ready for use.
- A *processor status register,* or P register. Seven of the eight bits of this register are used as flags to record the outcome of processor activities, and can be checked by later instructions to determine what has happened and what the processor should do next.

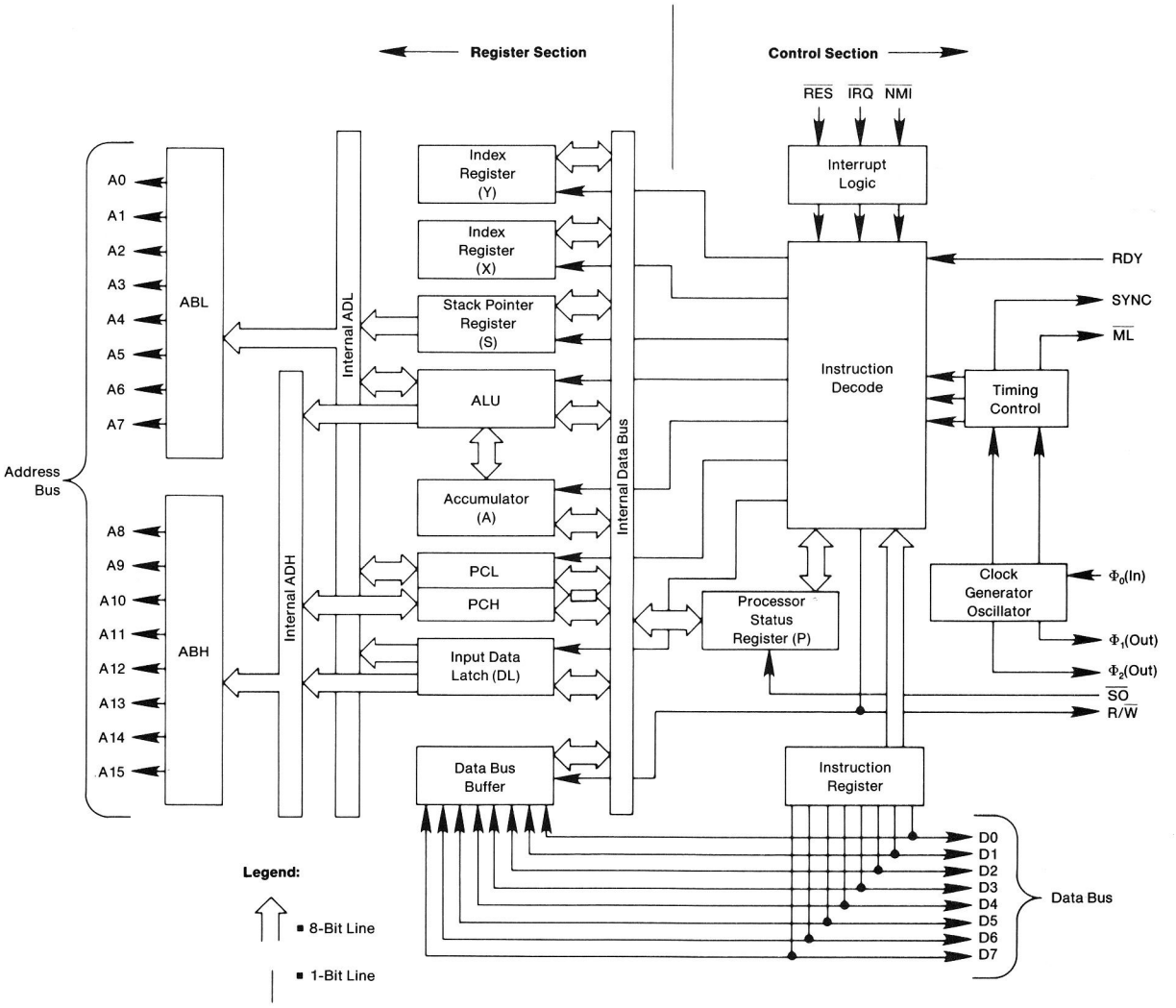


Figure 2-1
Internal model of the 65C02 microprocessor (copyright © 1982 by NCR Corporation; used by permission)

Overview of the address space

Soft switches are described more fully under "Bank-Switched Memory" and "48K Memory."

The Apple IIc's 65C02 microprocessor can address 65,536 (64K) memory locations. All the Apple IIc's RAM, ROM, and input and output (I/O) devices are accessed using addresses in this 64K address range. Some functions have the same addresses—but not at the same time. The Apple IIc controls its shared addresses by using soft switches. A **soft switch** is a memory location that controls some aspect of the computer's operation when it is accessed.

There are two other ROMs in the Apple IIc: one to generate characters corresponding to keystrokes and another to generate characters for display. (See "The Keyboard" and "The Video Display" in Chapter 9.) However, these ROMs are not addressable by the microprocessor.

All input and output in the Apple IIc is memory mapped—that is, specific memory addresses (all in the $C0 page) are allocated to each I/O device. In this chapter, the I/O memory spaces are described simply as areas of memory. For details of the built-in I/O features and firmware, refer to the descriptions in Chapters 3 through 9.

A contiguous block of 256 address locations in the 65C02's address range is called a **page.** A 1-byte address counter or 8-bit register can specify 1 of 256 different locations. Thus, page $00 consists of memory locations from 0 through 255 (hexadecimal $00 through $FF); page $01 consists of locations 256 through 511 (hexadecimal $0100 through $01FF); and so on. In this manual, all page numbers are given in hexadecimal format.

❖ *Note:* The first two digits of a four-digit hexadecimal address are the page number. There are 256 pages of 256 bytes each in the address space. This kind of page is different from the display areas in the Apple IIc, which are sometimes referred to as *Page 1* and *Page 2.* In this manual, dollar signs ($) in addresses signify that the addresses are in hexadecimal notation.

Memory map and memory switching

Figure 2-2 is a map of the Apple IIc's memory address space and what the major blocks of addresses are used for. As you can see in the figure, addresses $C000 through $C0FF contain hardware only, and addresses $C100 through $CFFF contain ROM only. At all other addresses there are two to five blocks of RAM or ROM locations. At any given time, only one block of RAM or ROM occupies each set of addresses. As described later in this chapter, soft switches in the hardware page control that blocks the processor is currently using.

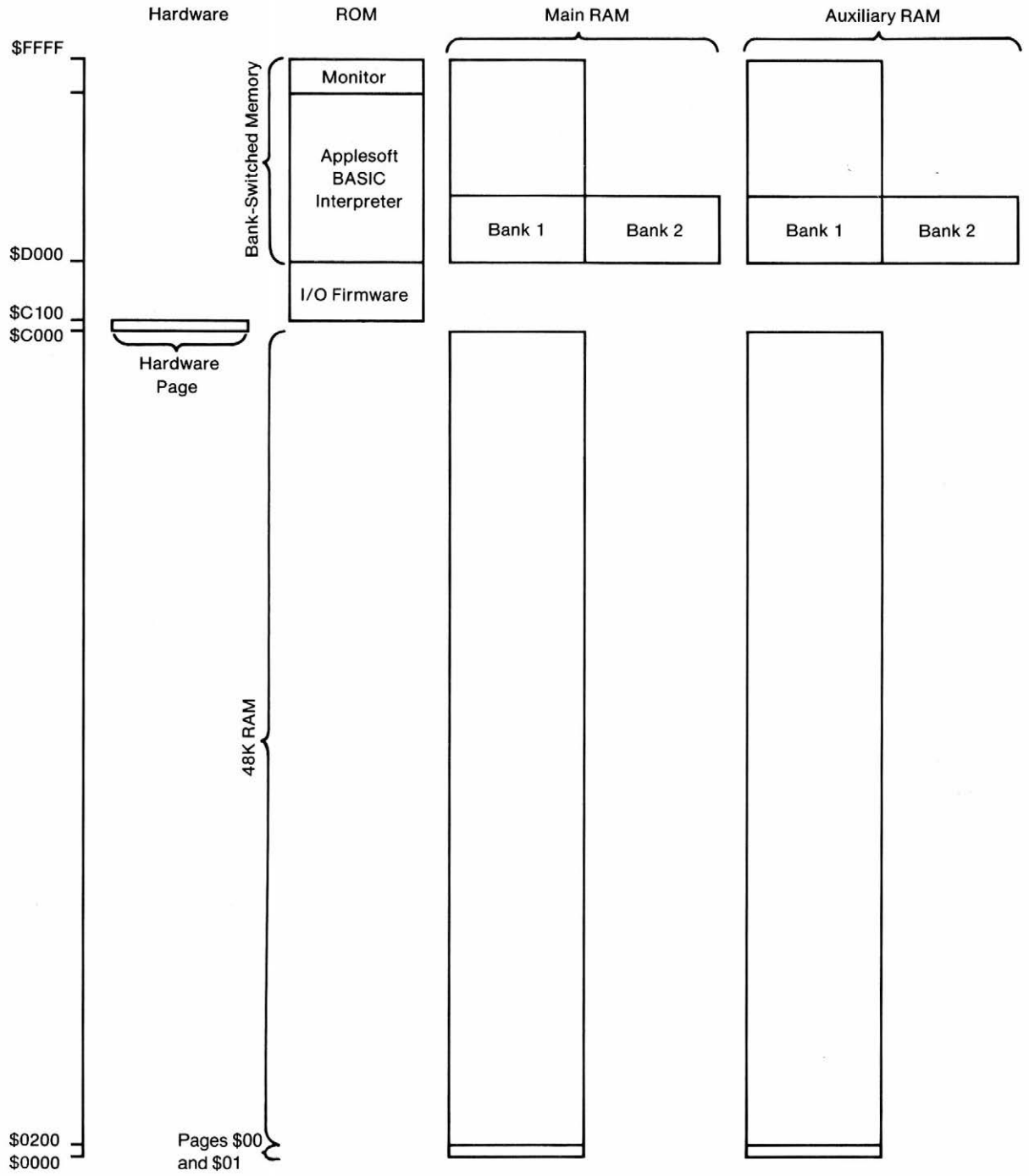


Figure 2-2
Apple IIc memory map

Main RAM addresses ($0000–$BFFF and $D000–$FFFF)

The area labeled *Main RAM* in Figure 2-2 is so called because some or all of it is present in all models of the Apple II series of computers. The Apple IIc has 64K bytes of main RAM.

Auxiliary RAM addresses ($0000–$BFFF and $D000–$FFFF)

The Apple IIc has 64K of auxiliary RAM built in. Some or all of that range of auxiliary memory is present in an Apple IIe with one of the 80-column text cards installed (see Appendix F), but there is no auxiliary RAM in the Apple II or II Plus.

A range of addresses in auxiliary RAM cannot be used simultaneously with the same range of addresses in main RAM; your programs must use the soft switches described in this chapter to select either main or auxiliary memory for any given range of addresses.

ROM addresses ($C100–$FFFF)

ROM addresses contain the built-in Apple IIc firmware. Addresses $C100 through $CFFF belong exclusively to ROM. Addresses $D000 through $FFFF are shared by ROM, main RAM, and auxiliary RAM; the selection techniques are described later in this chapter.

The Apple IIc's built-in ROM pages $C1 through CF (addresses $C100 through $CFFF) contain I/O firmware. The Apple IIc I/O firmware is roughly divided among the built-in I/O devices as follows:

- Serial port 1 (RS-232 device) firmware entry points are on page $C1. Much, but not all, of the firmware for the port is in the $C100 space.
- Serial port 2 (communication device) firmware entry points are on page $C2. Much, but not all, of the firmware for the port is in the $C100 space.

- Video output firmware entry points are on page $C3; the enhanced video firmware and miscellaneous I/O support routines occupy pages $C8 through $CF. This is partly because there are no slots 8 through F on the Apple IIc and because the firmware takes up more than one page of firmware memory space.
- Mouse firmware entry points are on page $C4 (page $C7 in the memory expansion version).
- Block device I/O firmware entry points are on page $C6.

❖ *Note:* This correspondence of ports and entry points does not imply that all of each port's firmware occupies a specific page. The Apple IIc I/O port firmware space is allocated in a way that provides the best possible performance in the available space.

The operation of the Applesoft interpreter firmware is described in the *Applesoft BASIC Programmer's Reference Manual.*

The ROM address range of pages $D0 through $FF contain the Applesoft BASIC interpreter and the Monitor firmware, allocated as follows:

- Pages $D0 through $F7 (addresses $D000 through $F7FF) contain the Applesoft interpreter firmware.
- Pages $F8 through $FF (addresses $F800 through $FFFF) contain the Monitor, described in Chapter 10. You can use some of the built-in Monitor routines to make input and output procedures in your assembly-language programs easier to write. These routines are described in Chapters 3 through 9.

Hardware addresses ($C000–$C0FF)

The soft switches that the Apple IIc and your programs use to control the Apple IIc's built-in input and output functions are all found in the $C0 memory page (addresses $C000 through $C0FF). In the same range of memory are the switches for selecting blocks of memory throughout the address space. This chapter describes the address space (memory) switches.

Chapters 3 through 9 describe the Apple IIc's input and output locations. Appendix B lists these locations in address order, rather than by function.

The hardware functions of the switches in this page fall into five basic categories:

- *Data inputs.* The only data input is location $C000, where the low-order seven bits (bits 6 through 0) represent the keyboard key just pressed. (These data are guaranteed valid only when bit 7 = 1.)

Bit numbering in a byte is explained in Appendix H.

- *Flag inputs.* Most built-in input locations are single-bit flags in the high-order (bit 7) position of their respective memory addresses. Flags have only two values: on (greater than or equal to 128 or $80) or off (less than 128 or $80).

The switch, hand controller (analog) and button inputs, and the keyboard strobe are examples of flag inputs. The locations for reading soft-switch states are also of this type.

- *Strobe outputs.* The clear keyboard strobe (Chapter 4) and paddle timer strobe (Chapter 9) outputs are controlled by memory locations. If your program reads the contents of one of these locations, then the function associated with that location will be activated.
- *Toggle switches.* The Apple IIc has only one toggle switch: the speaker switch. A toggle switch has only one address assigned to it; each time you access it, it changes to its other state (on or off).

  Reading the speaker toggle at location $C030 clicks the speaker once. However, if you write to the speaker location, the microprocessor activates the address bus twice during successive clock cycles, causing the speaker toggle to end up in its original state before the speaker cone can move. Therefore, you should read, rather than write, to use this device.

  The processor cannot read the on/off status of the speaker switch.
- *Soft switches.* Soft switches are two-position switches turned on by accessing one address and turned off by accessing another address. Most of these switches have a third address associated with them for reading the state of the switch.

  There are eight soft switches that select different combinations of bank-switched memory. Four of these eight switches require that your program read them twice in succession to activate them.

Bank-switched memory

The memory areas described in this section are called *bank-switched memory* (Figure 2-3) because so many banks (ranges) of addresses—one bank of ROM and up to four banks of RAM—occupy the same group of locations among the upper addresses of memory. Pages $00 and $01, at the low end of memory, are included here because the two sets of them—one in main RAM and one in auxiliary RAM—are controlled by the same switches as the high-address banks. The stack and zero page are switched this way so that system software running in the bank-switched memory space can maintain its own stack and zero page while it manipulates the 48K memory space.

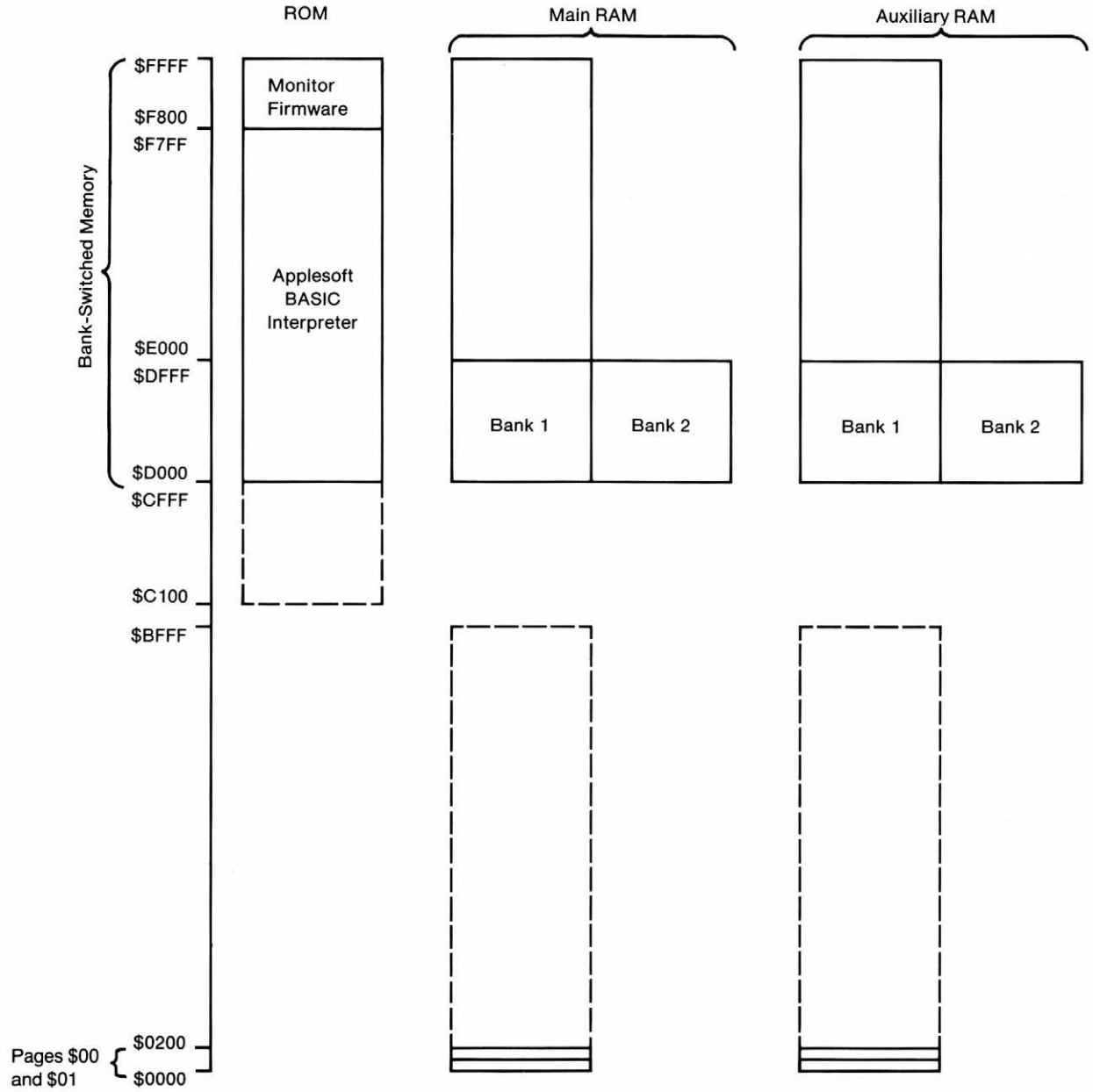


Figure 2-3
Bank-switched memory map

Page allocations

Pages $00 and $01 are used by many of the 65C02 instructions. The ROM and RAM addresses in bank-switched memory are usually occupied by system software such as interpreters, compilers, and operating systems.

Page $00 (one-byte addresses)

Several of the 65C02 microprocessor's addressing modes—for example, indirect addressing—require the use of addresses in page $00, or zero page. However, the Monitor, the interpreters, and the operating systems all make extensive use of page $00, too. One way to avoid conflicts is to use only those page-$00 locations not already used by these other programs. But there is another way.

As you can see from Table B-1 in Appendix B, page $00 is pretty well used up, except for a few bytes here and there. Rather than trying to squeeze your data into an unused corner, you may prefer a safer alternative: turn off interrupts, save the contents of part of page $00, use that part, then restore the previous contents to page $00, restore interrupts to their previous state, and then pass control to another program.

Page $01 (the 65C02 stack)

The 65C02 microprocessor uses page $01 as its stack—a place where it can store subroutine return addresses, in last-in, first-out sequence. Programs can also use the stack for temporary storage of registers (via push and pull instructions). However, programs should use the stack carefully.

Pages $D0–$FF (ROM and RAM)

These memory banks are controlled by the soft switches described under "Using Bank Selector Switches."

The memory address space from locations $D000 through $FFFF is used for both ROM and RAM. The 12K bytes of ROM in this address space contain the Monitor and the Applesoft BASIC interpreter.

There are 16K bytes of main RAM in this 12K space, with two banks occupying the 4K of addresses from $D000 through $DFFF. The RAM is normally used for storing other languages such as Pascal, or operating systems such as ProDOS®.

There are also 16K bytes of auxiliary RAM in this 12K space, again with double occupancy in the address range $D000 through $DFFF.

Using bank selector switches

You switch banks of memory in the same way you switch other functions in the Apple IIc: by using soft switches. These soft switches do four things:

- select either RAM or ROM in this memory space
- allow or inhibit (write-protect) writing to the RAM when RAM is selected
- select the first or second 4K-byte bank of RAM in the address space $D000 through $DFFF
- select either main RAM or auxiliary RAM

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

Table 2-1 shows the addresses of the soft switches for selecting all allowed combinations of reading and writing in this memory space, and the addresses of the locations to read the switch settings. Figures 2-4 through 2-10 illustrate how to select the combinations and what the resulting status of each switch is.

To make sure you do not inadvertently remove write protection from bank-switched RAM, the four write-enable addresses require that you read them twice in succession (indicated by *RR* in Table 2-1).

Because the AltZP switch shares the read keyboard address, you must write (*W* in Table 2-1) to its locations to change the switch setting.

To find out which way a switch is set, read the appropriate location and then check bit 7 (shown as *R7* in Table 2-1). If the bit is a 1, the answer to the question given in the table is affirmative.

Note that there is no way to check whether write protection is on or off.

Important You can't read one RAM bank and write to the other; if you select either RAM bank for reading, you get that one for writing as well. However, you can read ROM and write RAM (Figures 2-5 and 2-6), which makes it easy to transfer firmware to bank-switched RAM if you want to use it with a program there.

Table 2-1
Bank selector switches

Name	Action	Hex	Dec	Function
	R	$C080	49280	Read RAM; no write; use $D000 bank 2
	RR	$C081	49281	Read ROM; write RAM; use$D000 bank 2
	R	$C082	49282	Read ROM; no write; use $D000 bank 2
	RR	$C083	49283	Read and write RAM; use $D000 bank 2
	R	$C088	49288	Read RAM; no write; use $D000 bank 1
	RR	$C089	49289	Read ROM; write RAM; use$D000 bank 1
	R	$C08A	49290	Read ROM; no write; use $D000 bank 1
	RR	$C08B	49291	Read and write RAM; use $D000 bank 1
RdBnk2	R7	$C011	49169	Read whether $D000 bank 2 (1) or bank 1 (0)
RdLCRAM	R7	$C012	49170	Read RAM (1) or ROM (0)
AltZP	W	$C008	49160	Off: Use main bank, page $00 and page $01
AltZP	W	$C009	49161	On: Use auxiliary bank, page $00 and page $01
RdAltZP	R7	$C016	49174	Read whether auxiliary (1) or main (0) bank

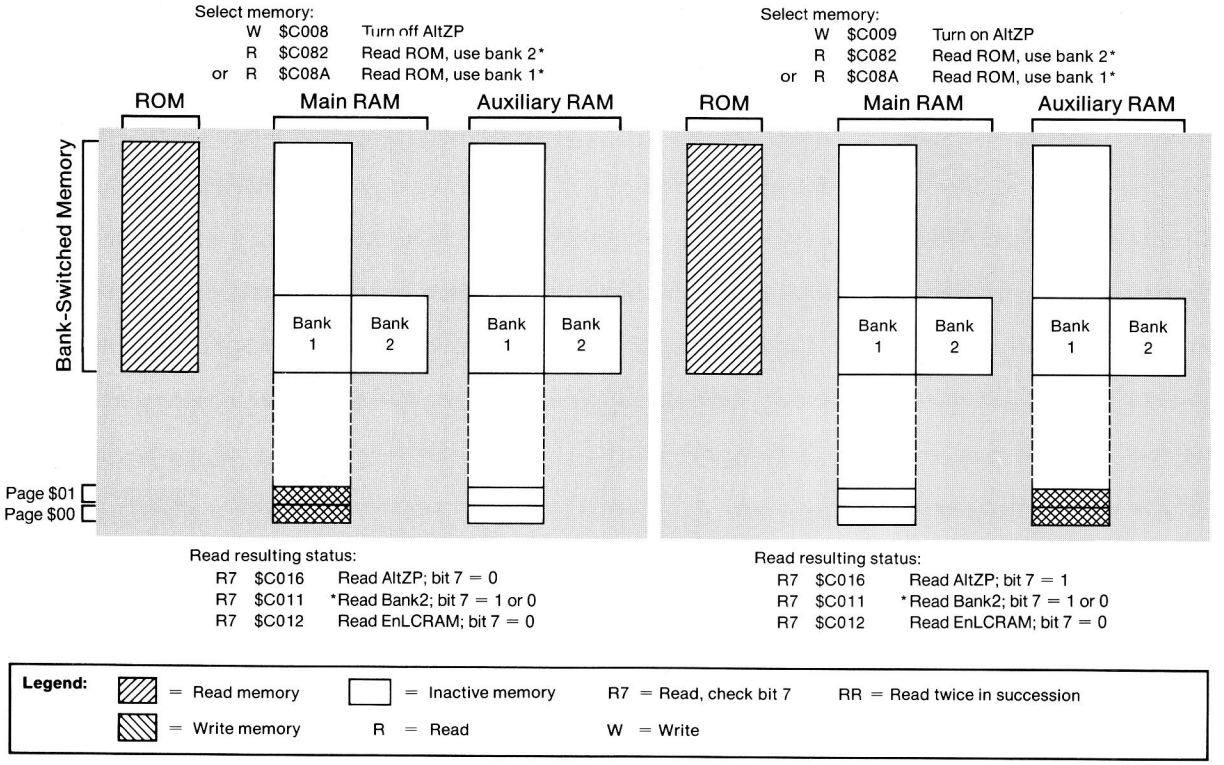


Figure 2-4
Read ROM

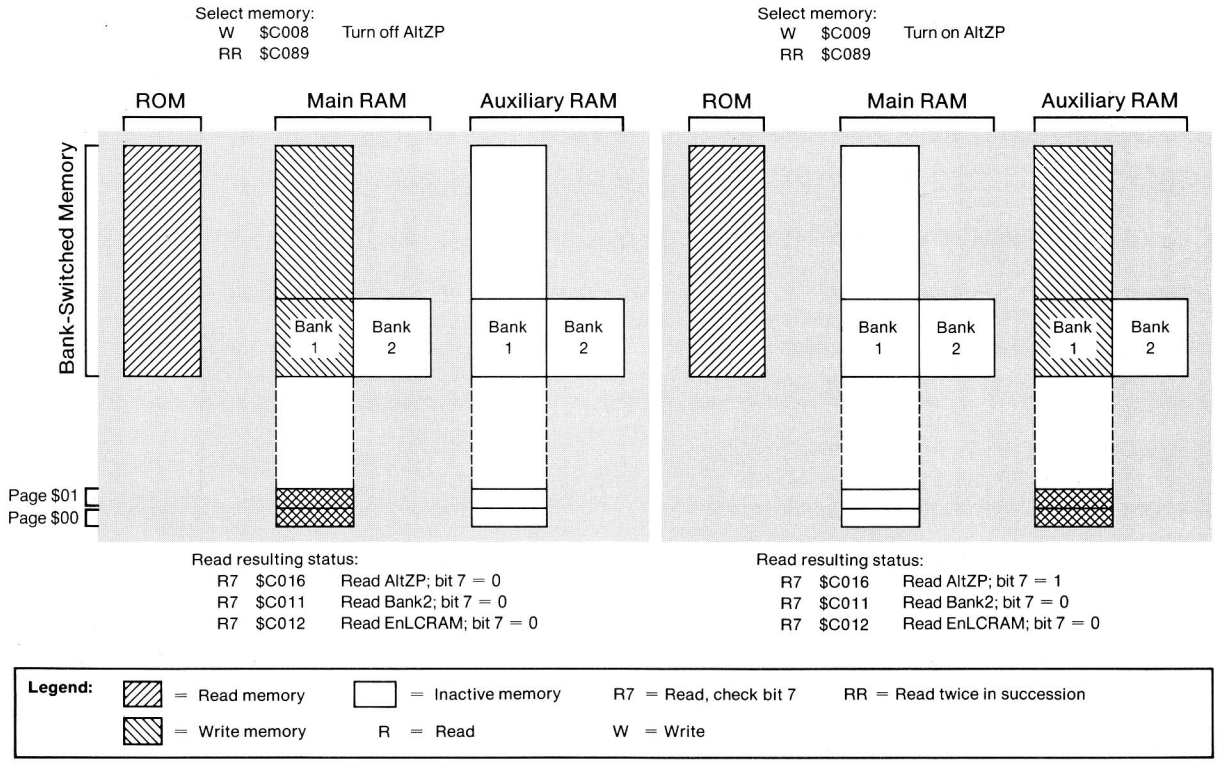


Figure 2-5
Read ROM, write RAM, and use first $D0 bank

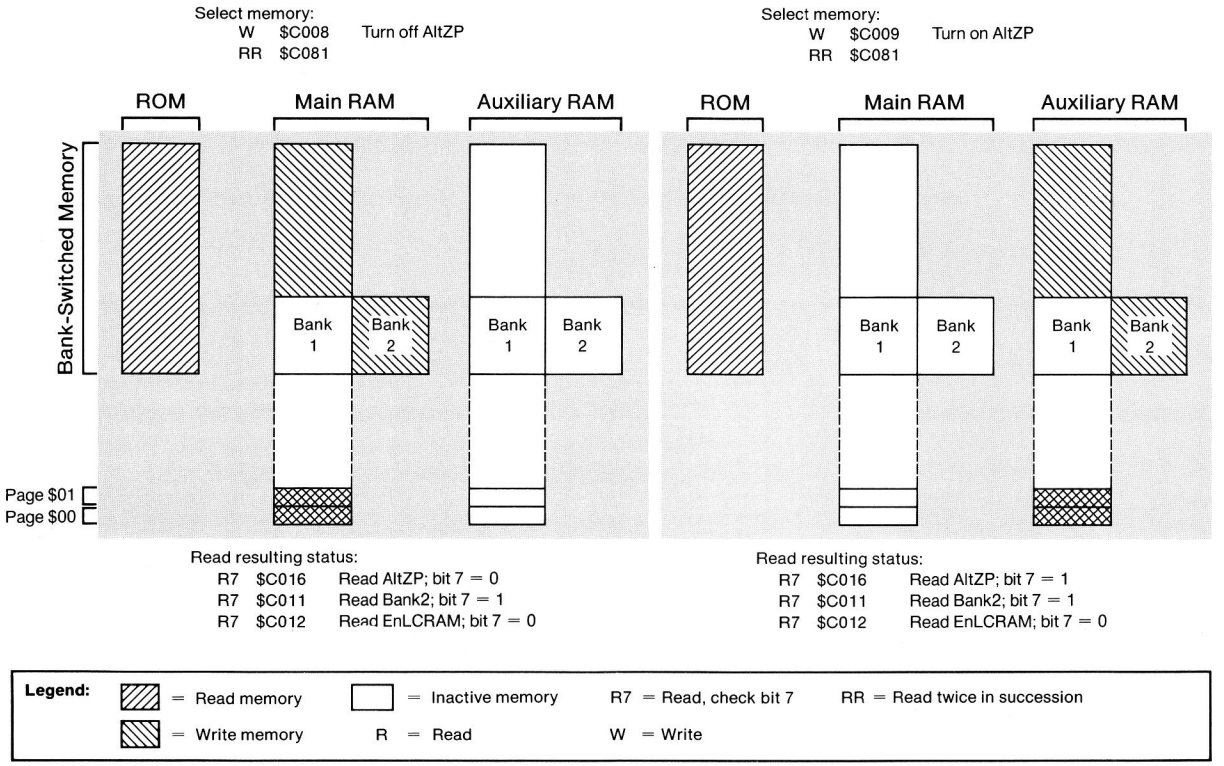


Figure 2-6
Read ROM, write RAM, and use second $D0 bank

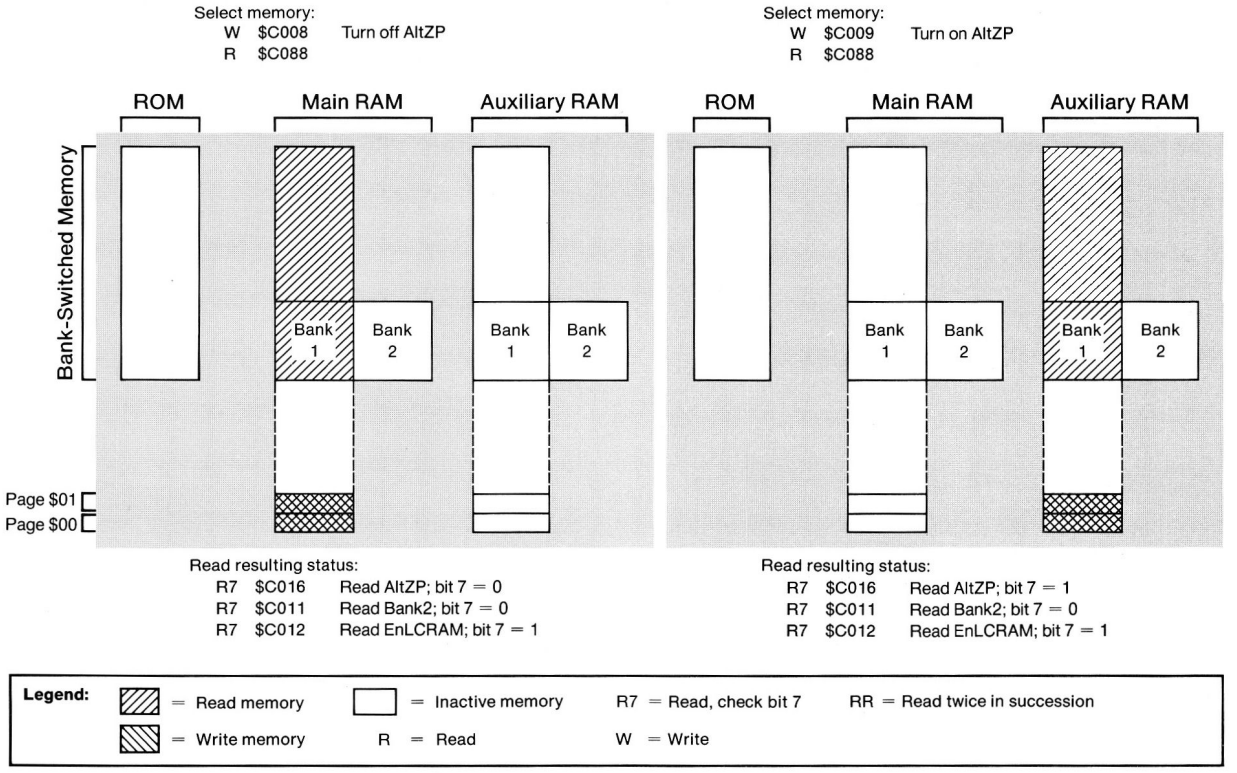


Figure 2-7
Read RAM and use first $D0 bank

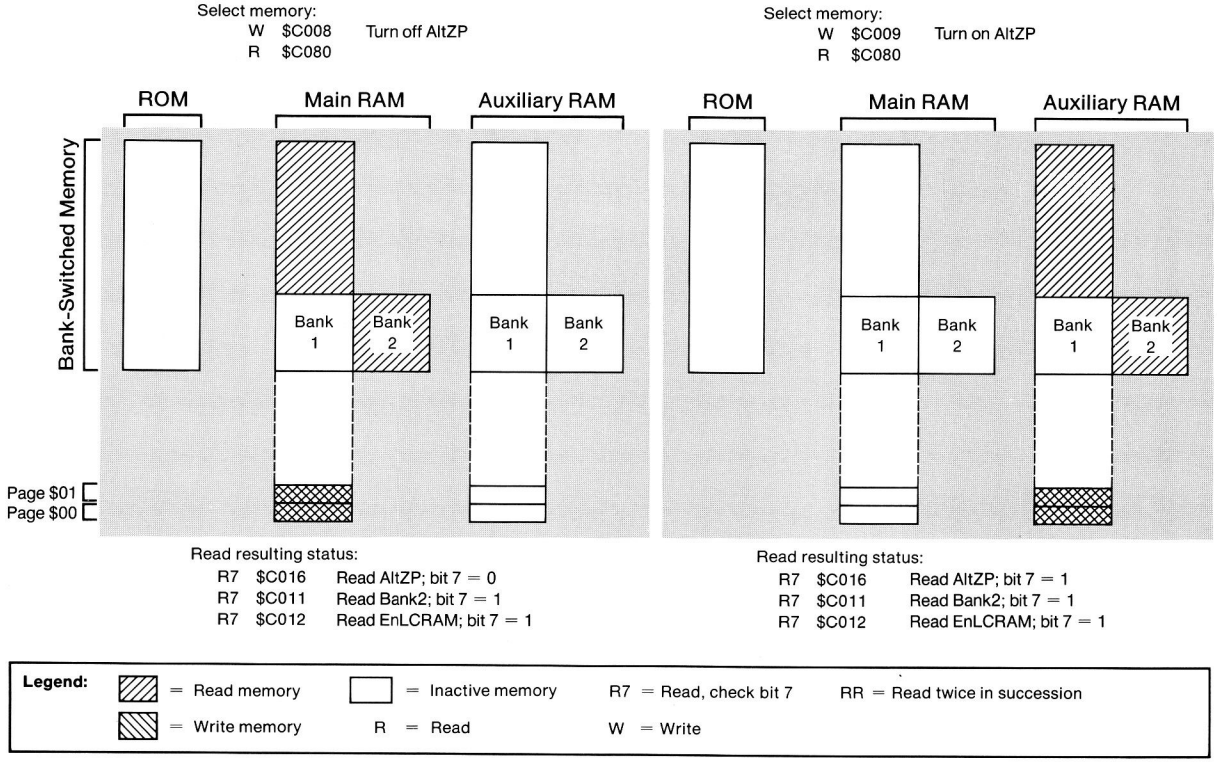


Figure 2-8
Read RAM and use second $D0 bank

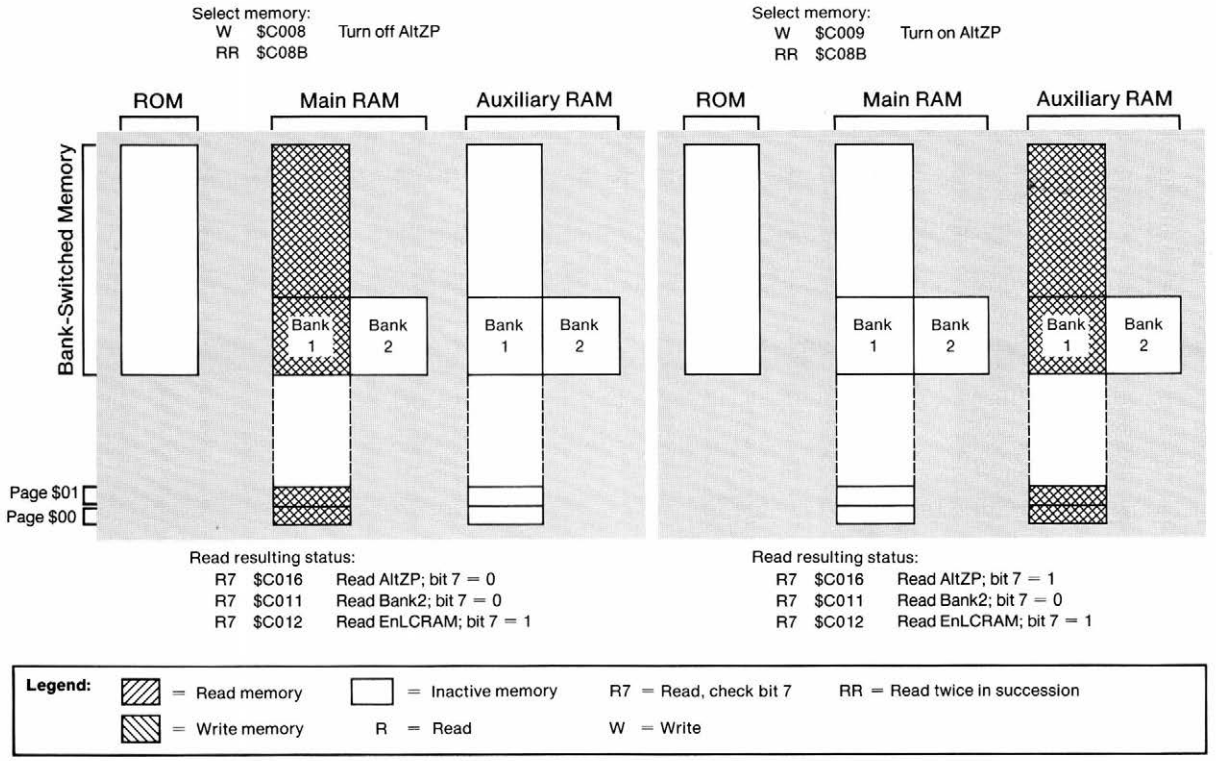


Figure 2-9
Read and write RAM and use first $D0 bank

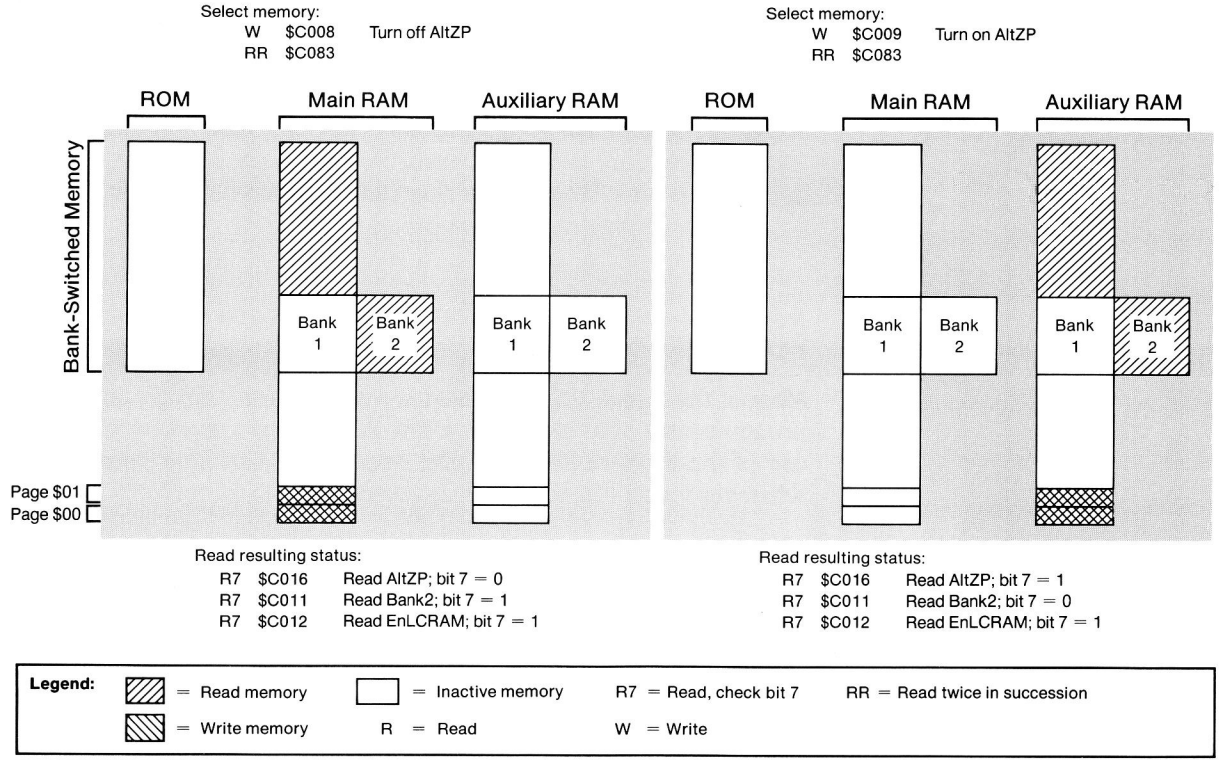


Figure 2-10
Read and write RAM and use second $D0 bank

48K memory

The 48K memory space (actually, 47.5K) extends from location $0200 to location $BFFF (Figure 2-11) in both main and auxiliary RAM. The amount of storage available in this address space depends on what language or operating system you are using, and what video display needs your program has.

Page allocations

Most of the Apple IIc's 48K RAM is available for storing your programs and data. However, a few RAM pages are reserved for the use of the Monitor firmware, the Applesoft BASIC interpreter, and whatever video display you may select.

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

Page $02 (the input buffer)

A **buffer** is any storage area set aside for one program or device to put information into and another to take information out of at a different time or rate.

The GetLn input routine uses page $02 as its keyboard-input **buffer.** The size of this buffer (256 bytes) sets the maximum size of input strings read by Applesoft or the Monitor. If you know that you won't be typing any long input strings (more than, say, 30 characters), you can store temporary data at the upper end of page $02.

Page $03 (global storage and vectors)

Refer to Appendix D and to the appropriate programmer and reference manuals for operating system use of page $03.

Global storage refers to an area reserved for information that programs use in common. **Vectors**—the addresses of special routines—are examples of this kind of information. See "The Reset Routine" about the global storage and vectors found on page $03.

The Monitor and operating systems use parts of page $03 for **global storage** and **vectors.** Table 2-7, later in this chapter, shows the part of page $03 the built-in firmware uses.

Pages $04–$07 (text and low-resolution Page 1)

The most often used display buffer is the text and low-resolution graphics Page 1 (TLP1 in Figure 2-11), which occupies main memory pages $04 through $07. It is not usable for program and data storage if you are using Monitor routines or Applesoft, or with almost any other program that uses text or low-resolution display.

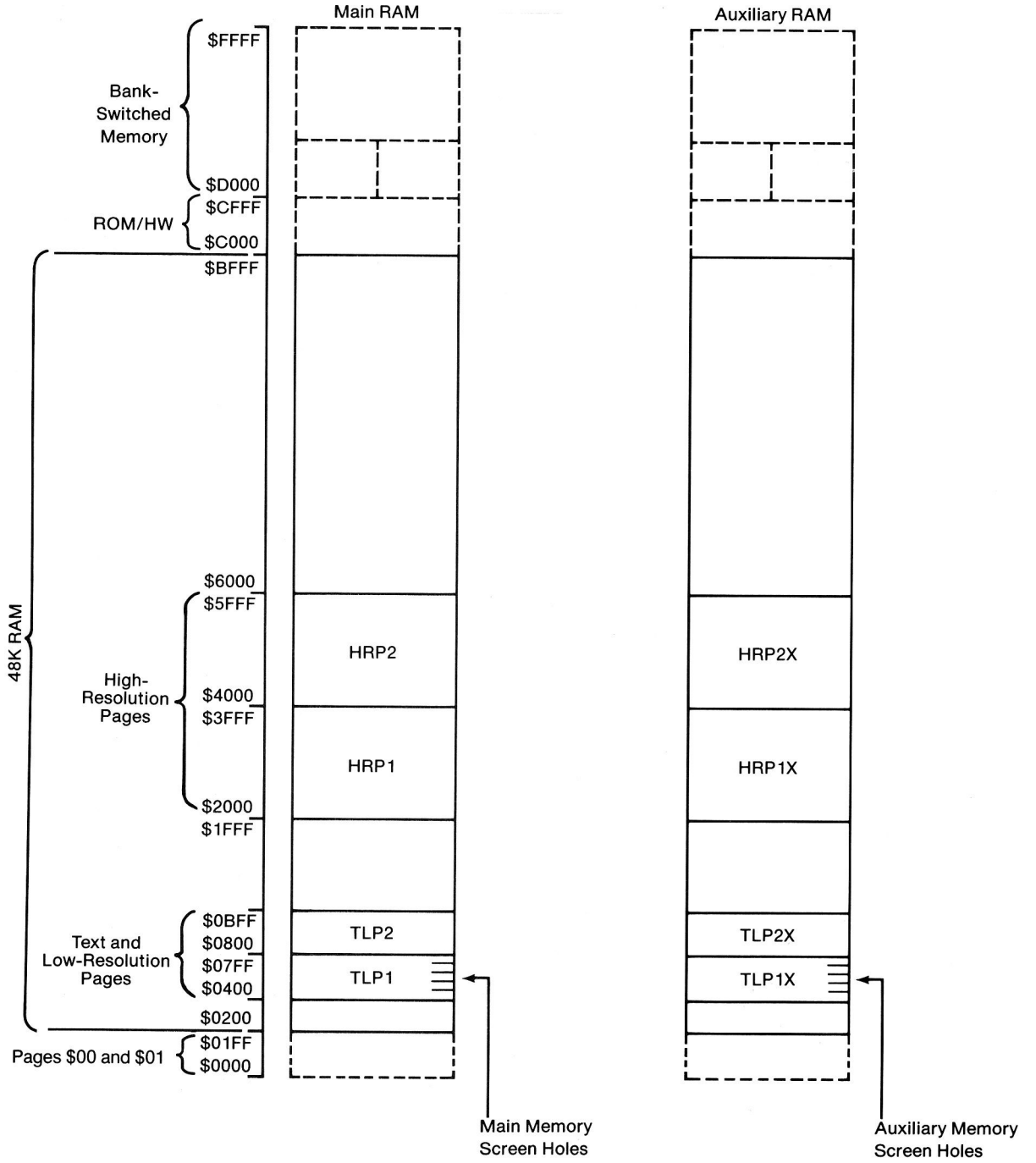


Figure 2-11
48K memory map

Text and low-resolution Page 1X (TLP1X) is an identical display page occupying auxiliary memory pages $04 through $07. This pair of text and low-resolution graphics pages are used together to produce 80-column text display.

See "Port Screen Hole RAM Space" in Chapter 3.

There are 128 locations in pages $04 through $07 (64 in main RAM, 64 in auxiliary RAM) that are not displayed on the screen. These locations are called *screen holes*.

Warning The screen holes are reserved for use by the built-in firmware.

Pages $08–$0B (text and low-resolution Page 2)

The second text and low-resolution graphics display buffer, TLP2, occupies main memory pages $08 through $0B. Most programs do not use Page 2 for displays, but TLP2 is there for display use if required.

Text and low-resolution Page 2X (TLP2X) is an identical display buffer occupying pages $08 through $0B in auxiliary memory.

Note that Apple IIc firmware does not provide a way to use the second pair of text and low-resolution graphics pages for 80-column text display.

Page $08 (communication port buffers)

For more on serial port 2, see Chapter 8.

Serial port 2 uses the first half of auxiliary memory page $08 (addresses $0800 through $087F) as a keyboard input buffer, and the second half of the page (addresses $0880 through $08FF) as a serial input buffer. These buffers increase the data transfer rates possible with the serial communication port. Appendix E explains how to use these features. If your program does not use this page for buffers, it can use it as part of TLP2X.

Pages $20–$3F (high-resolution Page 1)

The primary high-resolution graphics display buffer, high-resolution Page 1 (HRP1), occupies the 32 memory pages from $20 through $3F (locations $2000 through $3FFF). If your program doesn't use high-resolution graphics, this area is usable for programs or data.

High-resolution Page 1X (HRP1X) is an identical display page occupying auxiliary memory pages $20 through $3F.

See Chapter 5.

The Apple IIc can display double high-resolution graphics by interleaving HRP1 and HRP1X.

Pages $40–$5F (high-resolution Page 2)

High-resolution Page 2 occupies main memory pages $40 through $5F (locations $4000 through $5FFF). Most programs use this area for program or data storage, but it is also available as a second high-resolution page.

High-resolution Page 2X (HRP2X) occupies auxiliary memory pages $40 through $5F.

For more information about the display buffers, see Chapter 5.

Apple IIc firmware provides high-resolution graphics routines for HRP1 and HRP2 only. Refer to the *Applesoft BASIC Programmer's Reference Manual.*

Using 48K memory switches

For details, refer to "Using Display Memory Switches."

Two switches select main or auxiliary RAM in the 48K memory space: RAMRd determines which to use for reading, and RAMWrt determines which to use for writing. When these switches are on, they select auxiliary memory. When they are off, they select main memory. (This discussion assumes that the 80Store switch, used to control display memory, is off.)

Each switch has three locations assigned to it (Table 2-2): one to turn it on, one to turn it off, and a third to read its state. Because the memory locations for turning the switches on and off are shared with keyboard reading functions, you must write to these addresses to use them for memory switching. For each switch, you can read bit 7 at its third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0.

Table 2-2
48K memory switches

Name	Action	Hex	Dec	Function
RAMRd	W	$C002	49154	Off: Read main 48K RAM
RAMRd	W	$C003	49155	On: Read auxiliary 48K RAM
RdRAMRd	R7	$C013	49171	Read whether main (0) or aux. (1)
RAMWrt	W	$C004	49156	Off: Write to main 48K RAM
RAMWrt	W	$C005	49157	On: Write to auxiliary 48K RAM
RdRAMWrt	R7	$C014	49172	Read whether main (0) or aux. (1)

Note: 80Store must be off to switch all memory in this range, including display memory (Table 2-6).

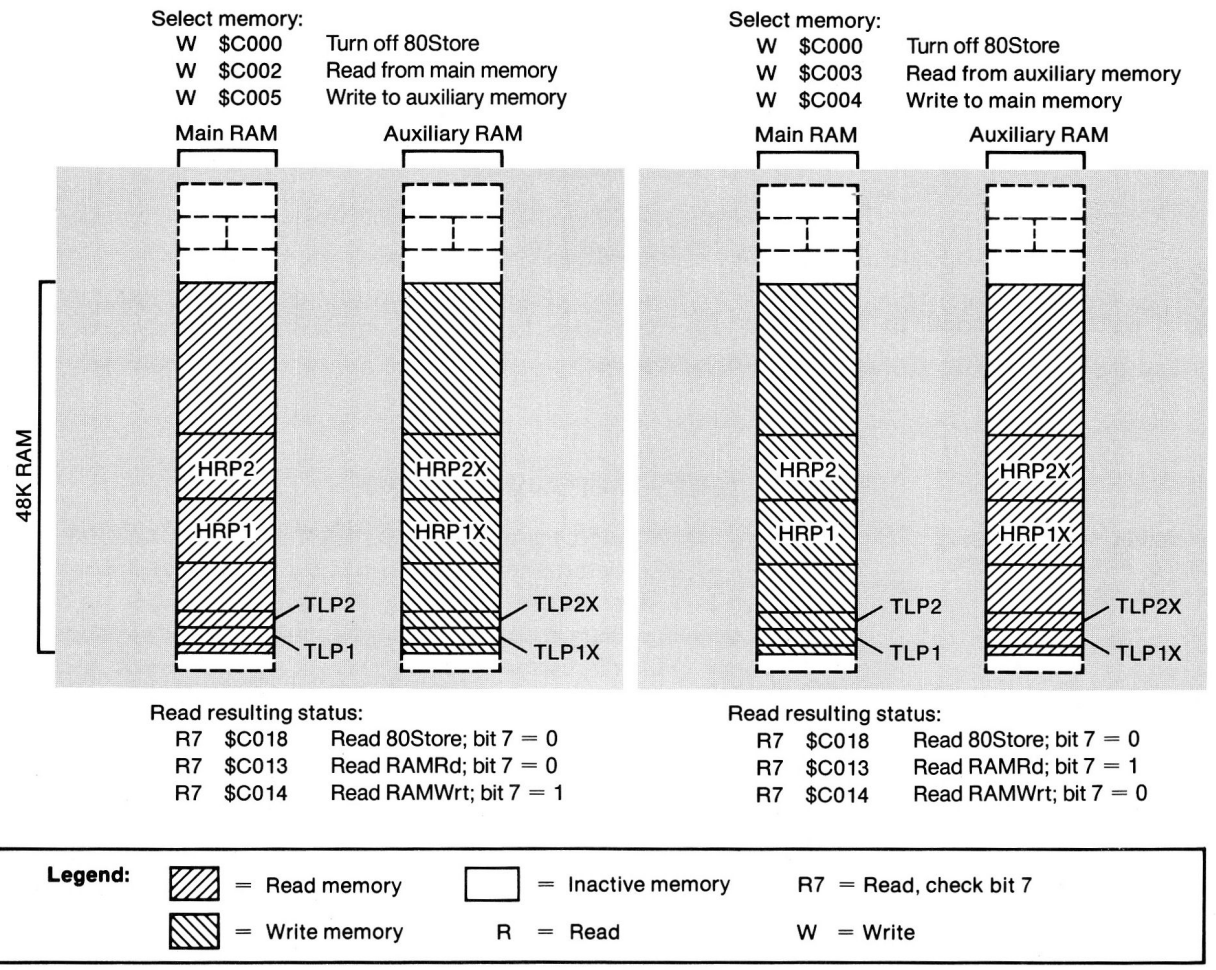


Figure 2-12
48K RAM selection, split pairs

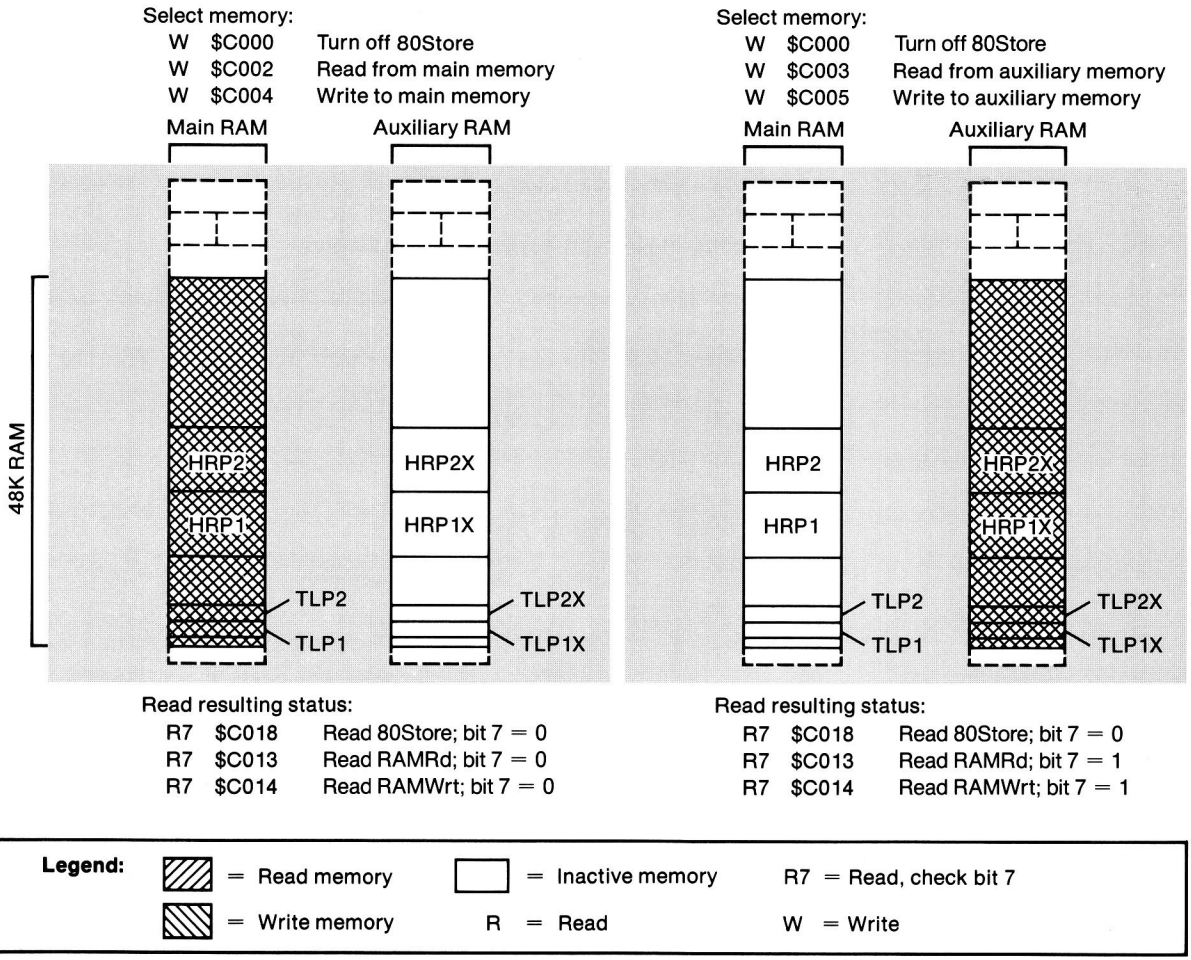


Figure 2-13
48K RAM selection, one side only

Transfers between main and auxiliary memory

If you want to write assembly-language programs that use auxiliary memory but you don't want to manage the auxiliary memory yourself, you can use the built-in 48K RAM transfer routines. These routines (listed in Table 2-3) make it possible to move between main and auxiliary memory without having to manipulate the soft switches described earlier in this chapter.

Important The routines described below make it easier to use auxiliary memory, but they do not protect you from errors. You still have to plan your use of auxiliary memory to avoid catastrophic effects on your program.

Table 2-3
48K RAM transfer routines

Name	Action	Hex	Function
MoveAux	JSR	$C311	Move data blocks between main and auxiliary 48K memory.
XFer	JMP	$C314	Transfer program control between main and auxiliary 48K memory.

Transferring data

In your assembly-language programs, you can use the built-in routine named MoveAux to copy blocks of data from main memory to auxiliary memory or from auxiliary memory to main memory. Before calling this routine, you must put the data addresses into byte pairs in page $00 and set or clear the carry bit to select the direction of the move.

Warning Don't try to use MoveAux to copy data in bank-switched memory (page $00, page $01, or pages $D0 through $FF). MoveAux uses page $00 all during the copy.

The pairs of bytes you use for passing addresses to this routine are called A1, A2, and A4, and they are used for parameter passing by several of the Apple IIc's built-in routines. The addresses of these byte pairs are shown in Table 2-4.

Put the addresses of the first and last bytes of the block of memory you want to copy into A1 and A2. Put the starting address of the block of memory you want to copy the data to into A4.

Table 2-4
Parameters for MoveAux routine

Name	Location	Parameter passed
Carry		1 = Move from main to auxiliary memory. 0 = Move from auxiliary to main memory.
A1L	$3C	Source starting address, low-order byte.
A1H	$3D	Source starting address, high-order byte.
A2L	$3E	Source ending address, low-order byte.
A2H	$3F	Source ending address, high-order byte.
A4L	$42	Destination starting address, low-order byte.
A4H	$43	Destination starting address, high-order byte.
	X, Y, A	These registers are preserved.

The MoveAux routine uses the carry bit to select the direction to copy the data. To copy data from main memory to auxiliary memory, set the carry bit (SEC instruction); to copy data from auxiliary memory to main memory, clear the carry bit (CLC instruction).

When you make the subroutine call to MoveAux, the subroutine copies the block of data as specified by the A register and the carry bit. When it is finished, the accumulator and the X and Y registers are just as they were when you called it.

Transferring control

You can use the built-in routine named *XFer* to transfer control to and from program segments in auxiliary memory. You must set up three parameters before using XFer: the address of the routine you are transferring to, the direction of the transfer, and which page $00 and stack you want to use (Table 2-5).

Table 2-5
Parameters for XFer routine

Name	Location	Parameter passed
Carry		1 = Transfer from main to auxiliary memory. 0 = Transfer from auxiliary to main memory.
Overflow		1 = Use page $00 and stack in auxiliary memory. 0 = Use page $00 and stack in main memory.
	$03ED	Program starting address, low-order byte.
	$03EE	Program starting address, high-order byte.
	X, Y, A	These registers are preserved.

Put the transfer address into the two bytes at locations $03ED and $03EE, with the low-order byte first, as usual. The direction of the transfer is controlled by the carry bit: set the carry bit to transfer to a program in auxiliary memory; clear the carry bit to transfer to a program in main memory.

Use the overflow bit to select which page $00 and stack you want to use: clear the overflow bit to use the main memory; set the overflow bit (cause an overflow condition) to use the auxiliary memory.

After you have set up the parameters, pass control to the XFer routine by a jump instruction, rather than a subroutine call.

Warning It is your responsibility as the programmer to save the current stack pointer before using XFer and to restore it after regaining control. Failure to do so will cause program errors. Refer to Appendix E for instructions on how to do this.

Using display memory switches

Selection of main or auxiliary RAM for the 48K memory space is described earlier in this chapter. However, under many circumstances your program may want to control reading and writing to display pages separately. The switches discussed in this section override the effects of RAMRd and RAMWrt for display pages only.

Three switches are involved in the display page selection process. Each of them has three locations assigned to it: one to turn it on, one to turn it off, and a third to read its state (Table 2-6). One of the switches, 80Store, shares its on and off addresses with a keyboard reading function. As a result, your program must write to these locations to turn the switch on and off.

Table 2-6
Display memory switches

Name	Action	Hex	Dec	Function
80Store	W	$C000	49152	Off: RAMRd and RAMWrt determine RAM locations.
80Store	W	$C001	49153	On: Page2 switches between TLP1 and TLP1X, and (if HiRes on) between HRP1 and HRP1X.
Rd80Store	R7	$C018	49176	Read whether 80Store on (1) or off (0).
Page2	R	$C054	49236	Off: Select TLP1 and HRP1.
Page2	R	$C055	49237	On: If 80Store off, switch to TLP2, and (if HiRes on) to HRP2. If 80Store on, switch to TLP1X, and (if HiRes on) to HRP1X.
RdPage2	R7	$C01C	49180	Read whether Page2 on (1) or off (0).
HiRes	R	$C056	49238	Off: Display text and low-resolution page.
HiRes	R	$C057	49239	On: Display high-resolution pages; make Page2 switch between high-resolution pages.
RdHiRes	R7	$C01D	49181	Read whether HiRes on (1) or off (0).
IOUDis	W	$CO7E	49278	On: Disable IOU access for addresses$CO58 to $CO5F; enable access to DHiRes switch*.
IOUDis	W	$CO7F	49279	Off: Enable IOU access for addresses $CO58 to $CO5F; disable access to DHiRes switch*.

Table 2-6 (continued)
Display memory switches

Name	Action	Hex	Dec	Function
RdIOUDis	R7	$CO7E	49278	Read IOUDis switch (1=off)†.
DHiRes	R/W	$CO5E	49246	On: (If IOUDis on) turn on double high-resolution.
DHiRes	R/W	$CO5F	49247	Off: (If IOUDis on) turn off double high-resolution.
RdDHiRes	R7	$CO7F	49279	Read DHiRes switch (1=on)†.

* The firmware normally leaves IOUDis on.
† Reading or writing any address in the range $CO70–$CO7F also triggers the paddle timer and resets VBlInt (see Chapter 9).

For each switch, you can read bit 7 at its third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0.

Here is how these switches work for reading and writing:

- If HiRes is off, then Page2 switches between text and low-resolution graphics pages (TLP) only. If HiRes is on, then Page2 switches between TLP and high-resolution graphics pages (HRP).
- If 80Store is off, RAMRd and RAMWrt (Table 2-2) determine whether main or auxiliary RAM locations are used. Page2 selects pages for display (Chapter 5), but not for reading and writing.
- If 80Store is on, it overrides RAMRd and RAMWrt with respect to the display pages selected by HiRes and Page2 (Figures 2-14 and 2-15).

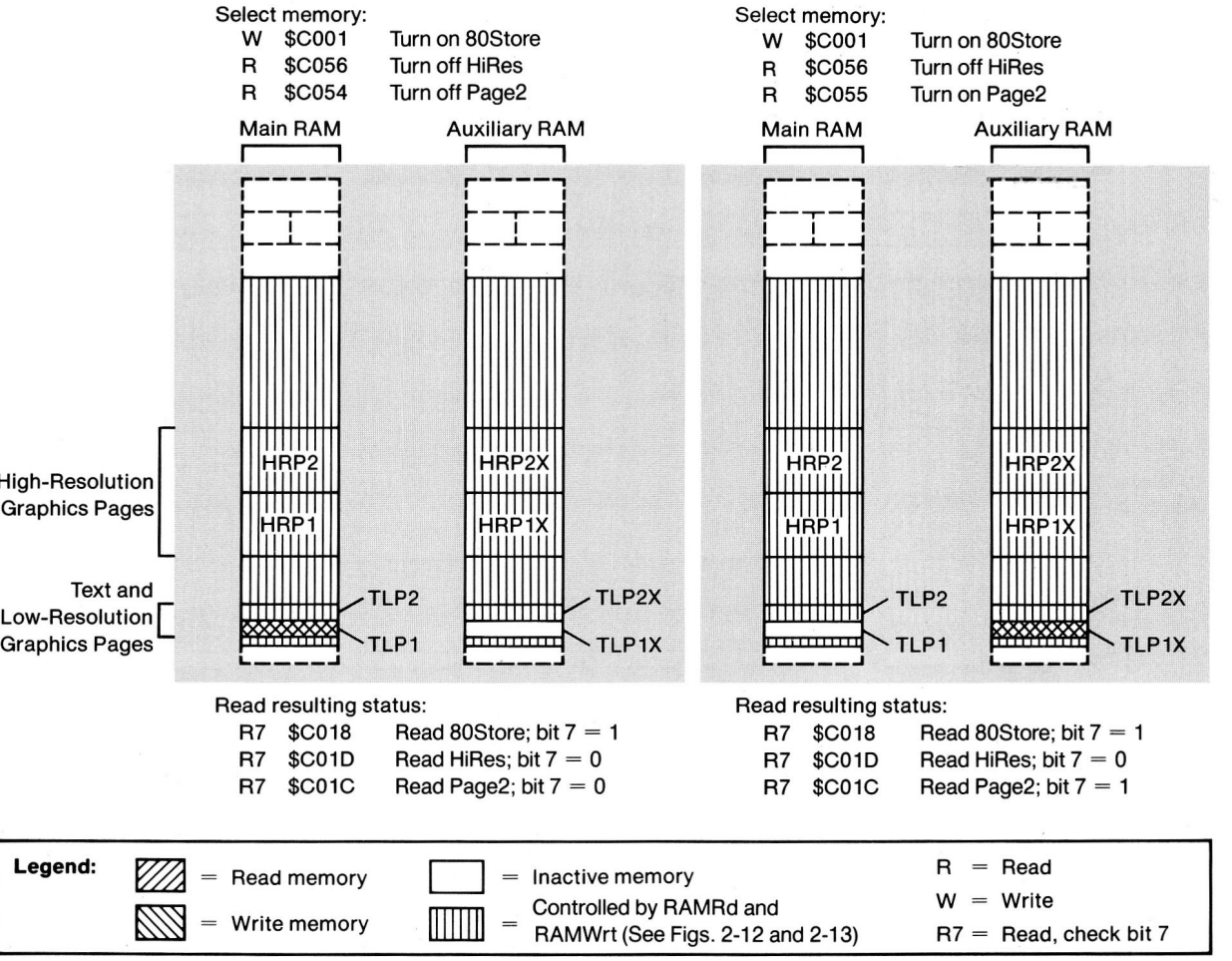


Figure 2-14
Page2 selections, 80Store on and HiRes off

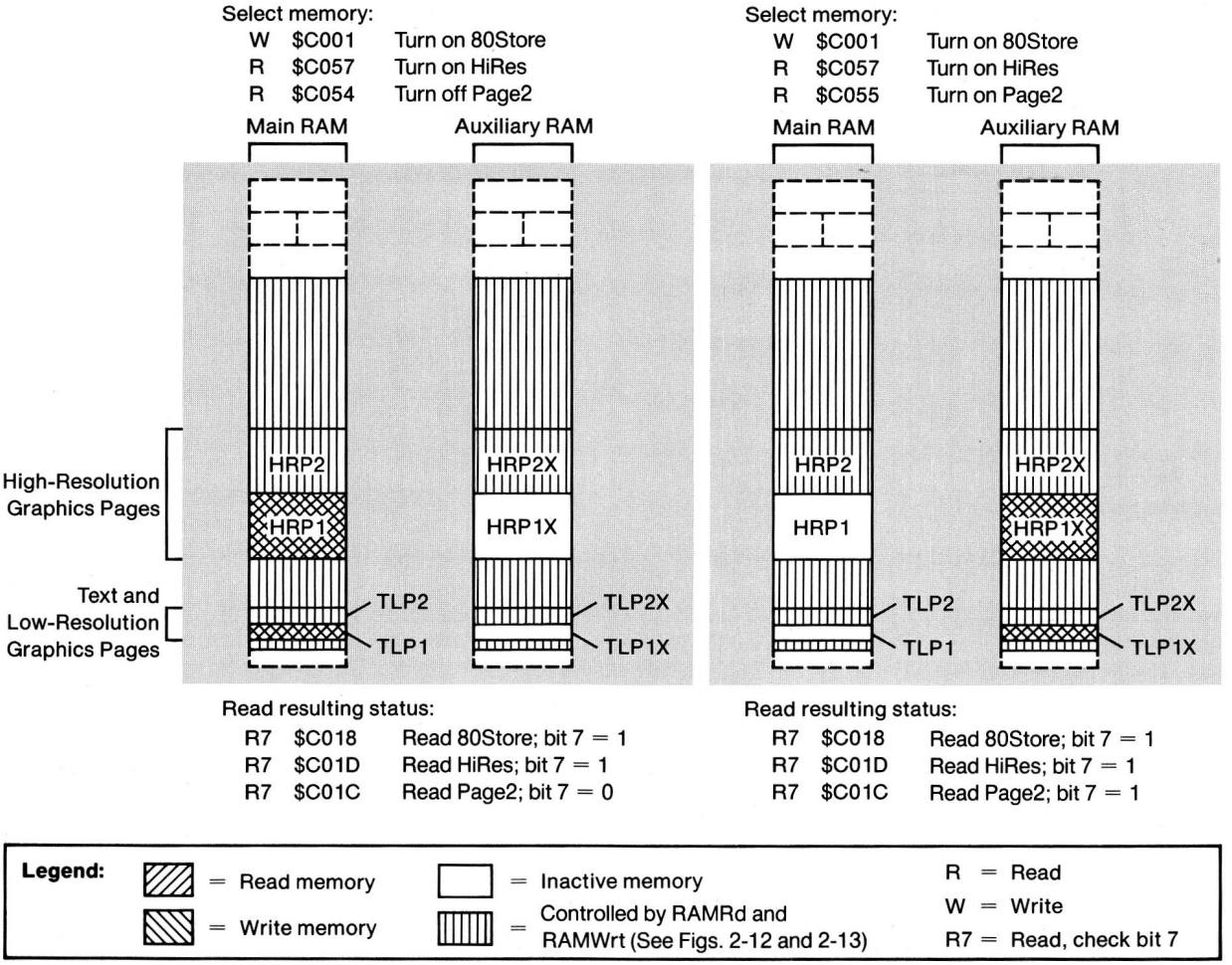


Figure 2-15
Page2 selections, 80Store on and HiRes on

The reset routine

A procedure called the *reset routine* (Figure 2-16) puts the Apple IIc into a known state when it has just been turned on or when you hold down Control while pressing Reset. The reset routine puts the Apple IIc into its normal operating mode and restarts the program indicated at locations $03F2 and $03F3 (Table 2-7).

When you initiate a reset, hardware in the Apple IIc sets the memory-controlling soft switches to normal: main ROM and RAM are enabled, auxiliary RAM is disabled and the bank-switched memory space is set up to read from ROM and write to RAM, using the second bank at $D000.

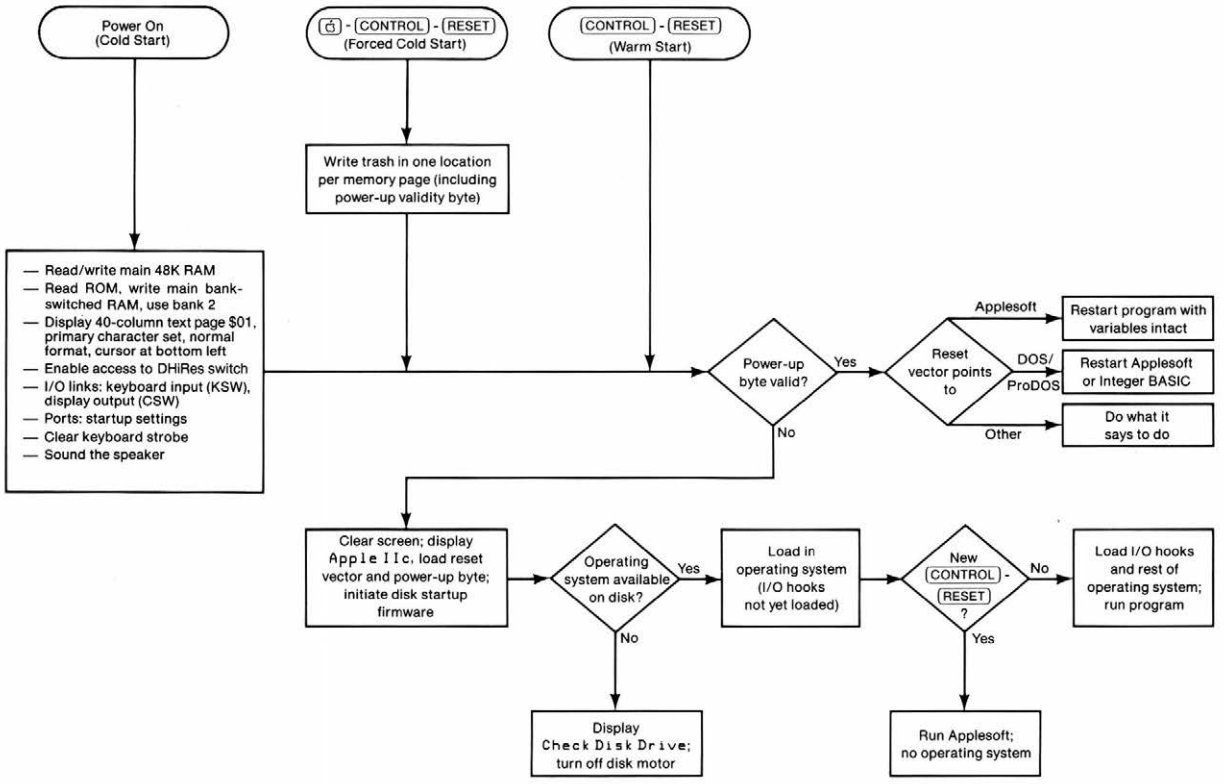


Figure 2-16
Reset routine flowchart

Table 2-7
Page $03 vectors

Vector address	Vector function
$03F0 (1008) $03F1 (1009)	Address of the subroutine that handles BRK requests (normally $59, $FA)
$03F2 (1010)	Reset vector (see text) $03F3 (1011)
$03F4 (1012)	Power-up byte (see text)
$03F5 (1013) $03F6 (1014) $03F7 (1015)	Jump instruction to the subroutine that handles Applesoft and commands (normally $4C,$58,$FF)
$03F8 (1016) $03F9 (1017) $03FA (1018)	Jump instruction to the subroutine that handles user Control-Y commands
$03FB (1019) $03FC (1020) $03FD (1021)	Jump instruction to the subroutine that handles nonmaskable interrupts (not used on Apple IIc)
$03FE (1022) $03FF (1023)	Interrupt vector (address of the subroutine that handles interrupt requests) (Appendix E)

The reset routine sets the display-controlling soft switches to display 40-column text Page 1 using the primary character set, then sets the display window equal to the full 40-column display, puts the cursor at the bottom of the screen, and sets the text display format to normal.

The reset routine also sets the keyboard and display as the standard input and output devices (Chapter 3). It masks mouse interrupts and sets mouse defaults (Table 9-1). Finally, it enables DHiRes switch access (by turning on IOUDis), clears the keyboard strobe, and sounds the speaker.

The Apple IIc has three types of reset: *power-on reset,* also called cold-start reset; *warm-start reset;* and *forced cold-start reset.* The procedure described above is the same for any type of reset. What happens next depends on the reset vector. The reset routine checks the reset vector to determine whether it is valid or not. If the reset was caused by turning the power on, the vector will not be valid, and the reset routine will perform the cold-start procedure. If the vector is valid, the routine will perform the warm-start procedure.

The reset vector validity check is described under "The Reset Vector."

The cold-start procedure (power on)

If the reset vector is not valid, either the Apple IIc has just been turned on or something has caused memory contents to be changed. The reset routine clears the display and puts the string `Apple© IIc` at the top of the display. It loads the reset vector and the validity-check byte, then initiates the startup routine that resides in the disk controller firmware. The bootstrap routine then loads whatever operating system resides on the disk in the built-in drive. When the operating system has been loaded, it displays other messages on the screen. If there is no disk in the disk drive, the drive motor keeps spinning for a brief time. Then the firmware shuts it off and displays the message `Check Disk Drive` at the bottom of the screen.

If you press Control-Reset again before the startup procedure is completed, the reset routine continues without using the disk, and passes control to the Applesoft BASIC interpreter.

The warm-start procedure (Control-Reset)

Whenever you press Control-Reset when the Apple IIc has already completed a cold-start reset, the reset vector is still valid and it is not necessary to reinitialize the entire system. The reset routine simply uses the vector to transfer control to the program it points to, which at power-up is the Applesoft interpreter.

If the vector does point to the Applesoft interpreter, your Applesoft program and variables are still intact. If you are using DOS or ProDOS, that operating system is the resident program and it restarts the BASIC interpreter you were using when you pressed Control-Reset.

Important A program residing only in bank-switched RAM cannot use the reset vector to regain control after a reset, because upon reset the hardware selects the ROM for reading in the bank-switched memory space.

Forced cold start (Open Apple-Control-Reset)

If a program has set the reset vector to point to its own warm-start address, as described below, pressing Control-Reset causes transfer of control to that program. If you want to stop such a program without turning the power off and on, you can force a cold-start reset by holding down Control and Open Apple, then pressing and releasing Reset.

Important When you want to stop a program unconditionally—for example, to start up the Apple IIc with some other program—you should use the forced cold-start reset, Open Apple-Control-Reset, instead of turning the power off and on.

UniDisk 3.5 You must hold Open Apple down until the built-in drive starts to spin. If you release Open Apple before the drive starts to spin, the Apple IIc drops into BASIC instead of rebooting.

The forced cold-start reset works as follows. First, it destroys the program or data in memory by writing two bytes of arbitrary data into each page of main RAM. The two bytes that get written over in page $03 are the ones that contain the reset vector. The warm-start reset routine finds the error, and so performs a normal cold-start reset.

Note that if you press both Open Apple and Solid Apple during power-up or Control-Reset, built-in exercise code is executed. This code is for production and has no end-user value.

The reset vector

The cold-start reset routine stores the starting address of the built-in Applesoft interpreter, low-order byte first, in the reset vector address at locations $03F2 and $03F3. It then stores a validity-check byte, also called the power-up byte, at location $03F4. The validity-check byte is computed by performing an exclusive-OR of the second byte of the vector with the constant 165 (hexadecimal $A5). Each time you reset the Apple IIc, the reset routine uses this byte to determine whether the reset vector is still valid.

You can change the reset vector so that the reset routine will transfer control to your program instead of to the Applesoft interpreter. For this to work, you must also change the validity-check byte to the exclusive-OR of the high-order byte of your new reset vector with the constant 165 ($A5). If you fail to do this, then the next time you reset the Apple IIc, the reset routine will determine that the reset vector is invalid and perform a cold-start reset, eventually transferring control to the disk bootstrap routine or to Applesoft.

There is a subroutine that generates the validity-check byte for the current reset vector. This subroutine, called *SetPWRC,* is at location $FB6F. When your program finishes, it can return the Apple IIc to normal operation by restoring the original reset vector and again calling the subroutine to fix up the validity-check byte.

Chapter 3

Introduction to Apple IIc I/O

This chapter is an introduction to the built-in I/O capabilities of the Apple IIc. It outlines

- standard I/O links and their functions
- I/O firmware protocols
- dedicated memory storage locations
- direct I/O

The next six chapters discuss these capabilities in detail.

The standard I/O links

You can use some of the routines in the Apple IIc's firmware for your own programs. This can save you both program space and the time and effort of writing all your own I/O routines.

To use the built-in firmware routines, your program must perform a JSR to the routine's entry address. The called routine then performs an indirect jump through an address stored somewhere in RAM and begins executing. When the routine has finished doing its work, it returns (with an RTS) to your program at the first instruction following the JSR used to call the routine. Memory locations used for transferring control to other subroutines, such as the indirect jump's address used by the character I/O routine, are sometimes called *vectors*. In this manual, the locations used for transferring control to the Apple IIc's I/O subroutines are called the *I/O links*.

In an Apple IIc running without an operating system, each I/O link normally contains the address of the standard input or output subroutine. An operating system will typically place addresses of its own I/O routines in these link locations instead.

By calling the I/O subroutines that then jump to the routines pointed to by the link addresses instead of calling the standard subroutines directly, you ensure that your program will work properly with other software, such as the operating system or a device driver. The I/O links contain the addresses of KeyIn and COut1 if the enhanced video firmware is off (when the display shows a flashing checkerboard cursor), and of C3KeyIn and C3COut1 if that firmware is on (when the display shows an inverse solid cursor).

The standard I/O links are two pairs of locations in the Apple IIc RAM in the range $36 through $39 that are used for controlling character input and output.

❖ *Note:* Not all operating systems use the standard I/O links. For example, Apple Pascal does not use them.

The link at locations $36 and $37 is called *CSW* (character output switch). Individually, location $36 is called *CSWL* (CSW low) and location $37 is called *CSWH* (CSW high). This link holds the starting address of the subroutine the Apple IIc is currently using for single-character output. This address is normally $FDF0, the address of routine COut1.

The Monitor is discussed in Chapter 10.

When you issue either a PR#n from BASIC or an n Control-P from the Monitor, the Apple IIc changes this link address to the first address in the ROM space allocated to port n. That address has the form $Cn00. Subsequent calls for character output are thus transferred to the firmware starting at that address. When it has finished, the firmware executes an RTS (return from subroutine) instruction to return control to the calling program. Sometimes a PR#n will cause both input and output switches to be changed (as in the 80-column firmware).

A similar link at locations $38 and $39 is called *KSW* (keyboard input switch). Individually, location $38 is called *KSWL* (KSW low) and location $39 is called *KSWH* (KSW high). This link holds the starting address of the routine currently being used for single-character input—normally $FD1B, the starting address of the standard input routine KeyIn.

When you issue an IN#n command from BASIC or an n Control-K from the Monitor, the Apple IIc changes the link address in KSW to $Cn00, the beginning of an I/O firmware subroutine. Subsequent calls for character input are thus transferred to that firmware. The firmware puts the input character, with its high bit set, into the accumulator and executes an RTS (return from subroutine) instruction to return control to the program that requested input.

When a disk operating system (DOS or ProDOS) is running, one or both of the standard I/O links hold addresses of the disk operating system's input and output routines. The operating system has internal locations that hold the addresses of the currently active character input and output routines.

Warning If a program that is running with DOS or ProDOS changes the standard link addresses, either directly or via IN# and PR# commands, the operating system may be disconnected from the system. To avoid this problem, when programming in BASIC you should always issue an empty PRINT statement (to be sure that what follows begins a new line) before issuing the PRINT statement containing Control-D and the IN# or PR# command.

Refer to the section on input and output link addresses in the operating system manuals for further details.

After changing either CSW or KSW, your assembly-language programs running under DOS should call the subroutine at location $03EA. This subroutine transfers the link address to a location inside the operating system and then restores the operating system link address in the standard link location.

Standard input features

GetLn also provides on-screen editing features. See "Editing With GetLn."

The Apple IIc's firmware includes two different subroutines for reading from the keyboard, *RdKey* (read key) and *GetLn* (get line).

RdKey calls the current character input routine (that is, the one whose address is stored at KSW). This is normally KeyIn or C3KeyIn, which accepts one character from the keyboard. GetLn accepts a *sequence* of characters terminated with a carriage return. Thus GetLn allows line-oriented input using the current input routine.

RdKey subroutine

A program can get a character from the keyboard by making a subroutine call to RdKey at memory location $FD0C. RdKey passes control via the input link KSW to the current input subroutine, which is normally KeyIn.

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 below).

KeyIn subroutine

KeyIn is the standard input subroutine. When your program calls it, KeyIn displays a cursor, waits until someone presses a key, then inserts the ASCII code of the key just pressed in the accumulator and returns to the calling program.

If the enhanced video firmware is inactive, KeyIn displays a cursor by alternately storing a checkerboard block in the cursor location, storing the original character, then storing the checkerboard again. If the firmware is active, C3KeyIn places a block cursor on the screen by inverting (swapping black for white) the character at the cursor position.

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 $4E and $4F. This number keeps increasing from 0 to $FFFF (65535), then starts over again at 0. The value of this number changes so rapidly that it is very difficult 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 pseudo–random number routine.

GetLn subroutine

Programs often need strings of characters as input. While you could call RdKey repeatedly to get several characters from the keyboard, there is an easier way to do it. The routine that you want to use in this case is named *GetLn,* and it 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 $0200 to $02FF. GetLn also provides you with some basic on-screen editing and control features.

The first thing GetLn does when you call it is to display 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. Table 3-1 shows the prompt characters used by different programs on the Apple IIc.

GetLn uses the character stored at memory location $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 3-1
Prompt characters

Prompt character	Program requesting input
?	User's BASIC program (INPUT statement)
]	Applesoft BASIC (Appendix D)
>	Integer BASIC (Appendix D)
*	Firmware Monitor (Chapter 10)

❖ *Note:* Applesoft uses GetLn1 ($FD6F) when a program is executing. GetLn1 does not print a prompt.

As the user types each character, GetLn sends the character to the standard output routine—normally COut1—which displays it at the current cursor position and then advances the cursor to indicate the next character position. Control characters echoed by GetLn are not executed.

GetLn stores the characters in its buffer, starting at memory location $0200 and using the X register to index the buffer. GetLn continues to accept and display characters until the user presses Return (or Control-X to cancel the line). Then it clears the remainder of the line the cursor is on, stores the carriage-return code to mark the end of the buffer, places the cursor at the beginning of the next line, and returns.

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.

❖ *Note:* The Applesoft interpreter accepts only 239 characters.

Escape codes with GetLn

GetLn has many special functions that you invoke by typing escape codes on the keyboard. An escape code is sent by pressing Escape, releasing it, and then pressing some other key, as shown in Table 3-2.

Important Be sure to release Escape right away. If you hold it too long, the auto-repeat mechanism begins, which may cancel the Escape.

Table 3-2
Escape codes with GetLn

Escape code	Function
Escape	Clears the window and homes the cursor (places it in the upper-left corner of the screen); exits from escape mode
Escape A or Escape a	Moves the cursor right one line; exits from escape mode
Escape B or Escape b	Moves the cursor left one line; exits from escape mode
Escape C or Escape c	Moves the cursor down one line; exits from escape mode
Escape D or Escape d	Moves the cursor up one line; exits from escape mode
Escape E or Escape e	Clears to the end of the line; exits from escape mode
Escape F or Escape f	Clears to the bottom of the window; exits from escape mode
Escape I or Escape i *or* Escape Up Arrow	Moves the cursor up one line; remains in escape mode
Escape J or Escape j *or* Escape Left Arrow	Moves the cursor left one space; remains in escape mode*
Escape K or Escape k *or* Escape Right Arrow	Moves the cursor right one space; remains in escape mode*
Escape M or Escape m *or* Escape Down Arrow	Moves the cursor down one line; remains in escape mode*
Escape 4	Switches to 40-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 3-5); exits from escape mode†

Table 3-2 (continued)
Escape codes with GetLn

Escape code	Function
Escape 8	Switches to 80-column mode; sets links to C3KeyIn and C3COut1; restores normal window size (Table 3-5); exits from escape mode†
Escape Control-D	Disables control characters; only carriage return, linefeed, bell, and backspace have an effect when printed
Escape Control-E	Reactivates control characters
Escape Control-Q	Deactivates the enhanced video firmware; sets links to KeyIn and COut1; restores normal window size (Table 3-5); exits from escape mode†

• Cursor-control key: see text.
† This code functions only when the enhanced video firmware is active.

In escape mode, you can keep using the arrow keys and the cursor movement keys I, J, K, and M without pressing Escape again. This enables you to perform repeated cursor moves by holding down the appropriate key.

When GetLn is in escape mode, it displays an inverse plus sign as the cursor. You leave escape mode by typing any key other than a cursor movement key.

❖ *Note:* The escape codes with the arrow keys are the standard cursor movement keys on the Apple IIc. The escape codes with I, J, K, and M are the standard cursor movement keys on the Apple II and II Plus, and are present on the Apple IIc for compatibility.

Escape sequences can be used in the middle of an input line to change the appearance of the screen. They have no effect on the input line.

Editing with GetLn

For an introduction to editing with these features, refer to the *Applesoft Tutorial.*

Subroutine GetLn provides the standard on-screen editing features used by the BASIC interpreters and the Monitor. 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 above.

Backspace

When you press Left Arrow (or Control-H), 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 cursor back one space. If you type another character now, it replaces the character you backspaced over, both on the display and in the line buffer.

Each time you press Left Arrow, it moves the cursor left and deletes another character, until you are back at the beginning of the line. If you then press Left Arrow one more time, you have effectively canceled the line, and GetLn issues a carriage return and displays the prompt. The cursor moves even if the deleted character is an invisible control character. Thus it is possible for screen alignment and buffer alignment to be different.

Retype

See "Escape Codes With GetLn."

Right Arrow (or Control-U) has a function that is complementary to the backspace function. When you press Right Arrow, GetLn picks up the character under the cursor just as if it had been typed on the keyboard. You can use this procedure to pick up characters that you 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.

Standard output features

The standard output routine is named *COut* (character output). COut normally calls COut1 or C3COut1, which sends one character to the display, advances the cursor position, and scrolls the display when necessary. COut1 and C3COut1 restrict their use of the display to an active area called the text window, described later in this chapter.

COut 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 C3COut1, 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 or C3COut1 displays it; if the accumulator contains a control character, COut1 or C3COut1 either performs one of the special functions described below or ignores the character.

Each time you send a character to COut1 or C3COut1, 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 edge of the window, COut1 or C3COut1 moves it to the leftmost position on the next line down. If this would move the cursor position past the end of the last line in the window, COut1 or C3COut1 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 $24 and $25. These locations are named CH, for cursor horizontal, and CV, for cursor vertical. COut1 and C3COut1 do not display a cursor, but the input routines described above do, and they use this cursor position. However, changing CV directly does not change the cursor's vertical position until the next carriage return or reaching the end of the current line causes a call to VTab (for setting the base address within windows). If some other routine displays a cursor, it will not necessarily put it in the cursor position used by COut1 or C3COut1.

Warning When the video firmware is set for 80-column display, the value of CH is kept at 0 and the true horizontal position is stored at $057B. When the 80-column video firmware is active, use $057B instead of CH.

Control characters with COut1

Escape codes are described under "Escape Codes With GetLn."

COut1 does not display control characters. Instead, the control characters listed in Table 3-3 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. The stop-list function, described separately, can only be invoked from the keyboard.

Table 3-3
Control characters with COut1

Control character	ASCII name	Apple IIc 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

Control characters with C3COut1

When the 80-column firmware is active, COut calls C3COut1 instead of COut1 for character output. C3COut1 does not display control characters, but you can use some control characters to control some of what the routine does. All other control characters are ignored.

The control characters listed in Table 3-4 are used to initiate some action by the firmware. Except for the stop-list function (Control-S) you can send control characters to C3COut1 either from a program or from the Apple IIc's keyboard. The stop-list function can only be invoked from the keyboard. Most of the functions listed here can also be performed by using an equivalent escape code.

Table 3-4
Control characters with C3COut1

Control character	ASCII name	Apple IIc name	Action taken by C3COut1
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*
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	Turns off enhanced 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*

Table 3-4 (continued)
Control characters with C3COut1

Control character	ASCII name	Apple IIc name	Action taken by C3COut1
Control-X	CAN	Disable MouseText	Disables MouseText character display; uses 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	Maps inverse uppercase characters to MouseText characters
Control-\	FS	Fwd. 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

* Doesn't work from the keyboard.
† Only works from the keyboard.

The stop-list feature

You can stop the Apple IIc from updating its display (if it is using either COut1 or C3COut1) by pressing Control-S. Whenever COut1 or C3COut1 gets a carriage return from the program, it checks the keyboard for a Control-S. If a Control-S has been pressed, COut1 or C3COut1 stops and waits for another key to be pressed before resuming. The character code of the key that is pressed is ignored unless it is Control-C, which is passed to the program. This feature lets you exit BASIC programs from stop-list mode.

The text window

The active portion of the display is called the *text window*. After you start up the computer or perform a reset, the entire display is the text window. COut1 or C3COut1 puts characters only into the window; when it reaches the end of the last line in the window, it scrolls only the contents of the window.

You can restrict video activity to any rectangular portion of the display by changing the current text window. Your programs can thus control the placement of text in the display and protect other portions of the screen from being written over by new text. To do this, store the appropriate values into four locations in memory to set the top, bottom, left margin, and width of the text window. The following memory locations control the text window:

- The left margin is stored in memory location $20. 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).
- The width of the text window is stored in memory location $21. For a 40-column display, this value is normally 40 (hexadecimal $28); for an 80-column display, it is normally 80 (hexadecimal $50).
- The position of the top line of the text window is stored in memory location $22. This is normally 0, the topmost line in the display. Its maximum value is 23 (hexadecimal $17).
- The position of the bottom line of the screen plus 1 is stored in memory location $23. It is normally 24 (hexadecimal $18) for the bottom line of the display. Its minimum value is 1.

Important Pascal does not use this method of supporting window widths.

Warning Be careful not to let the sum of the window width and the leftmost position in the window exceed the width of the display you are using (40 or 80 columns). If this happens, COut1 or C3COut1 may put characters into memory locations outside the display page, possibly destroying programs or data.

Table 3-5 summarizes the memory locations and the possible values for the text window parameters.

Table 3-5
Text window memory locations

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

Normal, inverse, and flashing text

The way that the Apple IIc displays characters is affected by two things: the value that is stored in the inverse flag (zero page location $32), and whether the enhanced video firmware is off or on. The inverse flag's influence is discussed in the next two subsections.

These display character sets are described in Chapter 5.

If the enhanced video firmware is off, the Apple IIc displays what is called the *primary character set;* if the video firmware is on, the Apple IIc displays what is called the *alternate character set.*

The primary character set includes normal (light on dark), inverse (dark on light), and flashing (alternating normal and inverse) characters. Lowercase inverse characters are not included in the primary character set.

The alternate character set includes normal and inverse characters (including lowercase inverse), and a set of graphic characters called *MouseText.* Flashing characters are not included in the alternate character set.

If you want your program to display a character, it should first load the character to be displayed in the accumulator, and then call the character-output subroutine COut. For example, to display the character corresponding to $C8, you can use something like this:

```
LDA #$C8
JSR COut
```

Primary character set display

The primary character set is displayed by COut1, which operates only when the enhanced video firmware is off. The primary character set includes text in normal, inverse, or flashing format, but not inverse or flashing lowercase text.

For a brief explanation of logical functions, refer to Appendix H.

If the value of the character sent to COut1 is greater than or equal to $A0, that value is logically ANDed with the value of the inverse flag (at location $32), then displayed. (If you're curious about which ASCII character is being sent, subtract $80 from the value being sent to COut1.) You can use the following inverse flag values:

- $FF (decimal 255) produces the normal character format.
- $3F (decimal 63) produces the inverse character format.
- $7F (decimal 127) produces the flashing character format.

Important To avoid unusual character display results, use only the three values $3F, $7F, and $FF.

COut1 interprets character values from $80 through $9F as control characters and tries to execute them.

Character values from $00 through $7F are all interpreted as display characters, not control characters.

Alternate character set display

The alternate character set includes normal and inverse format characters and the MouseText graphic characters. You should use C3COut1, the standard output link when the enhanced video firmware is *active,* to display the alternate character set. Here are the rules for using the alternate character set:

MouseText is described more fully in Chapter 5.

- Control characters are not displayed. Characters sent to C3COut1 are interpreted as control characters if they are in the range $00 through $1F or $80 through $9F.
- Characters in the range $20 through $7F and $A0 through $FF are displayed.
- If inverse flag (location $32) bit 7 is 1, the character is normal.
- If inverse flag bit 7 is 0, the character is inverse.
- If MouseText is off, characters $40 through $5F are remapped to the range $00 through $1F and are displayed as uppercase inverse characters.

See "MouseText" in Chapter 5.

- If MouseText is on, character values $40 through $5F are left unchanged, and the characters are displayed as MouseText.

Port I/O

The Apple IIc is a member of the Apple II family of computers; however, unlike the Apple II, II Plus, and IIe, the Apple IIc does not have peripheral connector slots. In place of these, it has **ports**—the equivalent of firmware interface cards installed in slots.

Standard link entry points

To maintain compatibility with existing software and its protocols, each port's I/O firmware has the same standard entry points ($Cn00) as its equivalent slot in another Apple II would have. Table 3-6 shows these equivalents, as well as listing the chapter where each port is described.

The section on the standard I/O links describes how and when these entry addresses are placed in CSW and KSW. For example, issuing PR#n or IN#n changes the output and input links, respectively, so that subsequent output or input is handled by the firmware starting at address $Cn00, and thus goes to or comes from the selected device.

Memory expansion The memory expansion version of the Apple IIc places the mouse at $C700 and the memory expansion card at $C400.

Table 3-6
Port characteristics

Port	Entry point	Port connector	Use	Chapter
1	$C100	Serial port 1	Printers	7
2	$C200	Serial port 2	Communication	8
3	$C300	Video connectors	Enhanced video firmware	5
4	$C400	Mouse	Mouse	9
5	$C500	Intelligent disk port devices		
6	$C600	Disk drives	Built-in and external drives	6
7	$C700	No device	Reserved	6

Important The addresses shown in Table 3-6 are not entry points in the sense that you can send characters to be printed by sending them to JSR $Cn00.

Firmware protocol

The Apple IIc supports a standard firmware protocol that, in addition to the standard link address, provides a table of device identification and entry points to standard and optional firmware subroutines. The protocol is equivalent to the Pascal 1.1 firmware protocol in use on other Apple II's, and is outlined in Table 3-7.

Table 3-7
Firmware protocol locations

Address	Value	Description
$Cn05	$38	Pascal firmware card/port identifier.
$Cn07	$18	Pascal firmware card/port identifier.
$Cn0B	$01	Generic signature byte of a firmware card/port.
$Cn0C	$ci	Device signature byte: i is an identifier (not necessarily unique). c = device class (not all used on the Apple IIc): $00 reserved $01 printer $02 hand control or other X-Y device $03 serial or parallel I/O card/port $04 modem $05 sound or speech device $06 clock $07 mass-storage device $08 80-column card/port $09 network or bus interface $0A special purpose (none of the above) $0B-0F reserved
$Cn0D	ii	$Cnii is the initialization entry address (PInit).
$Cn0E	rr	$Cnrr is the read routine entry address (PRead) (returns character read in A register).
$Cn0F	ww	$Cnww is the write routine entry address (PWrite) (enters with character to write in A register).
$Cn10	ss	$Cnss is the status routine entry address (PStatus) (enters with request code in A register: 0 to ask "Are you ready to accept output?" or 1 to ask "Do you have input ready?").
$Cn11	$00	If additional address bytes follow; nonzero if not.

Each table begins with identification bytes ($Cn05 through $Cn0C). Then, starting with address $Cn0D, each byte in the table represents the low-order byte of the entry-point address of a firmware routine. The high-order byte of each address is always $Cn, where n is the port number. Your program uses these byte values to construct its own jump table for subroutine calls to the ports.

All port routines require, on entry, that the X register contain $Cn and that the Y register contain $n0.

All routines, on exit, return an error code in the X register (0 means no error occurred; 3 means the request was invalid). The carry bit in the program status register usually contains a reply to a request code (0 means no; 1 means yes).

All the Apple IIc ports except the disk port conform to this protocol. The disk port is described in Chapter 6.

Port I/O space

For more information, refer to the hardware page memory map in Appendix B.

By a convention used in other Apple II series machines, each port or slot has exclusive use of 16 memory locations set aside for data input and output. The addresses of these locations are of the form $C080 + #n0, where n is the port or slot number. Table 3-8 lists the port I/O space used in the Apple IIc.

Table 3-8
Port I/O locations

Port	Locations
1	$C090–$C09F
2	$C0A0–$C0AF
6	$C0E0–$C0EF

Port ROM space

In the Apple II and IIe, one 256-byte page of memory space is allocated to each slot. This space is used for read-only memory (ROM or PROM on the interface card) with driver programs that control the operation of input/output devices, as outlined in Table 3-7. On the Apple IIc, this space is dedicated to port firmware. However, I/O ROM space in the Apple IIc is used as efficiently as possible, and there is not a strict correspondence between firmware for port n and the $Cn00 space, except as regards entry points.

Expansion ROM space

The 2K-byte memory space from $C800 to $CFFF in the Apple IIc—called *expansion ROM space* on the Apple II, II Plus, and IIe—contains the enhanced video firmware and port and memory transfer subroutines. The Apple IIc, unlike the II, II Plus, or IIe, always has this space switched in.

Port screen hole RAM space

There are 128 bytes of memory (64 in main memory, 64 in auxiliary memory) allocated to the ports, eight bytes per port, as shown in Table 3-9. These bytes are reserved for use by the system, except as described in Chapters 4 through 9.

Table 3-9
Port screen hole memory locations

Base address	Ports						
	1	2	3	4	5	6	7
$0478	$0479	$047A	$047B	$047C	$047D	$047E	$047F
$04F8	$04F9	$04FA	$04FB	$04FC	$04FD	$04FE	$04FF
$0578	$0579	$057A	$057B	$057C	$057D	$057E	$057F
$05F8	$05F9	$05FA	$05FB	$05FC	$05FD	$05FE	$05FF
$0678	$0679	$067A	$067B	$067C	$067D	$067E	$067F
$06F8	$06F9	$06FA	$06FB	$06FC	$06FD	$06FE	$06FF
$0778	$0779	$077A	$077B	$077C	$077D	$077E	$077F
$07F8	$07F9	$07FA	$07FB	$07FC	$07FD	$07FE	$07FF

These addresses are unused bytes in the RAM reserved for text and low-resolution graphics displays, and hence they are sometimes called screen holes. These particular locations are not displayed on the screen and their contents are not changed by the built-in output routines. In other words, they are used by the output routines but they are not part of the video display.

Warning All the screen holes in auxiliary memory, and many of them in main memory, are reserved for special use by Apple IIc firmware—for example, to store initialization information. Do not use any locations marked **reserved** in this manual.

The way that port firmware uses these RAM locations and their addresses is covered in Chapters 4 through 10.

Interrupts

Appendix E describes interrupt handling on the Apple IIc.

Interrupts are a way to more efficiently use the hardware in a computer. Interrupt support built into the Apple IIc's firmware is described briefly below.

When the IRQ line on the 65C02 microprocessor is activated, the 65C02 transfers program control through the vector in locations $FFFE through $FFFF of ROM or whichever bank of RAM is switched in (Chapter 2). If ROM is switched in, this vector is the address of the Monitor's interrupt handler, which determines whether the request is due to an interrupt that should be handled internally. If so, the Monitor handles it and then returns control to the interrupted program.

If the interrupt is due to a BRK ($00) instruction, control is transferred through the BRK vector ($03F0–03F1). Otherwise, control is transferred through the IRQ vector ($03FE–$03FF).

Chapter 4

Keyboard and Speaker

This chapter describes how to use two of the Apple IIc's built-in devices: the keyboard and the speaker.

Keyboard input

Table 4-1 describes the characteristics of the keyboard that relate to programming. You won't have to write routines to read the keyboard from all your assembly-language programs since the Apple IIc firmware Monitor provides keyboard support through the three standard input routines described in Chapter 3—RdKey, KeyIn, and GetLn. You *can* do all your keyboard handling directly in your programs if you want to, but it's nice to know that you're not forced to.

Reading the keyboard

For a description of how the keyboard strobe works, refer to Appendix E.

The keyboard encoder and ROM (see Chapter 11) can generate all 128 ASCII codes, so all the special character codes in the ASCII character set are available from the keyboard. Your machine-language programs can call RdKey to get characters from the keyboard. RdKey reads characters a byte at a time from the keyboard data location ($C000) shown in Table 4-1.

Here is how your programs should go about reading the keyboard:

1. Test bit 7 of address $C000 to see if a key has been pressed. Bit 7 is the keyboard strobe bit.
2. When bit 7 goes to a 1, you know that the low-order seven bits of $C000 are a valid character.
3. Clear the keyboard strobe (bit 7) at $C000 by reading or writing *anything* to address $C010.

$C010 has another function besides clearing the keyboard strobe: its high bit is a 1 while a key is pressed (except the Apple keys, Control, Shift, Caps Lock, and Reset). Bit 7 at this location is therefore called *any-key-down*. You could use this to let a program do something useful other than just waiting for the next key to be pressed. (People are generally a *lot* slower than the Apple IIc.) Check $C010 occasionally to see if something should be done.

Important If your program needs to read both the keyboard flag and the strobe, it must read the strobe bit first. Any time you read the any-key-down bit at $C010, you also clear the keyboard strobe bit at $C000.

Table 4-1
Keyboard input characteristics

Port number	None
Commands	Keyboard is always on, in the sense that any keypress generates a KSTRB.
Initial characteristics	Reset routine clears the keyboard strobe and sets the keyboard as the standard input device (that is, sets KSW to point to RdKey).

Hardware locations

$C000	Keyboard data and strobe
$C010	Any-key-down flag and clear-strobe switch
$C060	40-column switch status on bit 7; 1 = 40-column display = switch down
$C061	Open Apple status on bit 7; 1 = pressed (also game input switch 0)
$C062	Solid Apple status on bit 7; 1 = pressed

On game input switches, see Chapter 9.

Monitor firmware routines

Location	Name	Description
$FD6A	GetLn	Gets an input line with prompt
$FD67	GetLnZ	Gets an input line with preceding carriage return
$FD6F	GetLn1	Gets an input line, but with no preceding prompt
$FD1B	KeyIn	The keyboard input subroutine
$FD35	RdChar	Gets an input character or escape code
$FD0C	RdKey	The standard character input subroutine

On GetLn, GetLn1, and RdKey, see Chapter 3.

Use of other pages

Page 2	The standard character string input buffer (see GetLn description)

After your program has cleared the keyboard strobe, the strobe remains low until another key is pressed.

Table 4-2 shows the ASCII codes generated by all the keys on the Apple IIc keyboard. Remember, if the strobe bit is set, the character values that your program sees will be equal to the values given in Table 4-2 plus $80.

Table 4-2
Keys and ASCII codes

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
Delete	7F	DEL	7F	DEL	7F	DEL	7F	DEL
Left Arrow	08	BS	08	BS	08	BS	08	BS
Tab	09	HT	09	HT	09	HT	09	HT
Down Arrow	0A	LF	0A	LF	0A	LF	0A	LF
Up Arrow	0B	VT	0B	VT	0B	VT	0B	VT
Return	0D	CR	0D	CR	0D	CR	0D	CR
Right Arrow	15	NAK	15	NAK	15	NAK	15	NAK
Escape	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	~
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

Table 4-2 (continued)
Keys and ASCII codes

Key	Key alone		+ Control		+ Shift		+ Both	
	Code	Char	Code	Char	Code	Char	Code	Char
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

Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Keystrokes can also generate interrupts. See Appendix E.

There are several keys that do not generate ASCII codes themselves, but alter the characters produced by other keys. These modifier keys are Control, Shift, and Caps Lock.

Your programs can also use the Open Apple and Solid Apple as character modifier keys while handling keyboard input, and, if one or both of them are pressed, branch to a special routine, such as a help program. Your program can read Open Apple at $C061 and Solid Apple at $C062.

The reset routine is described in Chapter 2.

For information on how to have programs interpret keystrokes in a standard way, refer to the *Apple II Design Guidelines* listed in the Bibliography.

Another key that doesn't generate a code is Reset, located at the upper-left corner of the keyboard; it is connected directly to the Apple IIc's processor. Pressing Reset with Control depressed normally causes the system to stop whatever program it's running and restart itself. If you hold Open Apple while pressing Control-Reset, the Apple IIc performs a forced cold start. The restart sequence is described in Chapter 2.

Monitor firmware support for keyboard input

Chapter 3 describes the three standard Monitor input routines serving the keyboard: GetLn, RdKey, and KeyIn. This section discusses the three other available Monitor routines.

GetLnZ

GetLnZ (at address $FD67) is an alternate entry point for GetLn that first sends a carriage return to the standard output, then continues into GetLn.

GetLn1

GetLn1 (at address $FD6F) is an alternate entry point for GetLn that does not issue a prompt before it accepts the input line. However, if the user cancels the input line with too many backspaces or with Control-X, then GetLn1 issues the prompt stored at location $33 when it gets another line.

RdChar

RdChar (at address $FD35) is a subroutine that gets characters from the standard input subroutine, and also interprets the escape codes listed in Chapter 3.

If the enhanced video firmware is active, Right Arrow (Control-U) reads a character from the screen as if it were typed from the keyboard. This is a function of the Monitor's built-in editing capability described in Chapter 3.

Speaker output

Electrical specifications of the speaker circuit appear in Chapter 11.

The Apple IIc has a small speaker mounted near the front of the bottom plate of its case. The speaker is connected to a soft switch that toggles; that is, the switch has two states, off and on, and it changes from one to the other each time it is accessed. Table 4-3 describes the speaker output characteristics.

Table 4-3
Speaker output characteristics

Port number	None.
Commands	Some programs sound the speaker in response to Control-G.
Initial characteristics	Reset routine sounds the speaker.
Hardware location	
$C030	Toggle speaker (read only).

Monitor firmware routines

Location	Name	Description
$FBDD	Bell1	Sends a beep to the speaker.
$FF3A	Bell	Sends Control-G to the current output.

Using the speaker

If you switch the speaker once, by reading or writing to $C030, it emits a click; to make longer sounds, access the speaker repeatedly. The switch for the speaker uses memory location $C030. You can make various tones and buzzes with the speaker by using combinations of timing loops in your program.

Important 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.

Monitor firmware support for speaker output

See Chapter 3.

The Monitor supports the speaker with one simple routine, Bell1. A related routine, Bell, supports the current output device—the one that CSW points to.

Bell1

Bell1 (at address $FDBB) makes a beep through the speaker by generating a 1-kHz tone in the Apple IIc's speaker for 0.1 second. This routine scrambles the A and X registers.

Bell

The Monitor routine Bell (at location $FF3A) writes a bell control character (ASCII Control-G) to the current output device. This routine leaves the accumulator holding $87.

Chapter 5

Video Display Output

NTSC stands for *National Television Standards Committee*, a group that formulates broadcast and reception guidelines used by the USA and several other countries.

The Apple IIc's primary output device is its video display. You can use any ordinary color or monochrome video monitor with the Apple IIc. An ordinary monitor is one that accepts **NTSC**-compatible composite video. If you use Apple IIc color graphics with a black-and-white monitor, the display will appear as black, white, and two shades of gray.

If you are only using graphics modes and 40-column text, 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 IIc; otherwise, you must attach an RF video modulator between the Apple IIc and the television set.

Important The Apple IIc 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.

Table 5-1 summarizes the video output port's characteristics and points to other information in this chapter.

Table 5-1
Video output port characteristics

Port number	Output port 3.
Commands	See Figure 5-3.
Initial characteristics	See Figure 5-3. *Note:* If a program is to use the enhanced video firmware, it should turn it on and then immediately check the 80/40 switch. If the switch is in the 40 position, the program should issue a Control-Q.
Hardware locations	See Table 5-7.
Monitor firmware routines	See Table 5-11.
I/O firmware entry points	See Table 5-12.

Video display specifications

Table 5-2 summarizes the video display's specifications, and provides a further guide to other information in this chapter.

Table 5-2
Video display specifications

Display modes	40-column text; map: Figure 5-5 80-column text; map: Figure 5-6
	Low-resolution color graphics; map: Figure 5-7
	High-resolution color graphics; map: Figure 5-8
	Double high-resolution color graphics; map: Figure 5-9
Text capacity	24 lines by 80 columns (character positions)
Character set	96 ASCII characters (uppercase and lowercase)
Display formats	Normal, inverse, flashing, MouseText (Table 5-3)
Low-resolution graphics	16 colors (Table 5-4): 40 horizontal by 48 vertical; map: Figure 5-7
High-resolution graphics	6 colors (Table 5-5): 140 horizontal by 192 vertical (restricted)
	Black and white: 280 horizontal by 192 vertical; map: Figure 5-8
Double high-resolution graphics	16 colors (Table 5-6): 140 horizontal by 192 vertical (no restrictions)
	Black and white: 560 horizontal by 192 vertical; map: Figure 5-9

The video signal produced by the Apple IIc is NTSC-compatible composite color video available at two places on the back panel of the Apple IIc: the RCA-type phono jack and the 15-pin D-type connector. Use the RCA-type phono jack to connect a video monitor, and the DB-15 connector for an external video modulator or other video expansion hardware.

See "Video Output Signals" in Chapter 11 for more on video expansion hardware.

Text modes

Either of the Apple IIc's two text modes can display all 96 ASCII characters: 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 (with a few exceptions, such as underscore), leaving two blank columns of dots between characters in a row. Except for lowercase letters with descenders, the characters are only seven dots high, leaving one blank line of dots between rows of characters.

The normal display has white (or other monochrome color used by your monitor) dots on a dark background. Characters can also be displayed as black dots on a white background; this is called **inverse video.**

Text character sets

The Apple IIc can display either of two text character sets: the *primary set* and an *alternate set* (Table 5-3). 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

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

The alternate character set trades the flashing format for a complete set of inverse characters. With the alternate character set, the Apple IIc can display uppercase letters, lowercase letters, numbers, and special characters in either normal format or inverse format. It can also display MouseText.

See "MouseText."

To identify particular characters and values, refer to Table 4-2.

You can select between character sets with the alternate-text soft switch, described later in this chapter. Table 5-3 shows the character codes in decimal and hexadecimal for the Apple IIc primary and alternate character sets in normal, inverse, and flashing formats.

Table 5-3
Display character sets

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

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 format and the group within ASCII.

MouseText

The alternate character set contains 32 graphics characters called *MouseText* in place of the primary set's inverse uppercase characters from $40 through $5F. These graphics are especially convenient to use with a mouse because they can be generated by character codes instead of groups of high-resolution byte values, and they can be moved around quickly. To use MouseText characters, do the following:

1. Turn on the enhanced video firmware with PR#3 or 6 Control-P.
2. Set inverse mode: use the INVERSE command or put $3F in location $32, or print Control-O.
3. Turn on MouseText with PRINT CHR$(27); or pass $1B to COut in the accumulator.
4. Print the uppercase letter (or other ASCII character in the range $40 through $5F:@[\] ^ or _) that corresponds to the MouseText character you want.
5. Turn off MouseText with PRINT CHR$(24); or pass $18 to COut1 in the accumulator.
6. Set normal mode: use the NORMAL command or put $FF in location $32, or print a Control-N.

Here is a sample Applesoft program that prints all the MouseText characters:

```
10 D$=CHR$(4)
20 PRINT PRINT D$;"PR#3"
30 INVERSE
40 PRINT CHR$(27);"ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_";
50 PRINT CHR$(24);
60 NORMAL
```

MouseText characters and their corresponding ASCII characters are shown in Figure 5-1.

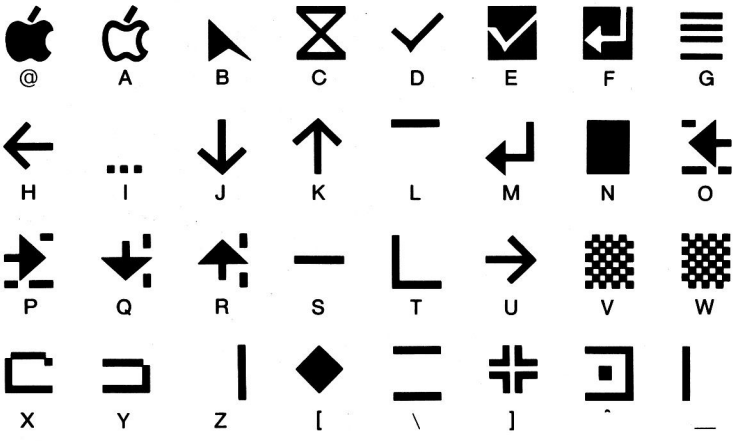


Figure 5-1
MouseText characters

40-column versus 80-column text

The Apple IIc has two text display modes: 40-column and 80-column. The number of dots in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare the two displays in Figure 5-2. On an ordinary color or black-and-white television set, the narrow characters in the 80-column display blur together; you must use the 40-column mode to display text on a television set.

```
]LIST 0,100

10  REM  APPLESOFT CHARACTER DEMO

20  TEXT : HOME
30  PRINT : PRINT "Applesoft Char
     acter Demo"
40  PRINT : PRINT "Which characte
     r 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 th
     e same line, first"
100  PRINT " in NORMAL, then INVE
      RSE ,then FLASH:": PRINT
```

```
]LIST

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

Figure 5-2
40-column and 80-column text with alternate character set

Figure 5-3 shows the characteristics of the text display modes and how to switch between them.

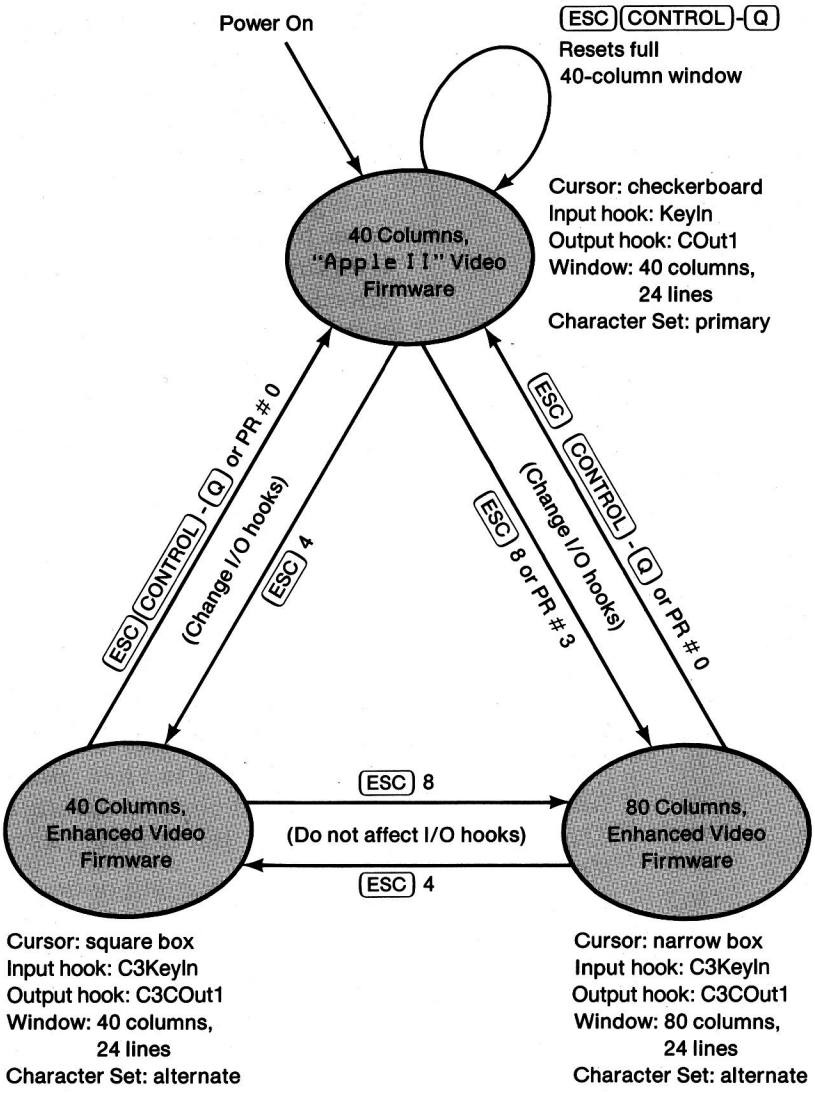


Figure 5-3
Text mode characteristics and switching

Graphics modes

The Apple IIc can produce color video graphics in any of three different modes:

- low-resolution graphics, 48 rows by 40 columns
- high-resolution graphics, 192 rows by 280 columns
- double high-resolution graphics, 192 rows by 560 columns

Each graphics mode treats 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 on the screen; this section describes the way the resulting graphics data are stored in the Apple IIc's memory.

Low-resolution graphics

The Apple IIc displays an array of 48 rows by 40 columns of colored blocks in the low-resolution graphics mode. 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 two shades of gray. There are no blank dots between blocks; adjacent blocks of the same color merge to make a larger shape.

The low-resolution graphics display data are 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 5-4. 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 5-4
Low-resolution graphics colors

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

Note: colors may vary, depending on adjustment of monitor or television set.

As explained earlier in this chapter, the text display and the low-resolution graphics display use the same area in memory. Your programs should usually clear this part of memory when they change display modes, but you can store data as text and display them as graphics, or vice versa. All you have to do is change the mode switch, described later in this chapter, 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 IIc 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 below, by the color of adjacent dots. Adjacent dots of the same color merge to form a continuous colored area.

High-resolution graphics display data 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 display data buffers. 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 IIc's memory.

The Apple IIc high-resolution graphics display is bit-mapped: each dot on the screen corresponds to a bit in the Apple IIc's memory. The seven low-order bits of each display byte control a row of seven adjacent dots on the screen, and 40 adjacent bytes in memory control a row of 280 (7 times 40) dots. The eighth bit (the most significant) of each byte is not displayed; it selects one of two color sets, as described below. The least significant bit of each byte is displayed as the leftmost dot in a row of seven, followed by the next-least significant bit, and so on, as shown in Figure 5-4.

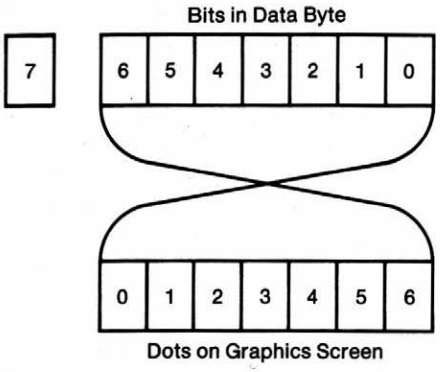


Figure 5-4
High-resolution display bits

There is a simple correspondence between bits in memory and dots on the screen on a black-and-white monitor. 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 merge together; alternating black and white dots merge to a continuous gray.

A dot whose controlling bit is off (0) is black on an NTSC color monitor or a color television set. If the bit is on, the dot is white or a color, depending on its position, the dots on either side, and the setting of the high-order bit of the byte. Call the leftmost column of dots column 0, and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control them are on, dots in even-numbered columns, 0, 2, 4, and so forth, are purple, and dots in odd-numbered columns are green—but only if the dots on either side 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 either side 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 brief, 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-numbered columns can be black, purple, or blue.
- Dots in odd-numbered columns can be black, green, or orange.
- If adjacent dots in a row are both on, they are both white.
- The colors in each row of seven dots controlled by a single byte are either purple and green, or blue and orange, depending on whether the high-order bit is off (0) or on (1).

These rules are summarized in Table 5-5. The blacks and whites are numbered to remind you that the high-order bit is different.

Table 5-5
High-resolution graphics colors

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

Note: Colors may vary, depending on adjustment of monitor or television set.

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

For more details about the way the Apple IIc produces color on a TV set, see Chapter 11. For a table of reversed bit patterns, refer to Appendix H.

Double high-resolution graphics

The horizontal resolution of double high-resolution graphics is 560 dots per line, with 192 lines. Double high-resolution graphics maps the low-order seven bits of the bytes in the two double high-resolution graphics pages. A double high-resolution page is made up of a 8192-byte page in main memory and an equivalent page having the same address in auxiliary memory. In most cases, only the first double high-resolution graphics page is used.

The bytes in the main-memory and auxiliary-memory pages are displayed 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. A dot whose controlling bit is off (0) is black when displayed.

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 four-dot-wide window moving across the screen: at any given time, the color displayed corresponds to the 4-bit value from Table 5-6 that corresponds to the window's position (Figure 5-9). Effective horizontal resolution with color is 140 (560 divided by 4).

Table 5-6 describes the data values used to produce colors in double high-resolution graphics. To use the table, divide the column number by four and use the remainder to find the correct column: *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, 2, 9 and so on), and similarly for *ab2* and *mb3*.

Mixed-mode displays

Any of the graphics displays can have four lines of text, either 40-column or 80-column, at the bottom of the screen. Graphics displays with text at the bottom are called *mixed-mode displays*. To use them, the TEXT switch must be off (read $C050) and the MIXED switch on (read $C053).

Important You cannot display 40-column text with double high-resolution graphics.

To determine what appears where in mixed-mode displays, refer to Figures 5-5 through 5-9 later in this chapter. See the bottom sixth of the appropriate text display (Figure 5-5 or 5-6) and the upper five-sixths (down to the heavy horizontal line) in the appropriate graphics display (Figures 5-7 to 5-9).

Table 5-6
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

Note: Colors may vary, depending on adjustment of monitor or television set.

Display pages

The Apple IIc uses data stored in specific areas in memory to generate its video displays. 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—a character, a colored block, or a group of adjacent dots—at a certain location on the display, depending on the current display mode.

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 $0400 through $07FF and $0800 through $0BFF in main memory. Normally, only Page 1 is used, but you can put text or graphics data into Page 2 and switch between displays. Either page can be displayed as 40-column text, low-resolution graphics, or mixed-mode (four lines 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 when the enhanced video firmware is active. The 80-column text display uses a combination page made up of text Page 1 in main memory plus another page in auxiliary memory. This additional memory is *not* the same as text Page 2—in fact, it is text Page 1X, and it occupies the same address space as text Page 1 (see Figure 2-11). The built-in firmware I/O routines described in Chapter 3 take care of this extra addressing automatically; that is one reason to use these routines for all normal text output.

Important The built-in video firmware always displays Page 1 text. You cannot write text to Page 2 with the built-in firmware.

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 a low-resolution display, as shown in Table 5-7.

The double high-resolution graphics mode interleaves the two high-resolution pages (Pages 1 and 1X) in exactly the same way as 80-column text mode interleaves the text pages: column 0 and all subsequent even-numbered columns come from the auxiliary page; column 1 and all subsequent odd-numbered columns come from the main page.

Table 5-7
Video display page locations

Display mode	Display page	Lowest address	Highest address
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 6384	$5FFF 24575

* This is not supported by firmware; for instructions on how to switch pages, refer to "Display Mode Switching."
† See "Double High-Resolution Graphics."

Display mode switching

Table 5-8 shows the reserved locations for the soft switches that control the different display modes. The column of the table labeled *Action* indicates what to do to activate or read a switch setting: *R* means read the location, *W* means write anything to the location, *R/W* means read or write, and *R7* means read the location and then check bit 7.

Table 5-9 lists the display modes that the firmware can set up automatically. In the 40-column modes, the contents of the standard I/O hooks KSW and CSW (Chapter 3) determine whether the enhanced video firmware features are available or not. The firmware also takes care of setting or clearing AltChar.

Table 5-10 lists other display modes available but not supported by firmware. For modes that display Page 2 with the 80Col switch on, your program may have to turn 80Store off after the firmware has turned it on.

Double low-resolution shows on the display screen when HiRes is off and both 80Col and DHiRes are on. It is the low-resolution graphics equivalent of 80-column text, and it uses the same map (Figure 5-6), giving you 48 rows of 80 blocks.

The IOUDis ($C07E) switch must be on to allow you to use locations $C05E and $C05F to change DHiRes. The firmware in fact leaves it on—and your program should, too—unless it wants to use locations $C05E and $C05F to change mouse values (Chapter 9).

Table 5-8
Display soft switches

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 1X (80Store on) or 2
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	$C053	Off: Display only text or only graphics

Table 5-8 (continued)
Display soft switches

Name	Action	Hex	Function
MIXED	R/W	$C054	On: (If TEXT off) display text and graphics
RdMIXED	R7	$C01B	Read MIXED switch (1 = on)
HiRes	R/W	$C057	Off: (If TEXT off) display low-resolution graphics
HiRes	R/W	$C058	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*
RdIOUDis	R7	$C07E	Read IOUDis switch (1 = off)†
DHiRes	R/W	$C05E	On: (If IOUDis on) turn on double high-resolution
DHiRes	R/W	$C05F	Off: (If IOUDis on) turn off double high-resolution
RdDHiRes	R7	$C07F	Read DHiRes switch (1 = on)†

* The firmware normally leaves IOUDis on. See also the following footnote.

† Reading or writing any address in the range $C070–$C07F also triggers the paddle timer and resets VBLInt (Chapter 9).

Table 5-9
Display modes supported by firmware, including Applesoft

Display			Switches						
col/res	Type	Page	80Col	80Store	Page2	TEXT	MIXED	HiRes	DHiRes
40-column	Text	1	Off		Off	On	Off	Off	Off
80-column	Text	1	On	*		On			
Low-res	Graphics	1	Off		Off	Off	Off	Off	Off
40/low	Mixed	1	Off		Off	Off	On	Off	
80/low	Mixed	1	On	*	Off	Off	On	Off	Off
Hi-res	Graphics	1	Off		Off	Off	Off	On	
Hi-res	Graphics	2	Off		On	Off	Off	On	
40/high	Mixed	1	Off		Off	Off	On	On	
80/high	Mixed	1	On	*	Off	Off	On	On	Off

* 80Store is set by the firmware when 80Col is turned on.

Table 5-10
Other display modes

Display			Switches						
col/res	Type	Page	80Col	80Store	Page2	TEXT	MIXED	HiRes	DHiRes
40-column	Text	2	Off		On	On			
80-column		2	On	Off	On	On			
Low-res	Graphics	2	Off		On	Off	Off	Off	
40/low	Mixed	2	Off		On	Off	On	Off	
80/low	Mixed	2	On	Off	On	Off	On	Off	Off
Dbl-low	Graphics	1	On	*	Off	Off	Off	Off	On
Dbl-low	Graphics	2	On	Off	On	Off	Off	Off	On
80/dbl-low	Mixed	1	On	*	Off	Off	On	Off	On
80/dbl-low	Mixed	2	On	Off	On	Off	On	Off	On
40/high	Mixed	2	Off		On	Off	On	On	
80/high	Mixed	2	On	Off	On	Off	On	On	Off
Dbl-high	Graphics	1	On	*	Off	Off	Off	On	On
Dbl-high	Graphics	2	On	Off	On	Off	Off	On	On
80/dbl-high	Mixed	1	On	*	Off	Off	On	On	On
80/dbl-high	Mixed	2	On	Off	On	Off	On	On	On

* 80Store is set by the firmware when 80Col is turned on, and must be turned off to use the second 80-column or double high-resolution page. This means that you cannot use firmware routines such as COut when displaying Page 2 modes not supported by firmware.

For example, to switch to mixed 80-column and double high-resolution display Page 1, you can use these instructions in your program:

```
STA     $C00D   Turns on 80Col; firmware then turns on 80Store.
LDA     $C054   Turns off Page2; you could also have done a STA.
STA     $C050   Turns off TEXT; that is, turns on graphics mode.
STA     $C053   Turns on MIXED; it works now that TEXT is off.
STA     $C057   Turns on HiRes; it works now that TEXT is off.
STA     $C07E   Makes sure IOUDis is on so you can access DHiRes.
LDA     $C05E   Turns on DHiRes; it works now that IOUDis is on.
```

Display page maps

You should never have to store directly into display memory. Most high-level languages let you write statements that control the text and graphics displays. Similarly, if you are programming in assembly language, you should use the display features of the built-in I/O firmware.

Warning Never call any firmware with 80Col on or with 80Store and Page2 both on. If you do, the firmware will not function properly. As a general rule, always leave Page2 off.

All the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes ($80 hex). For example, the first 128-byte block contains the data for rows 0, 8, and 16. The next 128-byte block contains data for rows 1, 9, and 17, and so on.

The display memory maps are shown in Figures 5-5 through 5-9. For a full description of the way the Apple IIc hardware handles display memory, see Chapter 11.

High-resolution graphics data are 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 first 1024 bytes of the high-resolution display page contain the first row of dots from *each* of the 24 groups of eight rows of dots. The second 1024 bytes of the high-resolution display page contain the second row of dots from *each* group of eight rows of dots, and so on for all eight rows of all the groups. This fills up the 8192 bytes of the high-resolution display page.

The display maps show addresses only for each Page 1. To obtain addresses for text or low-resolution graphics Page 2, add 1024 ($0400); to obtain addresses for high-resolution Page 2, add 8192 ($2000).

The 80-column display works a little differently. Half of the data are stored in the normal text Page 1 memory, and the other half are stored in the *auxiliary* memory text Page 1. The display circuitry fetches bytes from the same address in both memory areas simultaneously and displays them sequentially: first the byte from the auxiliary memory, then the byte from the main memory. The characters in the even-numbered columns of the display are stored (starting with column 0) in main memory, and the characters in the odd-numbered columns of the display are stored (starting with column 1) in main memory.

For more details about the way the displays are generated, see Chapter 11.

To store display data in auxiliary memory, first turn on the 80Store soft switch by writing to location $C001. 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 auxiliary memory. To select auxiliary memory, turn the Page2 soft switch on by reading or writing at location $C055.

The double high-resolution graphics display stores information in the same way as high-resolution graphics, except there is an auxiliary memory location as well as a main memory location corresponding to each address. The two sets of display information are interleaved in a manner similar to the interleaving of two 40-column displays to create an 80-column text display (Figure 5-9).

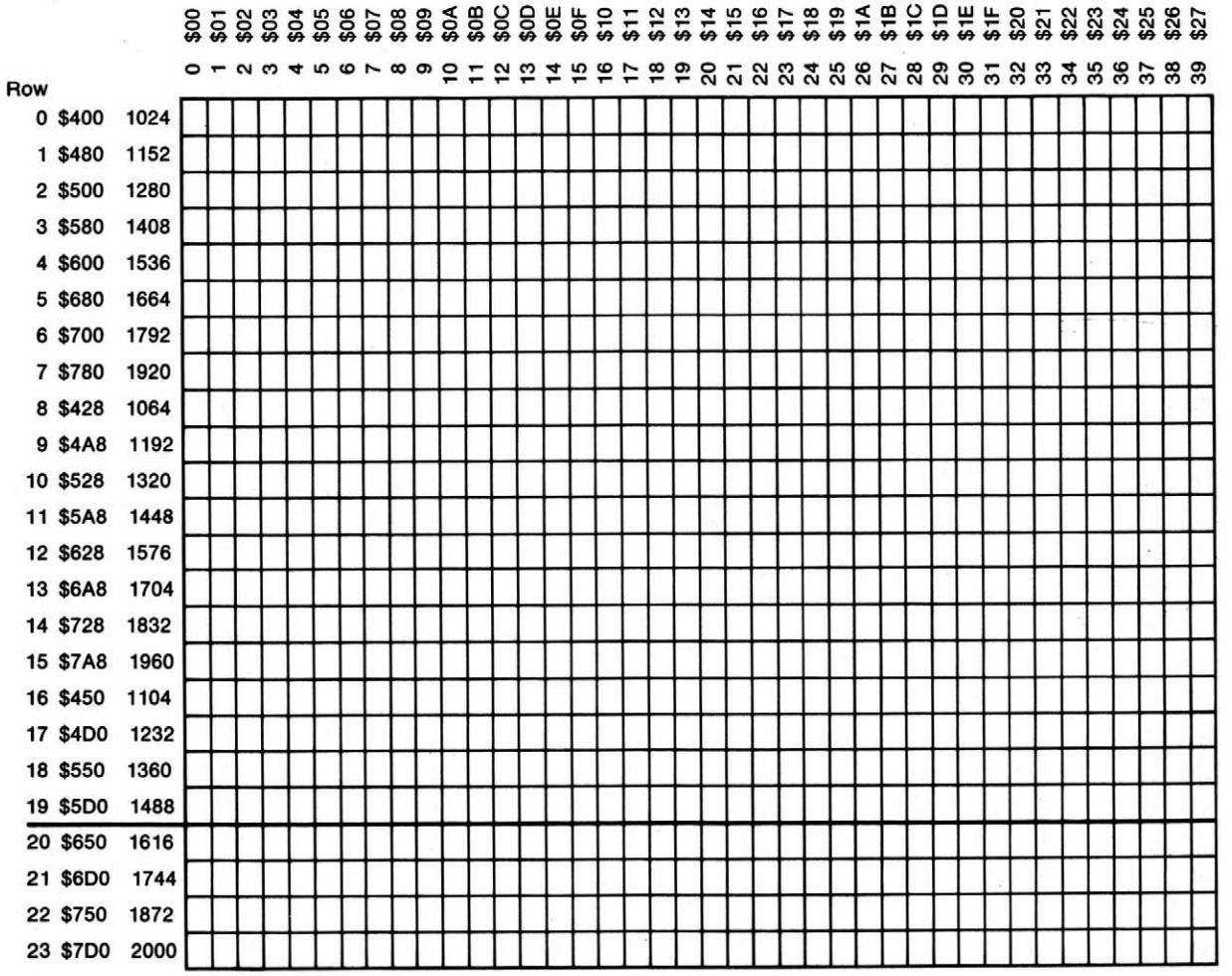


Figure 5-5
Map of 40-column text display

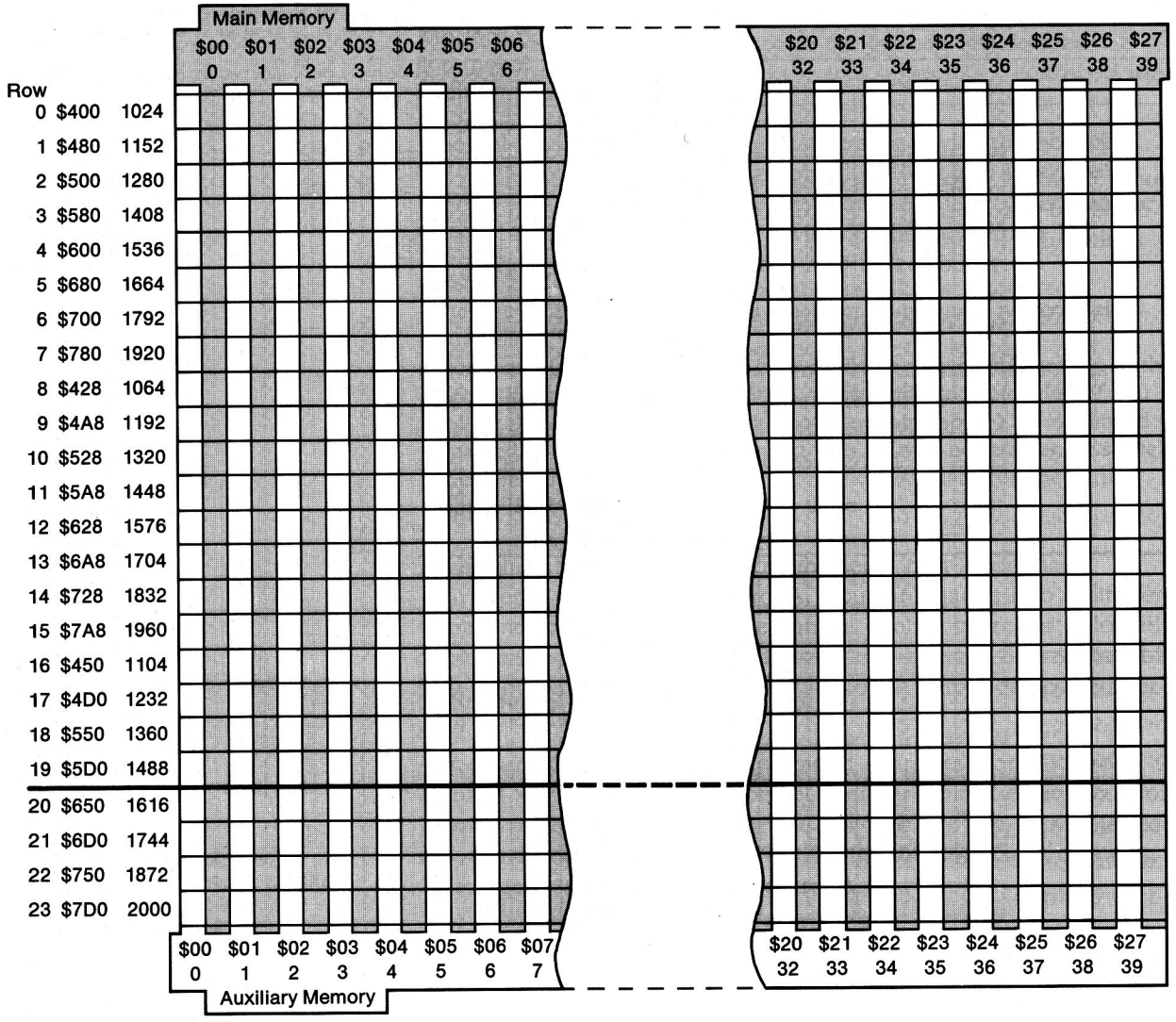


Figure 5-6
Map of 80-column text display

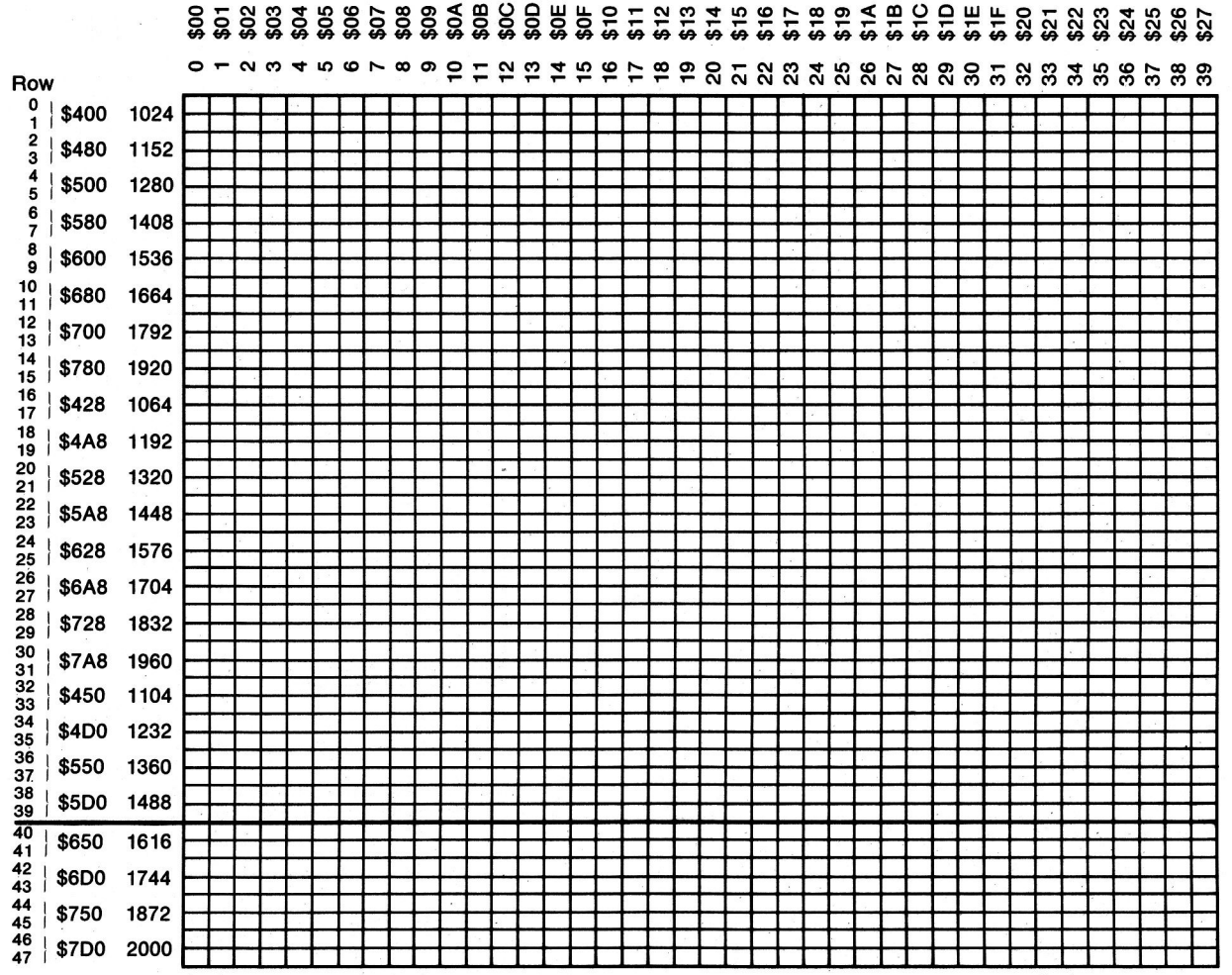


Figure 5-7
Map of low-resolution graphics display

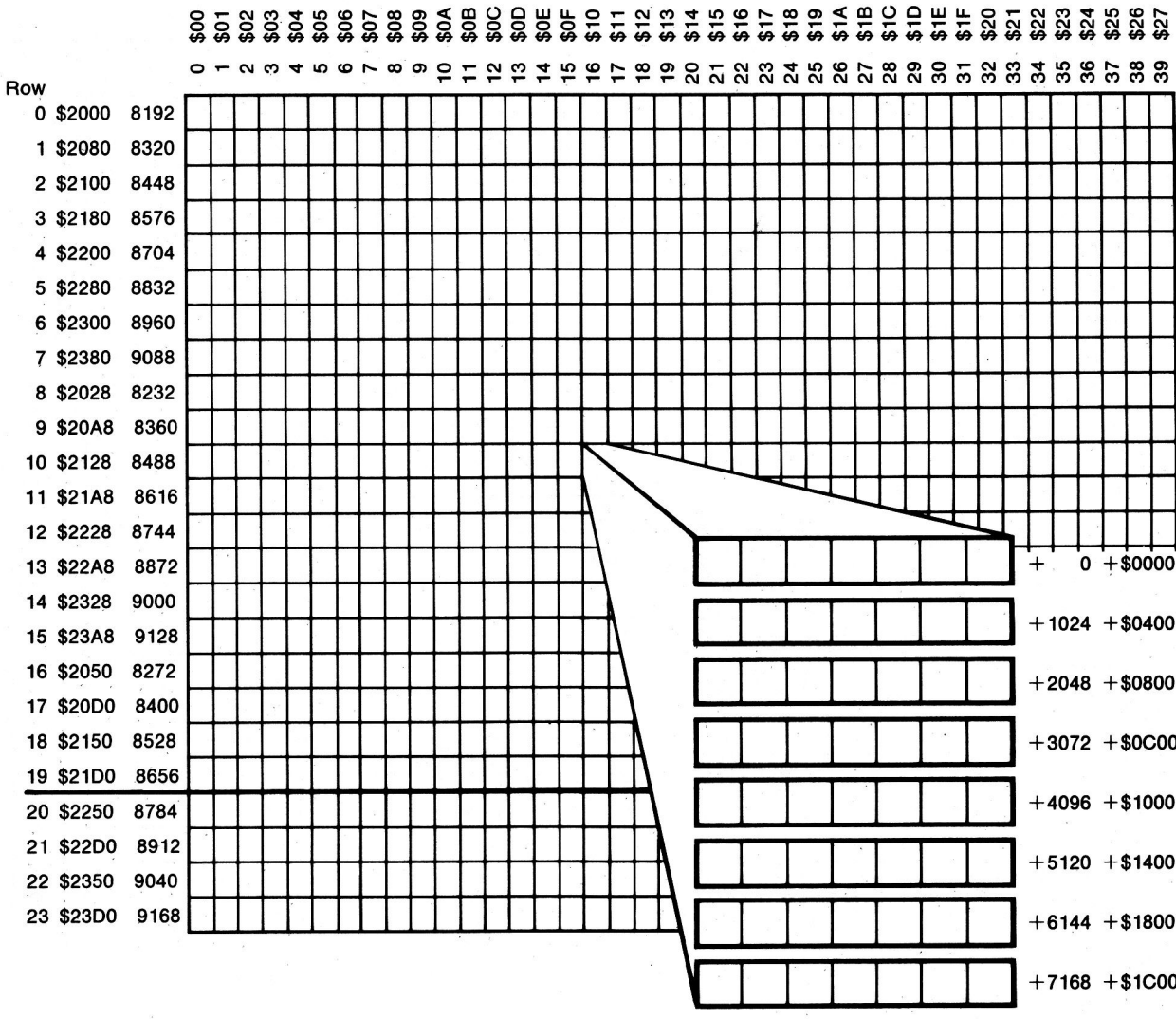


Figure 5-8
Map of high-resolution graphics display

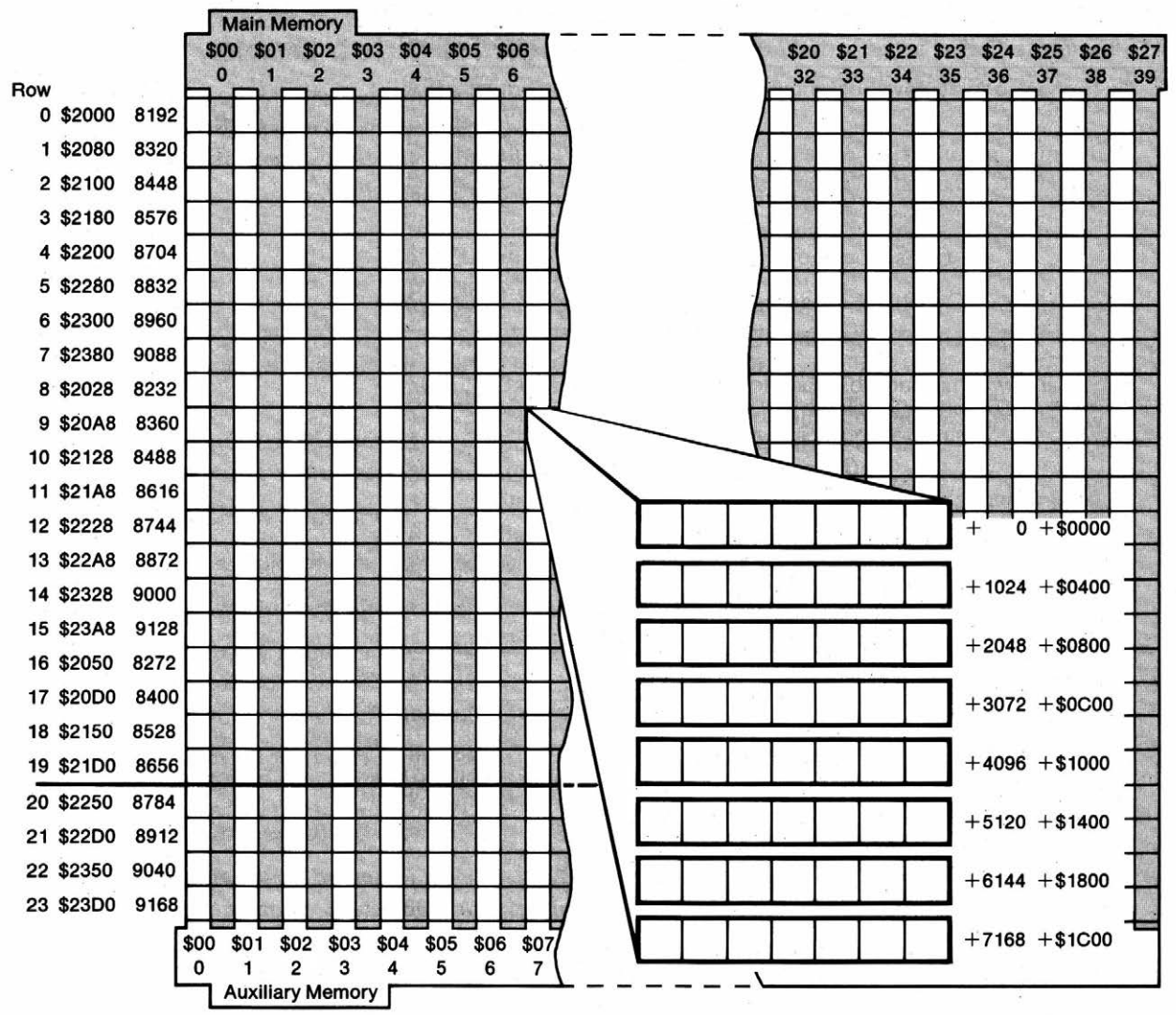


Figure 5-9
Map of double high-resolution graphics display

Monitor support for video display output

Table 5-11 summarizes the addresses and functions of the video display support routines the Monitor provides. Except for COut and COut1, which are explained in Chapter 3, these routines are described in the subsections that follow.

Table 5-11
Monitor firmware routines

Name	Location	Description
ClrEOL	$FC9C	Clears to end of line from current cursor position
ClEOLZ	$FC9E	Clears to end of line using contents of Y register as cursor position
ClrEOP	$FC42	Clears to bottom of window
ClrScr	F832	Clears the low-resolution screen
ClrTop	$F836	Clears top 40 lines of low-resolution screen
COut	$FDED	Calls output routine whose address is stored in CSW (normally COut1, Chapter 3)
COut1	$FDF0	Displays a character on the screen (Chapter 3)
CROut	$FD8E	Generates a carriage return character
CROut1	$FD8B	Clears to end of line, then generates a carriage return character
HLine	$F819	Draws a horizontal line of blocks
HOME	$FC58	Clears the window and puts cursor in upper-left corner of window
PLOT	$F800	Plots a single low-resolution block on the screen
PrBl2	$F94A	Sends 1 to 256 blank spaces to the output device whose address is in CSW
PrByte	$FDDA	Prints a hexadecimal byte
PrErr	$FF2D	Sends ERR and Control-G to the output device whose output routine address is in CSW
PrHex	$FDE3	Prints four bits as a hexadecimal number

Table 5-11 (continued)
Monitor firmware routines

Name	Location	Description
PrntAX	$F941	Prints contents of A and X in hexadecimal
SCRN	$F871	Reads color value of a low resolution block on the screen
SetCol	$F864	Sets the color for plotting in low resolution
VTabZ	$FC24	Sets cursor vertical position (setting CV at location $25 does not change vertical position until a carriage return)
VLine	$F828	Draws a vertical line of low-resolution blocks

ClrEOL

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

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

ClrEOP

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

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

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

COut

COut calls the current character output subroutine. 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), usually the standard character output COut1.

COut1

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

CROut sends a carriage return to the current output device.

CROut1

CROut1 clears the screen from the current cursor position to the edge of the text window, then calls CROut.

HLine

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

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

PLOT

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

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

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

PrErr

PrErr sends the word ERR, followed by a bell character (ASCII $07), to the standard output device. On return, the accumulator is scrambled.

PrHex

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

PrntAX 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

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

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 5-4.

VLine

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 for video display output

Apple IIc video firmware conforms to the I/O firmware protocol described in Chapter 3. 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 (port 3) protocol table is shown in Table 5-12.

Table 5-12
Port 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

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

PRead reads a character from the keyboard and places it in the accumulator with the high bit cleared. It also puts a 0 in the X register to indicate IOResult = GOOD.

PWrite

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
- carries out control functions as shown in Table 5-13

Table 5-13
Pascal video control functions

Control-	Hex	Function
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 position on 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
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

Table 5-13 (continued)
Pascal video control functions

Control-	Hex	Function
] 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

When PWrite has completed this, it

- turns the cursor back on (if it was not intentionally turned off)
- puts a 0 in the X register (IOResult = GOOD) and returns to the calling program

PStatus

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). If the request was not 0 or 1, PStatus returns with a 3 in the X register (IOResult = ILLEGAL OPERATION); otherwise, PStatus returns with a 0 in the X register (IOResult = GOOD).

Chapter 6

Block Device I/O

A **block-type device,** or block device, executes I/O operations by grouping data into bundles, called *blocks.* A block may be made up of virtually any number of bytes, but in the Apple IIc a standard block is 512 bytes.

The Apple IIc supports both built-in and external **block-type devices.** External block devices may be 5.25-inch Disk IIc drives, UniDisk 3.5-inch disk drives, a memory expansion card, and other similar devices. If you use a 5.25-inch Disk IIc as an external drive, you must install it as the last device in the daisy chain.

Original IIc

The original Apple IIc does not support devices other than its internal 5.25-inch disk drive and an (optional) external 5.25-inch Disk IIc drive.

The external block device interface is provided by the Smartport firmware. The Smartport is described later in this chapter.

UniDisk 3.5

The UniDisk 3.5 ROM contains an older version of the Smartport, the Protocol Converter. The description of the Smartport applies to the Protocol Converter, and vice versa.

Disk drive I/O

The external disk drive connector is described under "Disk I/O" in Chapter 11.

Disk I/O firmware for the 5.25-inch drives resides in the $C600 address space on the main side of the ROM. The built-in 5.25-inch drive is supported as if it were slot 6, drive 1, and the external 5.25-inch drive as if it were slot 6, drive 2.

Disk I/O firmware for the UniDisk 3.5 drive resides in the $C500–$C58D address space on the main side, and in the $C880–$CFFF address space on the auxiliary side of the ROM.

Table 6-1 summarizes the disk I/O port characteristics.

Table 6-1
Disk I/O port characteristics

Port number	I/O port 6 drive 1 (built-in 5.25-inch drive). I/O port 6 drive 2 (external 5.25-inch drive). I/O port 5 drive 1 (external 3.5-inch drive).
Commands	IN#6 or PR#6 CALL –151 (to get to the Monitor from BASIC), then 6 Control-K or 6 Control-P.
Initial characteristics	All resets except Control-Reset with a valid reset vector eventually pass control to the built-in disk drive.

Table 6-1 (continued)
Disk I/O port characteristics

Hardware location	
$C0E0–EF	Reserved.
Monitor firmware routines	None.
I/O firmware entry points	$C600 (port 6).
Use of screen holes	Port 6 main and auxiliary memory screen holes are reserved.

Startup

The Apple IIc has two ways to start up—a cold start and a warm start. A cold start clears the machine's memory and tries to load an operating system from disk. A warm start halts the program that is running and leaves the machine in Applesoft with the contents of memory intact.

Cold start

A cold start can be initiated by any of the following:

- turning the machine on
- pressing Open Apple-Control-Reset
- issuing a reboot command from the Monitor, BASIC, or a program
- pressing Control-Reset, if a valid reset vector does not exist

The startup routine first sets a number of soft switches to their initialization settings (see Chapter 2) and then passes control to the memory expansion card I/O entry point at $C400. Because the contents of the memory expansion card's RAM are invalid in all cold-start situations, the Apple IIc cannot boot from card and control is returned to the startup routine.

Original IIc The original Apple IIc does not support the memory expansion card; the restart routine in the original IIc begins with the internal 5.25-inch drive.

When control is returned to the startup routine by the memory expansion card, it will attempt to boot the Apple IIc from the internal 5.25-inch drive. Control is passed to the 5.25-inch disk I/O entry point at $C600. The code at this address turns on the internal drive motor, recalibrates the read/write head at track 0, then reads sector 0 from that track. The sector contents are loaded into main memory, starting at address $0800. Once the contents of sector 0 have been loaded into main memory, control passes to $0801. The program loaded depends on the operating system or application program on the disk in internal drive.

If for any reason the Apple IIc is unable to boot from the internal drive, control is returned to the startup routine. The startup routine then attempts to boot the Apple IIc from the external UniDisk 3.5 drive. Control is passed to the UniDisk 3.5 I/O entry point at $C500, and the startup attempt proceeds in the same manner as that of the internal 5.25-inch drive.

Original IIc

The original Apple IIc does not support the UniDisk 3.5 drive. However, it is possible to start the original Apple IIc from the external 5.25-inch drive. If you want to start your Apple IIc from the external 5.25-inch drive, you must use the ProDOS operating system. To start from the external drive, insert a ProDOS disk in the drive and

- From the Monitor, type `CALL -151` and press 7 Control-P.
- From BASIC, type `PR#7`.

To force a cold restart of the system:

- From BASIC, issue a PR#6 command.
- From the Monitor, issue 6 Control-P.
- From a machine-language program, JMP $C600.

Memory expansion

To force a cold restart from a machine-language program in an Apple IIc that supports the memory expansion card, JMP $C400 (the memory expansion card entry point).

UniDisk 3.5

The Apple IIc that supports the UniDisk 3.5 can force a cold restart that skips the internal 5.25-inch drive and passes control to the external drive port at $C500 entry point. This allows the system to start up from the first *intelligent* drive connected to the external drive port. You can use the ProDOS or Pascal operating system if you want to start the system from an external drive, but DOS and versions of Pascal earlier than 1.3 will not work.

Warm start

A warm start is initiated by pressing Control-Reset. The warm start routine checks $F800–$FFFF on the main side ROM for a valid reset vector. Provided a valid reset vector exists, control is turned over to the entry point specified by the vector. Generally, a warm start leaves you in BASIC with memory unchanged.

If there is no valid reset vector, a number of things may happen:

- The Apple IIc passes control to $C600 on the main side ROM and the cold-start boot procedure begins.
- The Apple IIc beeps.
- The Apple IIc does nothing.

Memory expansion

In the Apple IIc that supports the memory expansion card, control is turned over to $C400 on the main side ROM in the event there is no valid reset vector.

Memory expansion card I/O

The memory expansion card provides up to 1Mb of RAM, in 256K steps, for storage of program and data files. In this sense, it is like a very fast disk drive. Programs can be loaded into the memory expansion card's RAM, but in order to be executed they must be moved, in whole or in part, to the Apple IIc's main memory.

The memory expansion card is a block-type device, so I/O operations involving the card use the operating system or Smartport I/O interface. The Smartport I/O interface is described later in this chapter.

More information on the memory expansion card can be found in the *Apple IIc Memory Expansion Card Technical Reference.*

The Smartport I/O interface

Important

The rest of this chapter applies only to the UniDisk 3.5 and memory expansion versions of the Apple IIc.

UniDisk 3.5 The Smartport and the Protocol Converter are essentially the same firmware interface with different names. All the specifications given in this manual for the Smartport interface apply to the Protocol Converter as well.

The rest of this chapter is about the Smartport, which is a set of assembly-language routines used to support external I/O devices, such as UniDisk 3.5. To ProDOS and Pascal 1.3, the Smartport appears to be a block device.

At the end of this chapter is an example of an assembly-language program that uses a Smartport call.

Locating the Smartport

The Smartport code in the Apple IIc's firmware always begins at address $C500. To ensure compatibility of your programs with the Apple IIe, however, your Smartport routines should always begin with a search for the Smartport. Your program can identify the Smartport by finding the following bytes:

$Cn01=$20
$Cn03=$00
$Cn05=$03
$Cn07=$00

where n can be an integer from 1 to 7. The Smartport entry point is then found at address $Cn00 + ($CnFF) + 3, where ($CnFF) refers to the value of the byte located at $CnFF. The sample program at the end of this chapter illustrates such a search.

Important The Smartport firmware is present even when the Memory Expansion Card is not. To check for the Memory Expansion Card, issue a STATUS call, code $03, from the operating system or the Smartport. If the data returned indicates 0 bytes available, the card is not present.

Issuing a call to the Smartport

On **MLI** calls, see the *ProDOS Technical Reference Manual*, Chapter 4.

Smartport calls are coded like ProDOS Machine Language Interface (**MLI**) calls: the program executes a JSR to a dispatch routine at address $C500 + ($C5FF) + 3, where ($C5FF) refers to the value of the byte located at $C5FF.

The Smartport call number and a two-byte pointer to the call's parameter list must immediately follow the call. Here is an example of a call to the Smartport:

```
IWMCALL
JSR DISPATCH     Calls PC command dispatcher
DFB CmdNum       Specifies the command type
DW  CmdList      2-byte (low, high) pointer to parameter list
BCS ERROR        Sets carry on an error
```

The command number (CmdNum) defines which Smartport call you want to make. Most Smartport calls include a two-byte pointer to a parameter list. The parameter list can contain information to be used by the call, or can provide space for information to be returned by the call. The length and content of the parameter list depend on the call being made. The format of each Smartport call's parameter list is described later in this chapter.

When the call has finished, the program resumes execution at the statement following the pointer to the parameter list. In the example above, the DFB and DW statements are skipped and execution resumes with the BCS statement. If the call is successful, the C flag (in the processor status register) is cleared (0), and the accumulator (the A register) is cleared to all 0's. If the call is unsuccessful, the C flag is set (1) and the error code is placed in the A register. After the Smartport call, the contents of the 65C02's registers are as follows:

Register	Processor status								X	Y	A	PC	S
	N	V	1	B	D	I	Z	C					
Successful call	x	x	1	u	0	u	x	0	x	x	0	JSR+3	u
Unsuccessful call	x	x	1	u	0	u	x	1	x	x	Error	JSR+3	u

x = undefined, except in cases where index information is returned in X and Y registers
u = unchanged

Cautions

You must observe the following cautions when using the Smartport, *or your program will crash:*

- Leave space on the stack for the Smartport. The Smartport requires up to 35 bytes of stack space. Be sure to take this into account when calculating the stack space used by your program. If you don't do this, your program will fail if it tries to access data that *used* to be on the stack.
- Be sure that all RAM that you intend the Smartport to access is both read-enabled and write-enabled. The Smartport must be able to read from the RAM after writing to it, to obtain a checksum. Failure to observe this rule results in an error (BusErr $06).
- Don't pass data to or from the Smartport through any zero page locations. Some of these locations are reserved for temporary storage of data by the Smartport, and your data will get changed.

On reading and writing to RAM, see "Bank-Switched Memory" in Chapter 4.

Descriptions of the Smartport calls

Calls to the Smartport are used

- to obtain status information about a device
- to reset a device
- to format the medium in a device
- to read from a device
- to write to a device
- to send control information to a device

The Smartport calls, in command-number sequence, are

STATUS ($00)	Returns status information about a particular device, including general status (character or block device, read or write protection, format allowed, device on line); the device control block (set with the CONTROL call); the device newline status (character devices only); and device-specific information (number of blocks, ID string, device name, device type, device firmware version).
READ BLOCK ($01)	Reads one 512-byte block from a disk device, and writes it to memory.

WRITE BLOCK ($02)	Writes one 512-byte block from memory to a disk device.
FORMAT ($03)	Prepares all blocks on a block device for reading and writing.
CONTROL ($04)	Controls some device functions, including soft resets, setting the device control block (which controls global aspects of the device's operating environment), setting newline status (character devices only), and device interrupts. Several CONTROL calls are device-specific.
INIT ($05)	Resets all resident devices. A global reset is done automatically on startup or system resets from the keyboard; an application should never have to reset all devices.
OPEN ($06)	Prepares a character device for reading or writing.
CLOSE ($07)	Tells a character device that a sequence of reads or writes is over.
READ ($08)	Reads a specified number of bytes from a specified device.
WRITE ($09)	Writes a specified number of bytes from memory to a specified device.

The following sections describe each Smartport call, including the command number, the parameter list, and error codes. The calls are discussed in command-number order in this format:

Command name: The name used to identify the call.

Command number: A hexadecimal number that specifies which call is being made to the Smartport.

Parameter list: A list of required call parameters.

General description: What the call does and what you use it for.

Parameter descriptions: A description of each parameter and the data it refers to. When a parameter refers to a status or control code, the meaning of each code number is discussed.

Possible errors: A list of the error codes that can be returned by this call. A complete list of Smartport error codes is included at the end of this chapter.

STATUS

Command number $00

Parameter list $03 (parameter count)
Unit number
Status list pointer (low byte, high byte)
Status code

The STATUS call returns status information about a specified device. The type of information returned is determined by the device and its status-code parameter. The status list pointer defines where the status information is returned to.

STATUS returns the number of bytes of status information that it generates in the X and Y registers, the low byte of this number in the X register, and the high byte in the Y register.

Parameter descriptions

Parameter count
1-byte value — Three for this call.

Unit number
1-byte value — The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01–$7E and are assigned according to the devices' positions in the chain.

Important You can get the status of the Smartport itself if you use a unit number of $00 and a status code of $00 in a STATUS call (see the discussion beginning "Status code = $00," below).

Status list pointer

2-byte value — Points to the buffer to which the status is to be returned. The length required for the buffer varies depending on the status request being made.

Status code

1-byte value — Indicates what kind of status request is being made. Status codes are in the range $00–$FF, as follows:

Code	Status returned
$00	Return device status
$01	Return device control block (DCB) (not supported by UniDisk 3.5)
$02	Return newline status (character devices only) (not supported by UniDisk 3.5)
$03	Return device information block (DIB)
$05	Return UniDisk 3.5 status

Status code = $00 returns a device status consisting of four bytes. The first is the general status byte, with the following format:

Bit	Description
7	0 = character device, 1 = block device
6	1 = write allowed
5	1 = read allowed
4	1 = device on line or disk in drive
3	0 = format allowed
2	0 = medium write protected (block devices only)
1	1 = device currently interrupting
0	1 = device currently open (character devices only)

If the STATUS call is for a block device, the next three bytes (low byte first) are the size in 512-byte blocks. The maximum size is 16 million ($FFFFFF) blocks (about 8 gigabytes). If the call is for a character device, these three bytes must be set to 0.

A STATUS call with status code = $00 and unit number = $00 returns the status of the Smartport itself. In this case, the status list consists of 8 bytes, as follows:

```
STAT_LIST   DFB   Number_Devices      Devices hooked to PC
            DFB   Interrupt_Status   Bit 6 clear = interrupt sent
            DFB                      Reserved
            DFB                      Reserved
            DFB                      Reserved
            DFB                      Reserved
            DFB                      Reserved
            DFB                      Reserved
```

The Number_Devices byte returns the total number of intelligent devices attached to the Smartport. The Interrupt_Status byte is a copy of the asynchronous communications interface adapter (ACIA) status register at the time of the interrupt, and is used to indicate that a device requires interrupt servicing. If the sixth bit of this byte equals 0, one or more devices in the Smartport bus daisy chain must be serviced; your interrupt handler must poll each device on the chain to determine which ones.

- *About interrupts:* Devices that require interrupt servicing must use the EXTINT line on the Apple IIc's external disk port connector to be supported by the Smartport.

  For example, UniDisk 3.5 does not support this line, and so cannot generate interrupts to the Smartport. See the description of the CONTROL command for instructions on enabling Smartport interrupts. See Appendix E for more information about programming with interrupts.

Status code = $01 returns the device control block (DCB). The DCB is used to control various operating characteristics of a device and is device dependent. Each device has a default DCB, which can be altered with a CONTROL call. The first byte (the count byte) gives the number of bytes in the control block (*not* including the count byte), so the length never exceeds 256 bytes (257 including the count byte). Note that UniDisk 3.5 has no DCB and returns an error (BadCtl $21) in response to this call.

On newline read mode, see Chapter 4 in the *ProDOS Technical Reference Manual.*

Status code = $02 returns newline status. Newline status applies only to character devices. A status code = $02 passed to a block device returns a BadCtl ($21) error.

Status code = $03 returns the device information block (DIB). The device's information block identifies the device, its type, and various other attributes. The returned status list has the following form:

```
STAT_LIST   DFB   Device_Statbyte1      Same as byte 1 in status
                                        code = 0
            DFB   Device_Size_Lo        Number of blocks
                                        (block device)
            DFB   Device_Size_Med       Number of blocks
                                        (middle byte)
            DFB   Device_Size_Hi        Number of blocks (high
                                        byte)
            DFB   ID_String_Length      Length in bytes (16 max.)
            ASC   '<device name>'       7-bit ASCII, uppercase,
                                        padded with spaces, 8th
                                        bit always=0 (16 bytes)
            DFB   Device_Type_Code
            DFB   Device_Subtype_Code
            DW    Version               Device firmware version
                                        number
```

Status code = $05 returns the UniDisk 3.5 status. This call allows a diagnostic program to get more detailed information about the cause of a read or write error, and to examine the contents of the 65C02's registers after a CONTROL call with control code = $05. The returned status list has this form:

```
STAT_LIST   DFB   $00
            DFB   Error       Soft Error byte (see below)
            DFB   Retries     Number of retries (see below)
            DFB   $00
            DFB   A_Value     Acc value after a CONTROL EXECUTE
                              call
            DFB   X_Value     X value after EXECUTE
            DFB   Y_Value     Y value after EXECUTE
            DFB   P_Value     Processor status value after EXECUTE
```

The Error byte returned by a STATUS call with status code = $05 contains the following bits:

Bit	Description
7	0
6	0
5	1 = address field mark or checksum error
4	1 = data field checksum error
3	1 = data field bitslip mark mismatch
2	1 = seek error; unexpected track value found in address field
1	0
0	0

The Retries byte returned by a STATUS call with status code = $05 specfies the number of address fields that had to be passed before the operation was completed. This information could be used, for example, to determine the number of passes necessary to read a data field correctly: If Retries is found to be greater than the number of sectors on the target track, then more than one pass was required.

The last four bytes of the status list are set only after a CONTROL call with control code = $05, and are 0 after any other call (STATUS calls do not clear the status bytes).

Possible errors

The following errors can be returned by the STATUS call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$21	BadCtl	Invalid status code
$30–$3F		Device-specific errors

READ BLOCK

Command number	$01
Parameter list	$03 (parameter count) $03 (parameter count) Unit number Data buffer (low byte, high byte) Block number (low byte, mid byte, high byte)

The READ BLOCK call reads one 512-byte block into memory from the block device specified by the unit-number parameter. The block of data is placed in a buffer starting at the address specified by the data-buffer parameter.

Parameter descriptions

Parameter count
1-byte value — Three for this call.

Unit number
1-byte value — The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01–$7E and are assigned according to the devices' positions in the daisy chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Smartport.

Data buffer
2-byte value — Points to the buffer into which the data are read. The buffer must be 512 or more bytes in length.

Block number
3-byte value — The logical address of a block of data to be read. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical blocks is performed by the device. (The most significant byte is 0 for all devices currently in use.)

Possible errors

The following errors can be returned by the READ BLOCK call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$27	IOError	I/O error
$28	NoDrive	No device connected
$2D	BadBlock	Invalid block number
$2F	OffLine	Device off-line or no disk in drive

WRITE BLOCK

Command number $02

Parameter list $03 (parameter count)
Unit number
Data buffer (low byte, high byte)
Block number (low byte, mid byte, high byte)

The WRITE BLOCK call writes one 512-byte block from memory to the disk device specified by the unit-number parameter. The block in memory starts at the address specified by the data-buffer parameter.

Parameter descriptions

Parameter count
1-byte value — Three for this call.

Unit number
1-byte value — The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01–$7E and are assigned according to the devices' positions in the daisy chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Smartport.

Data buffer
2-byte value — Points to the buffer from which the data are to be written.

Block number
3-byte value — The logical address of a block of data to be written. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical blocks is performed by the device. (The most significant byte is 0 for all devices currently in use.)

Possible errors

The following errors can be returned by the WRITE BLOCK call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$27	IOError	I/O error
$28	NoDrive	No device connected
$2B	NoWrite	Disk write protected
$2D	BadBlock	Invalid block number
$2F	OffLine	Device off-line or no disk in drive

FORMAT

Command number $03

Parameter list $01 (parameter count)
Unit number

The FORMAT call prepares all blocks on the recording medium of a block device for reading and writing. The formatting done by this call is specific to each device and is not linked to any operating system; for example, bitmaps and catalogs are not written by this call.

Parameter descriptions

Parameter count
1-byte value — One for this call.

Unit number
1-byte value — The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01–$7E and are assigned according to the devices' positions in the daisy chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Smartport.

Possible errors

The following errors can be returned by the FORMAT call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$27	IOError	I/O error
$28	NoDrive	No device connected
$2B	NoWrite	Disk write protected
$2F	OffLine	Device off-line or no disk in drive

CONTROL

Command number $04

Parameter list $03 (parameter count)
Unit number
Control list (low byte, high byte)
Control code

The CONTROL call sends control information to the device. The information can be of a general nature (such as resets or interrupts), or device-specific (such as Download to UniDisk 3.5 RAM).

Important A CONTROL call to unit number $00 sends control information to the Smartport itself. See the discussions of control code = $00 and control code = $01, below.

Parameter descriptions

Parameter count
1-byte value — Three for this call.

Unit number
1-byte value — The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01–$7E and are assigned according to the devices' positions in the daisy chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Smartport. Use a unit number of $00 in the CONTROL call to send control information to the Smartport itself.

Control list
2-byte value

Points to the buffer containing the control information. The first two bytes (the count bytes, low byte first) of the control list specify the number of bytes in the list (*not* including the count bytes); the remainder of the list contains the control information passed to the device.

Important Every CONTROL call must have a control list; if no control information is being passed, then the control list consists of the count bytes only:

```
CTRL_LIST DW $00
```

Control code
1-byte value

The number of the control request being made. Control codes are in the range $00–$FF. The following requests are not device specific:

Code	Control function
$00	Reset the device
$01	Set device control block (DCB)
$02	Set newline status (character devices only)
$03	Service device interrupt

Control requests to unit number $00 are sent to the Smartport itself:

Code	Control function
$00	Enable interrupts from Smartport
$01	Disable interrupts from Smartport

Specific devices may respond to some or all of these additional control requests:

Code	Control function
$04	Eject disk
$05	Run a 65C02 subroutine
$06	Set download address
$07	Download to device RAM

Control code = $00 performs a warm reset of the device and generally returns "housekeeping" values to some reset value. The control list for this call is device dependent.

The control list for this call for UniDisk 3.5 devices is

```
CTRL_LIST DW  $00     No parameters are passed.
```

A CONTROL call with control code = $00 and unit number = $00 enables interrupts from the Smartport. This informs the firmware that external interrupts are possible, and directs it to call the user's interrupt handler if an interrupt occurs. It also turns on the ACIA for port 1.

When the user's interrupt handler identifies an external interrupt, you can determine if it came from the Smartport by making a STATUS call with unit number = $00 and control code = $00. See Appendix E for more information on handling interrupts.

Control code = $01 alters the contents of the device control block (DCB). The DCB is used to set global aspects of a device's operating environment. Each device has a default setting for the DCB, set on initialization. Because the length of the DCB is device dependent, you should first read in the DCB with the STATUS call, then alter the bits of interest, and finally, use the same byte string as the control block for the CONTROL call. The first byte (the count byte) of the DCB gives the number of bytes in the control block (*not* including the count byte), so the length never exceeds 257 bytes, including the count byte.

Note that because UniDisk 3.5 has no DCB, a Set DCB CONTROL call to UniDisk 3.5 returns an error (BadCtl $21).

A CONTROL call with control code = $01 and unit number = $00 disables interrupts from the Smartport. This call turns off the ACIA for port 1 and sets the least significant bit of the ACIA control register to 0.

Control code = $02 sets a character device to newline enabled or newline disabled.

Control code = $03 sends a device service interrupt. This code is to be used as needed for interrupt-driven devices.

Control code = $04 ejects a disk. This code is to be used for devices that support an auto-eject feature. This code causes UniDisk 3.5 to auto-eject a disk. There are no parameters in the control list, and no errors are returned if the disk ejected correctly or there was no disk in the drive. Error code $27 (IOError) is returned if the eject failed—that is, if a disk is still in the drive. The control list for UniDisk 3.5 is

```
CTRL_LIST DW   $00     No parameters are passed.
```

Warning Control codes $05 and higher are reserved; use of some of these codes can cause your system to crash.

Possible errors

The following errors can be returned by the CONTROL call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$21	BadCtl	Invalid control code
$22	BadCtlParm	Invalid parameter list
$30–$3F		Device-specific errors

INIT

Command number $05

Parameter list $01 (parameter count)
$00 (unit number)

The INIT call resets all intelligent devices attached to the Smartport. The Smartport goes through an initialization sequence, cold-resetting all devices and sending each its unit number. This call is made automatically on startup; an application should never have to make this call.

Parameter descriptions

Parameter count
1-byte value — One for this call.

Unit number
1-byte value — The unit number used in this call is always $00.

Possible errors

The following errors can be returned by the INIT call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$28	NoDrive	No device connected

OPEN

Command number $06

Parameter list $01 (parameter count)
Unit number

The OPEN call prepares a character device for reading or writing.

Note that since UniDisk 3.5 is a block device, it does not accept this call. An attempt to use an OPEN call with UniDisk 3.5 will result in an error (BadCmd $01).

Parameter descriptions

Parameter count
1-byte value — One for this call.

Unit number
1-byte value — The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01–$7E and are assigned according to the devices' positions in the daisy chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Smartport.

Possible errors

The following errors can be returned by the OPEN call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$28	NoDrive	No device connected
$2F	OffLine	Device off-line or no disk in drive

CLOSE

Command number $07

Parameter list $01 (parameter count)
Unit number

The CLOSE call tells a character device that a sequence of reads or writes is over.

Note that since UniDisk 3.5 is a block device, it does not accept this call. An attempt to use a CLOSE call with UniDisk 3.5 will result in an error (BadCmd $01).

Parameter descriptions

Parameter count
1-byte value — One for this call.

Unit number
1-byte value — The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01–$7E and are assigned according to the devices' positions in the daisy chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Smartport.

Possible errors

The following errors can be returned by the CLOSE call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$28	NoDrive	No device connected
$2F	OffLine	Device off-line or no disk in drive

READ

Command number $08

Parameter list $04 (parameter count)
Unit number
Buffer pointer (low byte, high byte)
Byte count (low byte, high byte)
Address pointer (low byte, mid byte, high byte)

The READ call reads into memory the number of bytes specified by the byte-count parameter. The bytes are placed in a buffer starting at the address specified by the buffer-pointer parameter.

Parameter descriptions

Parameter count
1-byte value — Four for this call.

Unit number
1-byte value — The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01–$7E and are assigned according to the devices' positions in the daisy chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Smartport.

Buffer pointer
2-byte point — Points to the buffer into which the data is read. The buffer must be large enough to contain the number of bytes requested by the byte-count parameter.

Byte count
2-byte value — Specifies the number of bytes to be transferred.

Address pointer
3-byte value — Specifies the address to start reading from. The meaning of this parameter depends on the device being read.

Possible errors

The following errors can be returned by the READ call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$27	IOError	I/O error
$28	NoDrive	No device connected
$2D	BadBlock	Invalid block number
$2F	OffLine	Device off-line or no disk in drive

WRITE

Command number $09

Parameter list $04 (parameter count)
Unit number
Buffer pointer (low byte, high byte)
Byte count (low byte, high byte)
Address pointer (low byte, mid byte, high byte)

The WRITE call writes from memory the number of bytes specified by the byte-count parameter to the specified unit. The bytes in memory start at the address indicated by the buffer-pointer parameter. The meaning of the address pointer depends on the type of device (see parameter descriptions).

Parameter descriptions

Parameter count
1-byte value — Four for this call.

Unit number
1-byte value — The Smartport assigns each device a unique number during initialization (on startup and cold reset). The numbers are in the range $01–$7E and are assigned according to the devices' positions in the daisy chain. A unit number of $00 in the STATUS call returns the number of devices connected to the Protocol Converter.

Buffer pointer
2-byte value — Points to the buffer from which the data is to be written.

Byte count
2-byte value — Specifies the number of bytes to be transferred.

Address pointer
3-byte value — Specifies the address to start writing from. The meaning of this parameter depends on the device being written to.

Possible errors

The following errors can be returned by the WRITE call:

$01	BadCmd	An unimplemented command was issued
$04	BadPCnt	Bad call parameter count
$06	BusErr	Communications error
$27	IOError	I/O error
$28	NoDrive	No device connected
$2D	BadBlock	Invalid block number
$2F	OffLine	Device off-line or no disk in drive

An example: issuing a Smartport call

Here is an example of a program that issues a STATUS call to the Smartport to obtain information about a device.

The code for the Smartport in the version of the Apple IIc that supports UniDisk 3.5 always begins at address $C500; however, to ensure compatibility with the Apple IIe, your programs should always do a search for the Smartport, as in this example.

```
0000:                1 *
0000:                2 *
0000:                3 *
0000:                4 *  This example shows how to find
0000:                5 *  and use a PC interface. A search
0000:                6 *  is made for a PC, and when one is
0000:                7 *  found, a vector is set up which
0000:                8 *  points to the PC entry. Then a
0000:                9 *  Device Information Block STATUS call
0000:               10 *  is made, and if successful, the name
0000:               11 *  string embedded in the DIB is output
0000:               12 *  to the screen. Only the first device
0000:               13 *  in the chain is accessed.
0000:               14 *
0000:               15 *
0000:               16           MSB   ON
0000:               17 *
0000:               18 *
0000:       0006    19 ZPTempL   equ   $0006    ;Temporary zero
0000:               20 *                          page storage
0000:       0007    21 ZPTempH   equ   $0007
0000:               22 *
0000:       FDED    23 COut      equ   $FDED    ;Console output
0000:       FD8E    24 CROut     equ   $FD8E    ;Carriage return
0000:               25 *
0000:       0000    26 StatusCmd equ   0
0000:               27 *
0000:               28 *
0300:       0300    29           org   $300
0300:               30 *
0300:               31 * Find a Smartport in one of the
0300:               32 *  slots.
0300:               33 *
0300:20 43 03       34           jsr   FindPC
0303:B0 1C   0321   35           bcs   Error
0305:               36 *
0305:               37 * Now make the DIB call to the first guy
0305:               38 *
```

```
0305:20 67 03      39            jsr   Dispatch
0308:00            40            dfb   StatusCmd
0309:6A 03         41            dw    DParms
030B:B0 14   0321  42            bcs   Error
030D:              43 *
030D:              44 * Got the DIB; now print the name string
030D:              45 *
030D:A2 00         46            ldx   #0
030F:       030F   47 morechars  equ   *
030F:BD 74 03      48            lda   DIBName,x
0312:09 80         49            ora   #$80      ;COut wants high
0314               50 *                            Bit set
0314:              51 * 0314:20 ED FD       52            jsr   COut
0317:E8            53            inx
0318:EC 73 03      54            cpx   DIBNameLen
031B:90 F2   030F  55            blt   morechars
031D:              56 *
031D:20 8E FD      57            jsr   CROut     ;Finish it off
0320:              58 *                            with a return
0320:              59 *
0320:60            60            rts
0321:              61 *
0321:              62 *
0321:       0321   63 Error      equ   *
0321:              64 *
0321:              65 * There's either no PC around, or there
0321:              66 * was no Unit #1... give message
0321:              67 *
0321:A2 00         68            ldx   #0
0323:       0323   69 err1       equ   *
0323:BD 2F 03      70            lda   Message,x
0326:F0 06   032E  71            beq   errout
0328:20 ED FD      72            jsr   COut
032B:E8            73            inx
032C:D0 F5   0323  74            bne   err1
032E:              75 *
032E:       032E   76 errout     equ   *
032E:60            77            rts
032F:              78 *
032F:CE CF A0 D0   79 Message    asc   'NO PC OR NO DEVICE'
0341:8D 00         80            dfb   $8D,0
0343:              81 *
0343:              82 *
0343:       0343   83 FindPC     equ   *
0343:              84 *
0343:              85 * Search slot 7 to slot 1 looking for
0343:              86 *  signature bytes
0343:              87 *
0343:A2 07         88            ldx   #7        ;Do for seven
0345:              89 *                            slots
```

```
0345:A9 C7          90           lda   #$C7
0347:85 07          91           sta   ZPTempH
0349:A9 00          92           lda   #$00
034B:85 06          93           sta   ZPTempL
034D:               94 *
034D:       034D    95 newslot   equ   *
034D:A0 07          96           ldy   #7
034F:               97 *
034F:       034F    98 again     equ   *
034F:B1 06          99           lda   (ZPTempL),y
0351:D9 70 03      100           cmp   sigtab,y     ;One of four
0354:              101 *                           byte signature
0354:F0 07   035D  102           beq   maybe        ;Found one
0356:              103 *                           signature byte
0356:C6 07         104           dec   ZPTempH
0358:CA            105           dex
0359:D0 F2   034D  106           bne   newslot
035B:              107 *
035B:              108 * If we get here, it's because we couldn't
035B:              109 *  find a Smartport.
035B:              110 * Exit with the carry set.
035B:              111 *
035B:38            112           sec
035C:60            113           rts
035D:              114 *
035D:              115 * If we get here, it means that one or
035D:              116 *  more of the signature bytes
035D:              117 *  for this card are what we're looking
035D:              118 *  for. Decrement the byte
035D:              119 *  counter and branch back to verify any
035D:              120 *  remaining bytes.
035D:              121 *
035D:       035D   122 maybe     equ   *
035D:88            123           dey
035E:88            124           dey                ;If N=1 then
035F:              125 *                         all sig bytes okay
035F:10 EE   034F  126           bpl   again
0361:              127 *
0361:              128 * Found a Smartport interface.
0361:              129 *  Set up the call address.
0361:              130 * We already have the high byte ($CN);
0361:              131 *  we just need the low byte.
0361:              132 *
0361:       0361   133 foundPC   equ   *
0361:A9 FF         134           lda   #$FF
0363:85 06         135           sta   ZPTempL
0365:A0 00         136           ldy   #0           ;For
0367:              137 *                           indirect load
0367:B1 06         138           lda   (ZPTempL),y  ;Get the
0369:              139 *                             byte
```

```
0369:              140 *
0369:              141 * Now the Acc has the low order ProDOS
0369:              142 *  entry point. The PC entry is
0369:              143 *  three locations past this...
0369:              144 *
0369:18            145           clc
036A:69 03         146           adc   #3
036C:85 06         147           sta   ZPTempL
036E:              148 *
036E:              149 * Now ZPTempL has the PC entry point.
036E:              150 *  Return with carry clear.
036E:              151 *
036E:18            152           clc
036F:60            153           rts
0370:              154 *
0370:              155 *
0370:              156 * These are the PC signature bytes in
0370:              157 *  their relative order.
0370:              158 *  The $FF bytes are filler bytes and
0370:              159 *  are not compared.
0370:              160 *
0370:FF 20 FF 00   161 sigtab    dfb   $FF,$20,$FF,$00
0374:FF 03 FF 00   162           dfb   $FF,$03,$FF,$00
0378:              163 *
0378:              164 *
0378:       0378   165 Dispatch  equ   *
0378:6C 06 00      166           jmp   (ZPTempL)    ;Simulate
037B:              167 *                   an indirect JSR to PC
037B:              168 *
037B:              169 *
037B:       037B   170 DParms    equ   *
037B:03            171 DPParmCt  dfb   3             ;Status
037C:              172 *             calls have three parameters
037C:01            173 DPUnit    dfb   1
037D:80 03         174 DPBuffer  dw    DIB
037F:03            175 DPStatCode dfb  3
0380:              176 *
0380:              177 *
0380:       0380   178 DIB       equ   *
0380:00            179 DIBStatByte1 dfb 0
0381:00 00 00      180 DIBDevSize dfb  0,0,0
0384:00            181 DIBNameLen dfb  0
0385:       0010   182 DIBName   ds    16,0
0395:00            183 DIBType   dfb   0
0396:00            184 DIBSubType dfb  0
0397:00 00         185 DIBVersion dw   0
0399:              186 *
0399:              187 *
```

Summary of commands and parameters

The following is a summary of Smartport calls. In each case, byte 0 of the command parameter list (CmdLst) specifies the number of parameters in the command list (not including byte 0). Parameters that require more than one byte (the status list pointer, for example) are entered low byte first. The meaning of the address-pointer parameter is device specific. See the sections on the individual calls in this chapter for a discussion of each parameter.

Command	STATUS	READBLOCK	WRITEBLOCK	FORMAT	CONTROL
CmdNum	$00	$01	$02	$03	$04
CmdList Byte					
0	$03	$03	$03	$01	$03
1	Unit Num	Unit Num	Unit Num	Unit Num	Unit Num
2	Stat List Ptr	Buffer Ptr	Buffer Ptr		Ctl List Ptr
3					
4	Stat Code				Ctl Code
5		Block Num	Block Num		
6					

Command	INIT	OPEN	CLOSE	READ	WRITE
CmdNum	$05	$06	$07	$08	$09
CmdList Byte					
0	$01	$01	$01	$04	$04
1	$00	Unit Num	Unit Num	Unit Num	Unit Num
2				Buffer Ptr	Buffer Ptr
3					
4				Byte Count	Byte Count
5					
6					
7				Address Ptr	Address Ptr
8					

Unused bytes

Figure 6-1
Summary of Smartport calls

Summary of error codes

The following is a summary of Smartport call error codes, including a brief description of the possible causes for each. If there is no error, the C flag (in the processor status register of the 65C02 microprocessor) is cleared (0) and the accumulator (the A register) contains 0s. If the call was unsuccessful, the C flag is set (1) and the A register contains the error code.

$00		No error.
$01	BadCmd	A nonexistent command was issued. Check the command number in the Smartport call.
$04	BadPCnt	Bad call parameter count. The call parameter list was not properly constructed. Make sure the parameter list has the correct number of parameters.
$06	BusErr	A communications error between the device controller and the host. Make sure that RAM is both read-enabled and write-enabled. Check the hardware (cables and connectors) between the device and the host. Check for noise sources. Make sure the cable is properly shielded.
$11	BadUnit	Unit number $00 was used in a call other than STATUS, CONTROL, or INIT.
$21	BadCtl	The control or status code is not supported by the device.
$22	BadCtlParm	The control parameter list contains invalid information. Make sure each value is within the range allowed for that parameter.

$27	IOError	The device encountered an I/O error when trying to read or write to the recording medium. Make sure that the medium in the device is formatted and not defective and that the device is operating correctly.
$28	NoDrive	The device is not connected. This can occur if the device is not connected but its controller is, or if there is no device with the unit number specified.
$2B	NoWrite	The medium in the device is write protected.
$2D	BadBlock	The block number is outside the range allowed for the medium in the device. Note that this range depends on the type of device and the type of medium in the device (single-sided versus double-sided disk, for example).
$2F	OffLine	Device off-line or no disk in drive. Check the cables and connections. Make sure that the medium is present in the drive and that the drive is functioning correctly.
$30–$3F	DevSpec	Errors that differ from device to device. See the technical manual for the device in question for details. $40–$4F. Reserved for future expansion.
$50–$7F	NonFatal	A device-specific soft error. The operation completed successfully, but some exception condition was detected. See the technical manual for the device in question for details.

Chapter 7

Serial I/O Port 1

Serial port 1 is one of two serial I/O ports available on the Apple IIc. It is intended primarily as an output port for RS-232 devices, such as printers and plotters. It can be changed to a serial communication port (like port 2) by using the *System Utilities* disk or from a program.

Warning Although the Apple IIc serial ports are similar to the Apple IIe Super Serial Card, there are important differences. Refer to Appendix F for a summary of these differences.

Table 7-1 summarizes the characteristics of this port if used as a printer/plotter port, and is a guide to the other information in this chapter. If you change port 1 to a communication port, refer to the descriptions in Chapter 8, and use 1 instead of 2 for the port number when required.

The serial port back panel connectors are described in Chapter 11.

Table 7-1
Serial port 1 characteristics

Port number	Serial port 1.
Commands	Keyboard command: PR#1. BASIC command: PR#1. Monitor command: 1 Control-P (does not work if there is an operating system in RAM). All other commands: See Table 7-2.
Initial characteristics	See "Characteristics of Port 1 at Startup."
Hardware page locations	See Table 7-3.
Monitor firmware routines	None.
I/O firmware entry points	See Table 7-4.
Use of screen holes	See Table 7-5.
Use of other pages	None.

Using serial port 1

You can access the firmware from BASIC in the usual way—that is, by issuing Control-D (if DOS or ProDOS is in RAM) and PR#1. Subsequent output is directed to the printer (or other device) connected to serial port 1.

To direct Pascal output to the printer, you can use either #6: or PRINTER:.

Refer to Table 7-4 for the standard firmware entry points that Pascal 1.1 and 1.2 use.

Your programs can also access the port by changing the value of CSW (see Chapter 3).

Table 7-2 lists the commands you can use with serial port 1, either from a program or from the keyboard, after you issue PR#1.

UniDisk 3.5 Commands followed by an asterisk in Table 7-2 (with the exception of L) are available only on the version of the Apple IIc that supports UniDisk 3.5. These commands can be toggled by following them directly with E (enable) or D (disable).

Each command must be preceded by Control-I (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command. You do not have to press Return after commands that you have entered from the keyboard, or send the return character from your program if it is sending commands to the port. You can type more than one command on a line, but each must be preceded by the command character.

Table 7-2
Printer port commands

Command	Description
nnn	Sets new line width of nnn (from 1 through 255).This command must be followed by N (see below) or by a carriage return.
nnB	Sets baud rate to value corresponding to nn:

nn	Rate	nn	Rate	nn	Rate
1	50	6	300	11	3600
2	75	7	600	12	4800
3	110 (109.92)	8	1200	13	7200
4	135 (134.58)	9	1800	14	9600
5	150	10	2400	15	1920

Table 7-2 (continued)
Printer port commands

Command	Description
C*	When enabled, this command causes a carriage return character to be sent automatically whenever the column count exceeds the printer line width. The command is normally enabled.
nD	Sets data format to values corresponding to n:

n	Data bits	Stop bits	n	Data bits	Stop bits
0	8	1	4	8	2
1	7	1	5	7	2
2	6	1	6	6	2
3	5	1	7	5	2

F*	When this command is enabled, your Apple IIc accepts data from the keyboard as well as from the serial port. You can use this to disable the keyboard before receiving or sending data to prevent accidental keystrokes from disrupting the data flow. Be sure that your program reenables the keyboard when the data transfer is complete. This command is available only from BASIC and is normally enabled.
I	Echoes printer output on the screen.
K	Disables automatic line feed after carriage return.
L*	Generates line feed after carriage return. Normally, this command is enabled. Disabling it has the same effect as the K command.
M*	When this command is enabled, all incoming line feed characters are masked (removed from the data stream). Normally this command is enabled.

Table 7-2 (continued)
Printer port commands

Command	Description
nnnN	Changes line width to nnn (from 1 through 255; nnn is optional); does not echo printer output on the screen. *Note:* 0N does not disable automatic generation of carriage return; to do so, use Z command, put 0 directly in location $0579, or use the *System Utilities* disk.
nP	Sets parity corresponding to n:

n	Parity	n	Parity
0	None	4	None
1	Odd	5	MARK (1)
2	None	6	None
3	Even	7	SPACE (0)

R	Resets port 1 and exits from serial port 1 firmware.
S	Sends a 233-millisecond BREAK character (used with some printers to synchronize with serial ports).
X*	When enabled, this command turns on the XON/XOFF protocol: the Apple IIc looks for the XOFF ($13) character and responds by halting transmission until an XON ($11) is received. Normally this command is disabled.
Z	Zaps (ignores) further command characters until Control-Reset or PR#1. Does not format output or insert carriage returns into output stream.

Note: The commands themselves are letter commands, not control characters.

* Command (with the exception of L) is available only on the version of the Apple IIc that supports UniDisk 3.5. Command can be toggled: If you follow the command with E (with no intervening space), the command is enabled. If you follow the command with D (with no intervening space), command is disabled. The L command is available on the earlier Apple IIc, but cannot be toggled there.

The serial port 1 command character is set as Control-I when the Apple IIc is turned on. You can change it to a different control character by sending the current control character followed immediately by the new control character you want. This is useful if you want to be able to send Control-I to the printer without firmware intervention. For example, to change the command character from Control-I to Control-V, send Control-I Control-V either from the keyboard or from a program. (Control-V and Control-W are the recommended substitute control characters.) To change the command character back again, send Control-V Control-I. Don't slip any spaces between the control characters that you send.

Warning Do not use Control-A, -B, -C, -H, -J, -L, -M, or -Y: Apple IIc firmware may intercept these control characters, causing unpredictable results.

The following are examples of valid commands and command sequences. These examples all show commands being entered from the keyboard, but your programs can send the characters just as well. Remember to issue a PR#1 before starting to send commands to serial port 1.

To echo output to the display screen:

Control-I I

To set line width 72, disable line feed, and echo:

Control-I K Control-I 7 2 N

To change control character to Control-V:

Control-I Control-V Return

To set up the serial port to allow sending Control-I as part of a character stream:

Control-V (command) Return

Characteristics of port 1 at startup

After power-up, the printer firmware sets the following configuration:

- 9600 baud
- eight data bits, no parity bits, two stop bits
- 80-column line width; no echo to display screen
- firmware supplies line feed after carriage return
- command character is set to Control-I (see below)

These values are stored in the auxiliary memory screen holes (Table 7-5). You can change some of these settings from the keyboard by typing `PR#1`, the command character, and one of the commands listed in Table 7-2. How port characteristics change as a result of various activities is described under "Changing Port 1 Characteristics" later in this chapter.

Hardware page locations for port 1

ACIA stands for *asynchronous communication interface adapter,* a serial I/O chip. Note in Chapter 11 that some of the bit assignments for this port differ from those for port 2.

Table 7-3 lists for serial port 1 the addresses and bit assignments of its hardware registers on page $C0. The registers are internal to a 6551 **ACIA;** their bit assignments are described in Chapter 11.

Warning This table is for your information only. To avoid having problems with the system, you should **never** try to directly access the hardware. Instead, use the Apple IIc's built-in firmware in your programs.

Table 7-3
Port 1 hardware page locations

Location	Description
$C090–$C097	Reserved
$C098	ACIA transmit/receive data register
$C099	ACIA status register
$C09A	ACIA command register
$C09B	ACIA control register
$C09C–$C09F	Reserved

I/O firmware support for port 1

Table 7-4 lists the locations and values of the I/O firmware protocol table. This standardized protocol is available for use by any application program. Chapter 3 describes how to use this protocol.

Table 7-4
Port 1 I/O firmware protocol

Address	Value	Description
$C105	$38	Pascal ID byte.
$C107	$18	Pascal ID byte.
$C10B	$01	Generic signature byte of firmware cards.
$C10C	$31	Same ID as for Super Serial Card.
$C10D	$ii	$C1ii is entry point of initialization routine (PInit).
$C10E	$rr	$C1rr is entry point of read routine (PRead).
$C10F	$ww	$C1ww is entry point of write routine (PWrite).
$C110	$ss	$C1ss is entry point of the status routine (PStatus).
$C111	non-zero	No optional routines.

Screen hole locations for port 1

The ACIA register bits are defined in Chapter 11.

Table 7-5 lists the screen hole locations that serial port 1 uses. Note that the auxiliary memory locations are reserved for startup value settings, which are listed and interpreted in the table.

Table 7-5
Port 1 screen hole locations

Auxiliary memory screen holes (firmware loads values at power-up)

Location	Description
$0478	$9E (ACIA control reg: eight data + two stop bits, 9600 baud)
$0479	$0B (ACIA command reg: no parity)
$047A	$40 (flags: no echo, auto LF after CR, serial port)

Bit	Interpretation
7	Echo output on display (0 = no echo)
6	Generate LF after CR (0 = no LF)
5–1	Always = 0 (reserved)
0	1 = communication port; 0 = serial printer port

Table 7-5 (continued)
Port 1 screen hole locations

Auxiliary memory screen holes (firmware loads values at power-up)

Location	Description
$047B	$50 (printer width: 80 columns)

Bit	Interpretation
7–0	Printer width (0 = do not insert CR)

Main memory screen holes

Location	Description
$0479	Reserved
$04F9	Reserved
$0579	Printer width (1–255; 0 = disable formatting)
$05F9	Temporary storage location
$0679	Bit 7 = 1 while the firmware is parsing a command string
$06F9	Current command character (initially Control-I)
$0779	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return
$07F9	Current printer column

Changing port 1 characteristics

Figure 7-1 is a diagram of where the port characteristics are stored and moved under different circumstances. You can see the following from the figure:

- When the power is first turned on, the Monitor reset firmware moves the predefined set of port characteristics listed earlier in this chapter from ROM into the auxiliary memory screen holes listed in Table 7-5.
- If you specify new characteristics using the *System Utilities* disk, the SUD software changes the values in the auxiliary memory screen holes. Your programs can do the same thing.

- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either Open Apple-Control-Reset or a simple Control-Reset. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up. (See Figure 7-1.)
- PR#1 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main memory screen holes.
- A program can change values in the main memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location $0579.
- The firmware uses the port as it is defined in the main memory screen holes at any given time. You should use the commands listed in Table 7-2 to change them.

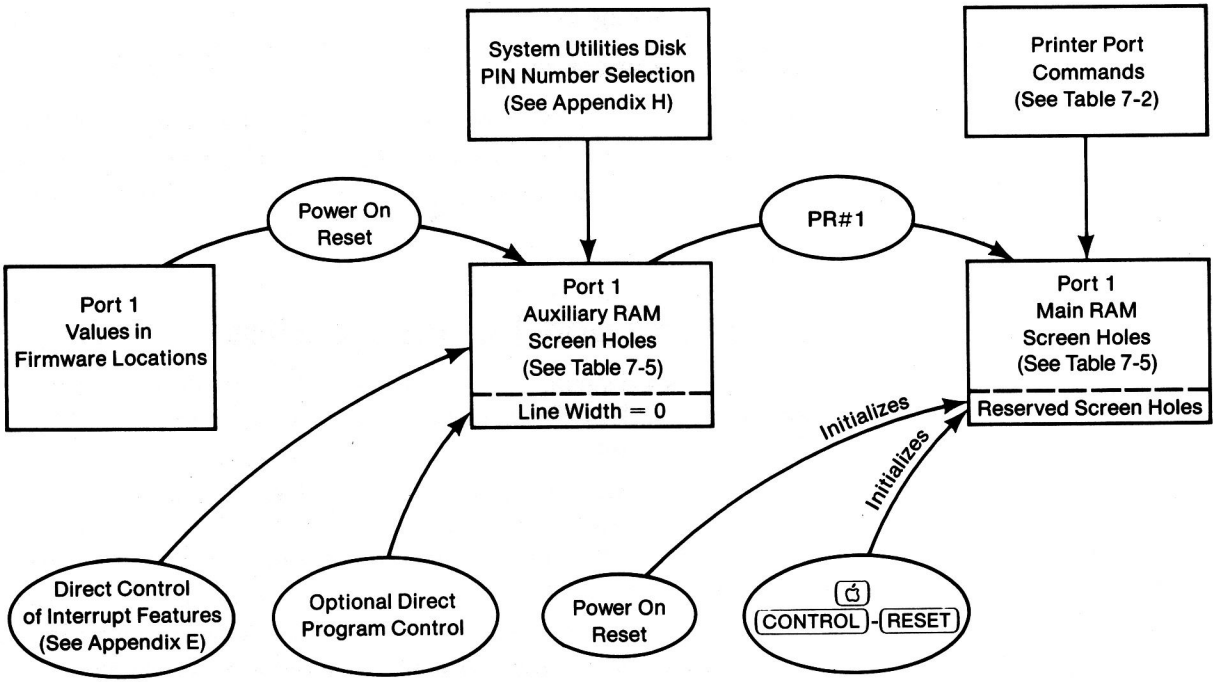


Figure 7-1
Diagram of port 1 characteristics storage

Data format and baud rate

Serial data transfer consists of a string of 1's and 0's sent down a wire at a prearranged rate of transmission, called the baud rate.

Before transfer begins, both sender and receiver look for a continuous value of 1: this is called the carrier (Figure 7-2). When the value goes to 0, the receiver presumes it is a start bit—that is, the bit that designates the beginning of a character of data. If it lasts longer than a bit could possibly last, it is considered a BREAK signal, which some printers use for synchronization.

If the first 0 proves to be a bit, it is interpreted as the start bit. Next come the seven or eight data bits (six is seldom used with computers), low-order bit first. If parity is on, it comes next in the message. Finally, one or two stop bits appear. The stop bits have a value of 1, like the carrier. The next start bit begins transfer of the next character of data.

The parity bit provides a simple check of data validity. Odd parity means the sender counts the number of 1's among the data bits, and sends the appropriate parity bit to make the total number of 1's odd. With even parity, the sender adds the appropriate parity bit to make the total number of 1 bits even. MARK parity is always a 1 bit; SPACE parity is always a 0. The receiver can then check that the parity bit is correct.

If the baud rate is 300 and the data format is one start bit plus seven data bits plus one parity bit plus one stop bit (totaling ten bits transmitted for each byte of data sent), then the actual transfer rate is about 30 characters per second.

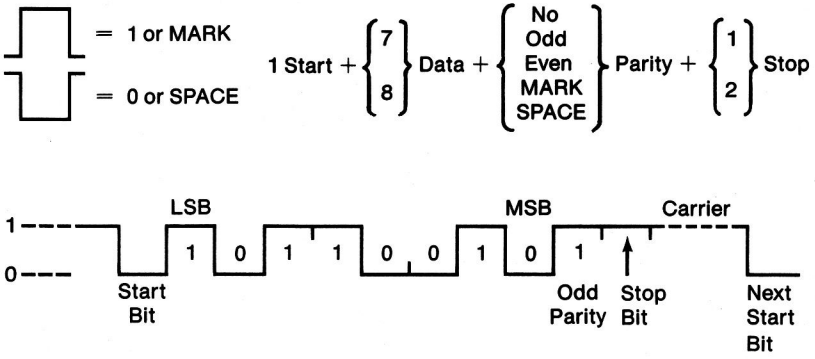


Figure 7-2
Data format

Carriage return and line feed

If you are using a typewriter and you push the carriage all the way to the right (in other words, position the printing mechanism at the left margin), you have performed a carriage return. On the other hand, turning the platen so the paper moves to the next line (or using the index key on an electric typewriter) is called a **line feed.** Most typewriters perform a line feed automatically after a carriage return, and so the two seem to be one—but they are not.

Carriage return and line feed are separate ASCII codes. Carriage return is sometimes denoted *CR;* it is ASCII code 13 ($0D). Line feed, sometimes denoted *LF,* is ASCII code 10 ($0A). Down Arrow on the Apple IIc keyboard generates a LF.

Some printers can supply a line feed automatically after detecting a carriage return; others cannot. If the printer does not supply a line feed after a carriage return and it is not supplied in the data stream, the printer keeps printing over and over on the same line. On the other hand, if both the printer and the Apple IIc firmware supply LF after CR, double line-spacing results.

If the print head keeps moving too far to the right across the page and then prints many characters on top of one another on the right, then the firmware should be instructed to furnish CR after a certain line width has been reached. If the printer prints too short a line before moving to the next line, then probably the firmware is using too small a line width.

If the printer misses characters at the beginning of each line but otherwise prints correctly, there is probably not enough time for the print mechanism to return to the left margin in response to CR. You must use a lower baud rate with such a printer.

Sending special characters

If you want to send special characters (control characters) to the printer without having them intercepted and executed by the Apple IIc firmware, use the Z command (see Table 7-2). If the only special character that causes a problem is the command character (normally Control-I for port 1), you can change just the command character instead of using the zap (Z) command. If you use the zap command, the firmware does no formatting; that is, it does not check line width or insert carriage returns or line feeds. This may be necessary to send graphics to a printer or plotter.

Displaying output on the screen

You can display printer output on the screen, but if the printer line width exceeds the 40 or 80 columns you have selected for display, you should turn off video display.

Chapter 8

Serial I/O Port 2

Serial port 2 is one of two serial I/O ports available on the Apple IIc. It is intended primarily as a communication port for modems. You can change it to a serial printer port (like port 1) using the *System Utilities* disk or from a program.

Warning Although the Apple IIc serial ports are similar to the Apple IIe Super Serial Card, there are important differences. Refer to Appendix F for a summary of these differences.

Table 8-1 summarizes the characteristics of this port and is a guide to the other information in this chapter. If you change port 2 to a serial printer port, refer to the descriptions in Chapter 7 and use 2 instead of 1 for the port number when required.

The serial port connectors are described in Chapter 11.

Table 8-1
Serial port 2 characteristics

Port number	Serial port 2.
Commands	Keyboard commands: IN#2 before Table 8-2 commands, IN#2 to accept port 2 input, PR#1 to echo input to printer, PR#2 to echo input back to port 2. BASIC commands: same. Monitor command: 2 Control-P (does not work if there is an operating system in RAM). All other commands: see Table 8-2.
Initial characteristics	See "Characteristics of Port 2 at Startup."
Hardware page locations	See Table 8-3.
Monitor firmware routines	None.
I/O firmware entry points	See Table 8-4.

Table 8-1 (continued)
Serial port 2 characteristics

Use of screen holes	See Table 8-5.
Use of other pages	In terminal mode, firmware uses auxiliary memory locations $0800–$087F to store keyboard input, and $0880–$08FF as a serial input buffer.

Using serial port 2

You can access the firmware from BASIC in the usual way—that is, by issuing Control-D (if DOS or ProDOS is in RAM) followed by IN#2 or PR#2. Subsequent input and output are routed through the modem (or other device) connected to serial port 2.

Important In terminal mode, the modem port commands listed in Table 8-2 must follow Control-D and IN#2 (*not* PR#2) and the command character (which is usually Control-A).

To transfer files to the modem under Pascal, specify REMOUT: or #8:. To transfer files from the modem under Pascal, specify REMIN: or #7:.

Refer to Table 8-4 for the standard firmware entry points that Pascal 1.1 and 1.2 use.

Table 8-2 lists the commands you can use with serial port 2, either from a program or from the keyboard, after you issue IN#2.

UniDisk 3.5 Commands followed by an asterisk in Table 8-2 (with the exception of L) are available only on the version of the Apple IIc that supports UniDisk 3.5. These commands can be toggled by following them directly with E (enable) or D (disable).

Each command must be preceded by Control-A (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command. If you press Return, you get the current video cursor again. You do not have to press Return (or send a return character) after commands. You can type more than one command on a line, but each must be preceded by the command character.

Table 8-2
Modem port commands

Command	Description
nnn	Sets new line width of nnn (from 1 through 255); this must be followed immediately by N (see below) or by carriage return.
nnB	Sets baud rate to value corresponding to nn:

nn	Rate	nn	Rate	nn	Rate
1	50	6	300	11	3600
2	75	7	600	12	4800
3	110 (109.92)	8	1200	13	7200
4	135 (134.58)	9	1800	14	9600
5	150	10	2400	15	19200

C*	When enabled, this command causes a carriage return character to be sent automatically whenever the column count exceeds the printer line width. The command is normally enabled.
nD	Sets data format to values corresponding to n:

n	Data bits	Stop bits	n	Data bits	Stop bits
0	8	1	4	8	2
1	7	1	5	7	2
2	6	1	6	6	2
3	5	1	7	5	2

F*	When this command is enabled, your Apple IIc accepts data from the keyboard as well as from the serial port. You can use this to disable the keyboard before receiving or sending data to prevent accidental keystrokes from disrupting the data flow. Be sure that your program reenables the keyboard when the data transfer is complete. This command is available only from BASIC and is normally enabled.
I	Echoes output on the screen.
K	Disables automatic line feed after carriage return.
L*	Generates line feed after carriage return. Normally, this command is enabled. Disabling it has the same effect as the K command.

Table 8-2 (continued)
Modem port commands

Command	Description
M*	When this command in enabled, all incoming line feed characters are masked (removed from the data stream). Normally this command is enabled.
nnnN	Sets line width to nnn (from 1 through 255); does not echo output on the screen. *Note:* 0N does not disable automatic generation of carriage return; to do so, use the Z command, put 0 directly in location $057A, or use the *System Utilities* disk.
nP	Sets parity corresponding to n:

n	Parity	n	Parity
0	none	4	none
1	odd	5	MARK (1)
2	none	6	none
3	even	7	SPACE (0)

Q	Quits terminal mode.
R	Resets port 2 and exits from serial port 2 firmware.
S	Sends a 233-millisecond BREAK character.
T	Enters terminal mode. Use this command after IN#2 only. Also, if you follow this command by PR#2, the Apple IIc echoes input to output. (If the other device does so too, the first character loops endlessly, locking up the system. Use Control-Reset to get out.)
X*	When enabled, this command turns on the XON/XOFF protocol: the Apple IIc looks for the XOFF ($13) character and responds by halting transmission until an XON ($11) is received. Normally this command is disabled.
Z	Zaps (ignores) further command characters until Control-Reset. Does not format output or insert carriage returns into output stream.
Control-T	This command from a remote device puts the Apple IIc in terminal mode if IN#2 is already in effect. It is the same as Control-A T typed locally.

Table 8-2 (continued)
Modem port commands

Command	Description
Control-R	This command from a remote device undoes the terminal mode command. If IN#2 and PR#2 are in effect, the remote keyboard and display become the input and output devices of the local Apple IIc. It is the same as Control-A Q typed locally.

Note: The commands themselves are letter commands, not control characters.

* Command (with the exception of L) available only on the version of the Apple IIc that supports UniDisk 3.5. Command can be toggled: If you follow the command with E (with no intervening space) the command is enabled. If you follow the command with D (with no intervening space) the command is disabled. The L command is available on the earlier Apple IIc, but it cannot be toggled there.

When the Apple IIc is turned on, the serial port 2 command character is defined as a Control-A. You can change it to a different control character by typing the current control character followed immediately by the new control character you want. This is useful if you want to be able to send Control-A to the output device without firmware intervention.

For example, to change the command character from Control-A to Control-V, send Control-A Control-V either from the keyboard or from a program. (Control-V and Control-W are the recommended substitute control characters.) To change the command character back again, send Control-V Control-A.

Warning Do not use Control-B, -C, -H, -I, -J, -L, -M, or -Y: Apple IIc firmware may intercept these control characters, causing unpredictable results.

The following are examples of valid commands and command sequences. These examples show commands being entered from the keyboard, but your programs can send the characters just as well.

To enable echo to the screen:

Control-A I

To send a break character to a remote device:

Control-A B

To change the control character to Control-V (for example, so you can send Control-A as part of a character stream):

Control-A Control-V Control-V(command)

Characteristics of port 2 at startup

After power-up, the firmware sets the following configuration:

- 300 baud
- eight data bits, no parity bits, one stop bit
- firmware does not supply line feed after carriage return
- firmware does not insert carriage returns into output stream
- firmware does not echo output to the display screen
- command character is set to Control-A

These values are stored in the auxiliary memory screen holes (Table 8-5). You can change some of these settings from the keyboard using the command character followed by one of the commands listed in Table 8-2. How port characteristics change as a result of various activities is described later in this chapter.

If you change any of these values using keyboard commands or commands from a program, subsequent accesses to the port firmware (even by another program) use the new settings instead of the power-up values. This allows you to change the settings once at system startup and get the desired configuration for subsequent uses.

Hardware page locations for port 2

Table 8-3 lists for serial port 2 the addresses of its hardware registers on page $C0. The registers are internal to a 6551 ACIA; their bit assignments are described in Chapter 11.

Warning This table is for your information only. To avoid having problems with your system, you should **never** try to directly access the hardware. Instead, use the Apple IIc's built-in firmware in your programs.

Table 8-3
Port 2 hardware page locations

Location	Description
$C0A0–$C0A7	Reserved
$C0A8	ACIA transmit/receive data register
$C0A9	ACIA status register
$C0AA	ACIA command register
$C0AB	ACIA control register
$C0AC–$C0AF	Reserved

Note in Chapter 11 that some of the bit assignments for this port differ from those for port 1.

I/O firmware support for port 2

Table 8-4 lists the values in the I/O firmware protocol table for serial port 2. This standardized protocol is available for use by any application program. Chapter 3 describes how to use this protocol.

Table 8-4
Port 2 I/O firmware protocol

Address	Value	Description
$C205	$38	Pascal ID byte.
$C207	$18	Pascal ID byte.
$C20B	$01	Generic signature byte of firmware cards.
$C20C	$31	Same ID as for Super Serial Card.
$C20D	$ii	$C2ii is entry point of initialization routine (PInit).
$C20E	$rr	$C2rr is entry point of read routine (PRead).
$C20F	$ww	$C2ww is entry point of write routine (PWrite).
$C210	$ss	$C2ss is entry point of the status routine (PStatus).
$C211	non-zero	No optional routines.

Screen hole locations for port 2

The ACIA register bits are defined in Chapter 11.

Table 8-5 lists the screen hole locations that serial port 2 uses. Note that the auxiliary memory locations are reserved for startup value settings, which are listed and interpreted in the table.

Table 8-5
Port 2 screen hole locations

Auxiliary memory screen holes (firmware loads values at power-up)

Location	Description
$047C	$16 (ACIA control reg: eight data + one stop bit, 300 baud)
$047D	$0B (ACIA command reg: no parity)
$047E	$01 (flags: no echo, no auto LF after CR, communication port)

Bit	Interpretation
7	Echo output on display (0 = no echo)
6	Generate LF after CR (0 = no LF)
5–1	Always = 0 (reserved)
0	1 = communication port; 0 = serial printer port

$047F	$00 (line length: do not add any CR to output stream)

Bit	Interpretation
7–0	Line length (0 = do not insert CR)

Main memory screen holes

Location	Description
$047A	Reserved
$04FA	Reserved
$057A	Line length (1–255; 0 = disable formatting)
$05FA	Temporary storage location
$067A	Bit 7 = 1 if and only if the firmware is currently parsing a command string
$06FA	Current command character (initially Control-I)
$077A	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return
$07FA	Current column

Changing port 2 characteristics

Figure 8-1 is a diagram of where the port characteristics are stored and moved under different circumstances. You can see the following from the figure:

- When the power is first turned on, the Monitor reset firmware moves the predefined set of port characteristics listed in Table 8-2 from ROM into the auxiliary memory screen holes listed in Table 8-5.
- If you specify new characteristics using the *System Utilities* disk, the utility software changes the values in the auxiliary memory screen holes.
- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either Open Apple-Control-Reset or a simple Control-Reset. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up.
- IN#2 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main memory screen holes.
- A program can change values in the main memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location $057A.
- The firmware uses the port as it is defined in the main memory screen holes at any given time. You should use the commands listed in Table 8-2 to change these characteristics.

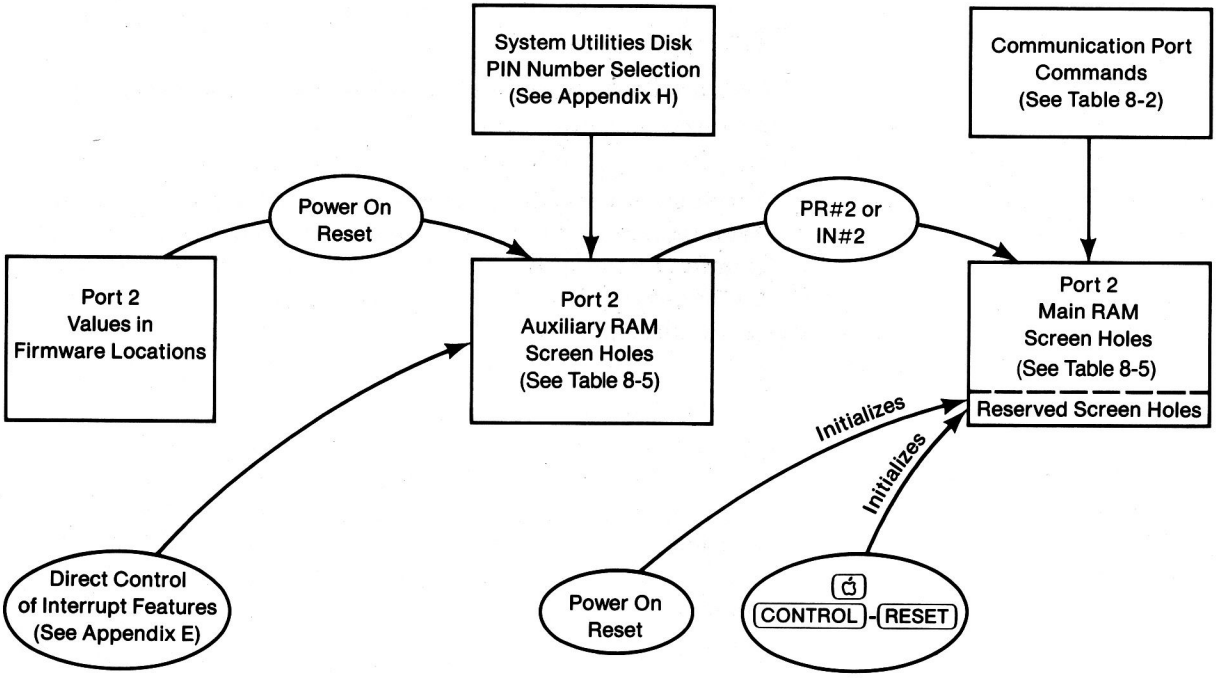


Figure 8-1
Diagram of port 2 characteristics storage

Data format and baud rate

Chapter 7 describes data format and baud rate, and explains how they apply to printers. Refer to that chapter for definitions of terms.

A noteworthy characteristic of data communication is its strangeness: sometimes the oddest changes make a given communication arrangement work or not work. You must keep this notion firmly in mind when working with serial port 2.

For example, modem communication involves quite a few elements (Figure 8-2):

- the Apple IIc and its firmware, with the baud rate, data format, and other characteristics you have selected
- the cable from the Apple IIc to the modem
- the modem
- possibly an acoustic coupler for a telephone handset

- the telephone lines, with their switching equipment, boosters, and noise
- some combination of modem, cable, and remote computer or terminal

As you can imagine, some method is required for successful data transmission. If you have problems, change only one variable at a time and then cycle through the other variables one at a time. Take nothing for granted. The data format advertised for an information service, for example, may be different from the one you end up using with the Apple IIc.

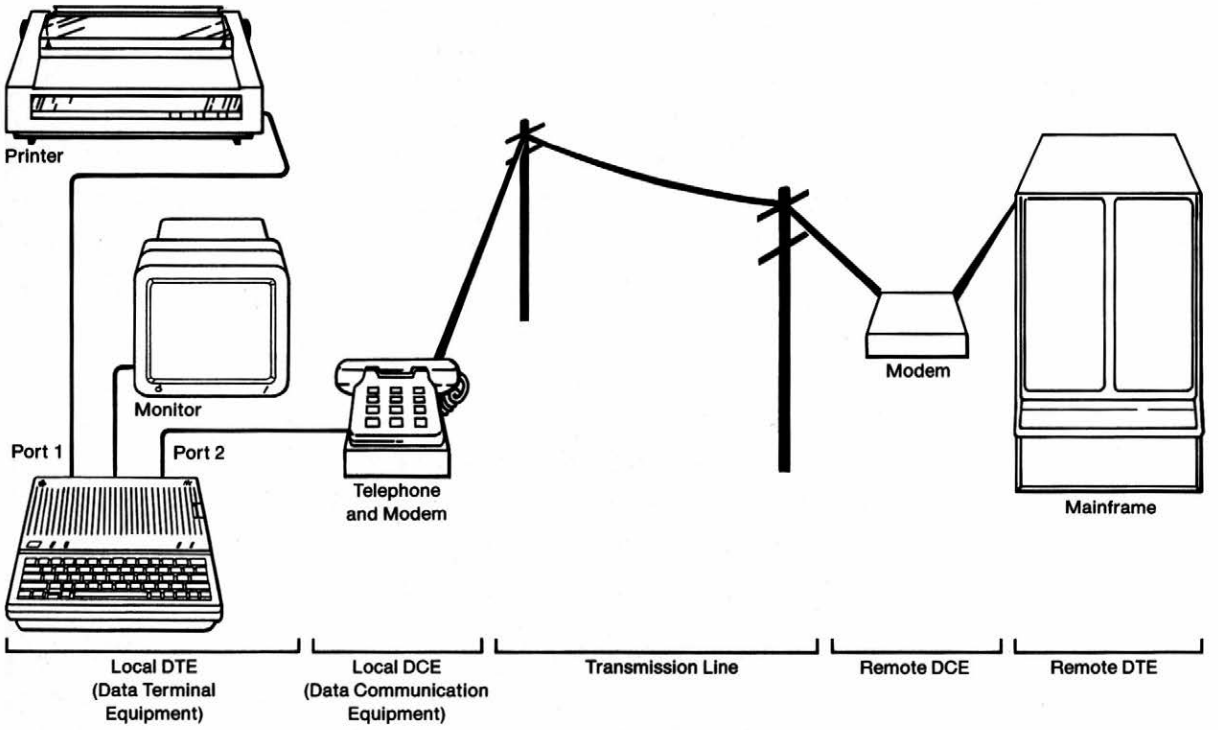


Figure 8-2
Devices in a typical communication setup

Carriage return and line feed

If you are communicating with a computer or terminal, carriage return and line feed may or may not be involved. Start off without generating them, and turn on automatic generation only as needed. They are described as used with printers in Chapter 7.

Routing input and output

This section discusses the possible ways that serial port 2 can route information. Sometimes the cause of communication problems is that information is not going where you think it is, or it is and you cannot see evidence of the fact. Figures 8-3 through 8-6 show some of the patterns of information flow you can select.

It is best to read all this material as a unit; questions that arise while you read one description may be answered elsewhere.

The simplest serial port 2 command is IN#2 (Figure 8-3). Port 2 becomes the input device. Data coming into the port gets passed to the input buffer (page $02 of main memory). Applesoft firmware and system software can see the data and carry out commands in the normal way.

Of course, you can also use just the PR#2 command—for example, if you want to send a listing to the modem.

To use port 2 for data communication, you ordinarily put it into terminal mode. Following IN#2, pressing Control-A gets the attention of the port 2 firmware, which displays a blinking question mark (?) as a prompt. Now type T to put the computer in terminal mode. In this mode, the firmware displays a blinking underscore character (_) as a prompt.

For a further description of what terminal mode does and how to get into and out of it, refer to the last section of this chapter.

In the discussion that follows, *local* refers to your Apple IIc. *Remote* refers to some other device, usually in a distant location and at the other end of a communication link. The remote device can be any ASCII-generating unit: a terminal or a computer.

If a remote computer is another Apple IIc or an Apple II series machine with a Super Serial Card in it, then *most* of the commands described here apply to it as well.

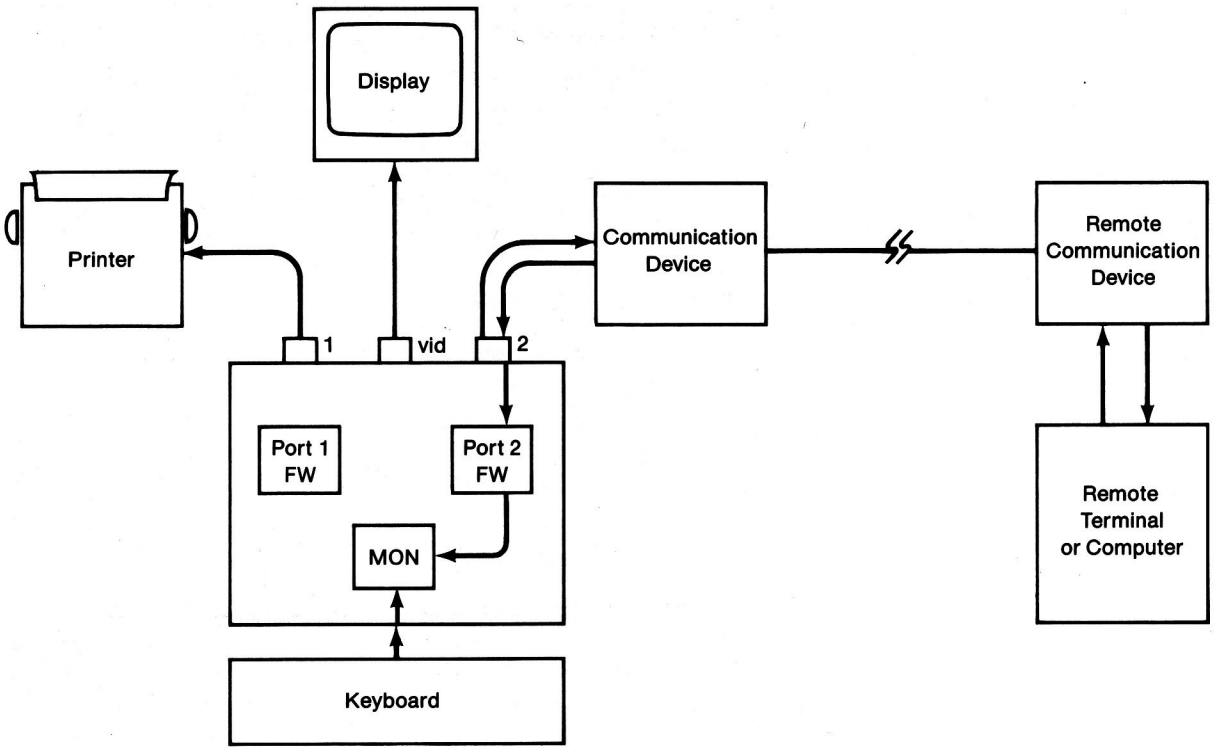


Figure 8-3
Effect of IN#2

Half-duplex operation

In half-duplex operation, information can flow from A to B or from B to A, but in only one direction at a time. In a half-duplex setup, the host does not echo back to the terminal what the terminal sends it. For half-duplex operation, use IN#2 and Control-A T (Figure 8-4) whether the Apple IIc is the host or the terminal.

IN#2 plus Control-A T is the best way to set up the computer for auto-answer operation. The T command allows port 2 firmware to exchange information with the local modem without interference from the local firmware or system software. (The remote device can always cancel the T command with Control-R if necessary, and restore terminal mode with Control-T.) Avoiding PR#2 at this point means that the Apple IIc can operate as a half-duplex terminal, half-duplex host, or full-duplex terminal. (The remote device can also issue Control-A PR#2 if PR#2 is required at the local computer.)

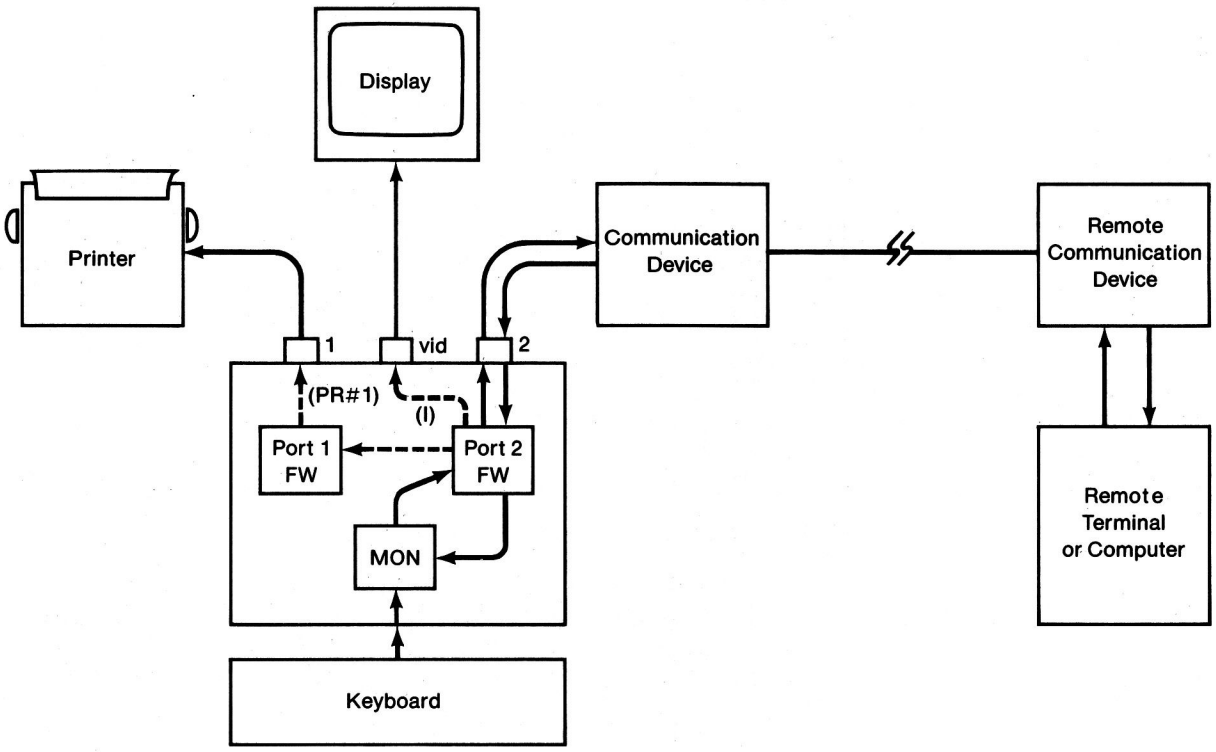


Figure 8-4
Effect of IN#2 and T command, half duplex

In half-duplex operation, the output hook is available for other uses. For example, you can issue PR#1 to print incoming messages from port 2. Use the Control-A I command to display information on the screen.

Full-duplex operation

In full-duplex operation, information can flow from A to B and from B to A simultaneously. Typically, one of the computers (the host computer) echoes its input to output, so the other computer (the terminal) can easily verify that the communication is taking place.

Figure 8-5 shows the flow of information when the Apple IIc is a full-duplex terminal. (The setup commands, IN#2 and Control-A T, are the same as for half duplex.)

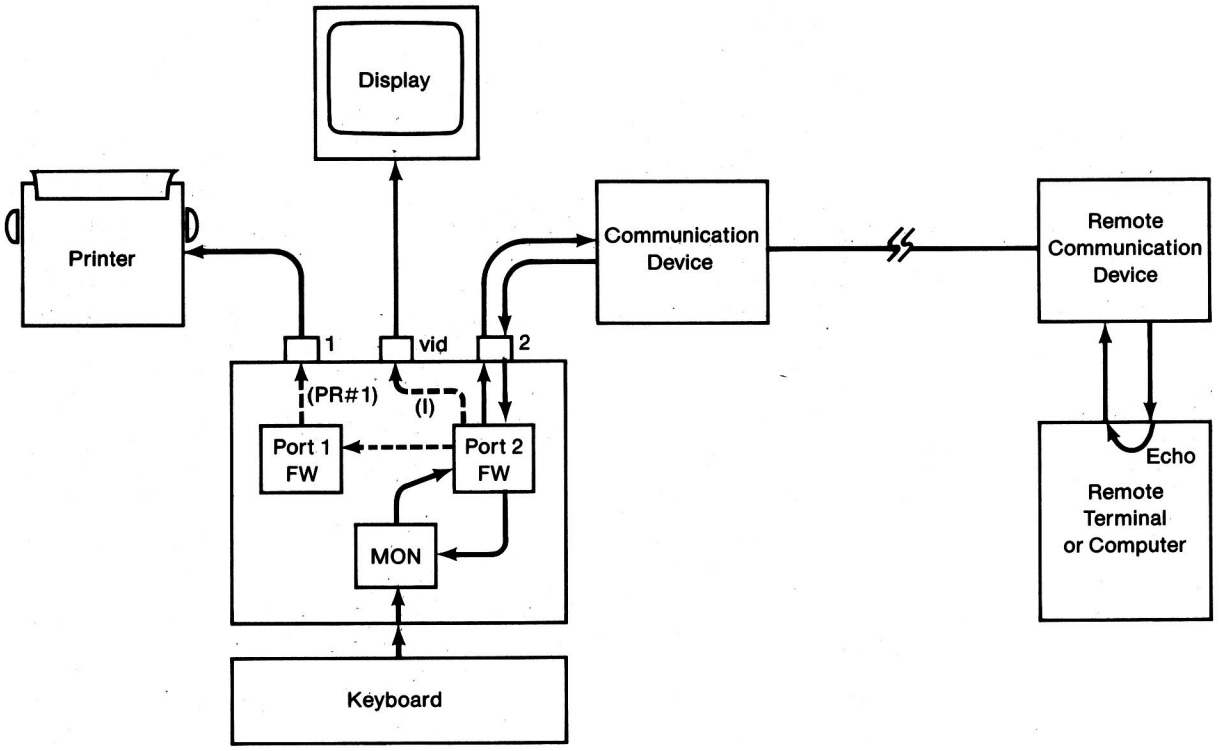


Figure 8-5
Effect of IN#2 and T command, full-duplex terminal

If your Apple IIc is the terminal in full-duplex operation, use the N command to turn off echoing input to the screen. If the Apple IIc does echo input to the screen in this setup, everything you type will appear twice: once from the Apple IIc and once from the host computer.

In this mode of operation, if you echo input to the printer you can get a printed record of both sides of the communication session: the input from the host, and the Apple IIc output as echoed by the host.

Figure 8-6 shows the flow of information when the Apple IIc is a full-duplex host. In this case, the local Apple IIc must echo input to output for the remote device. The setup commands include PR#2 in this case.

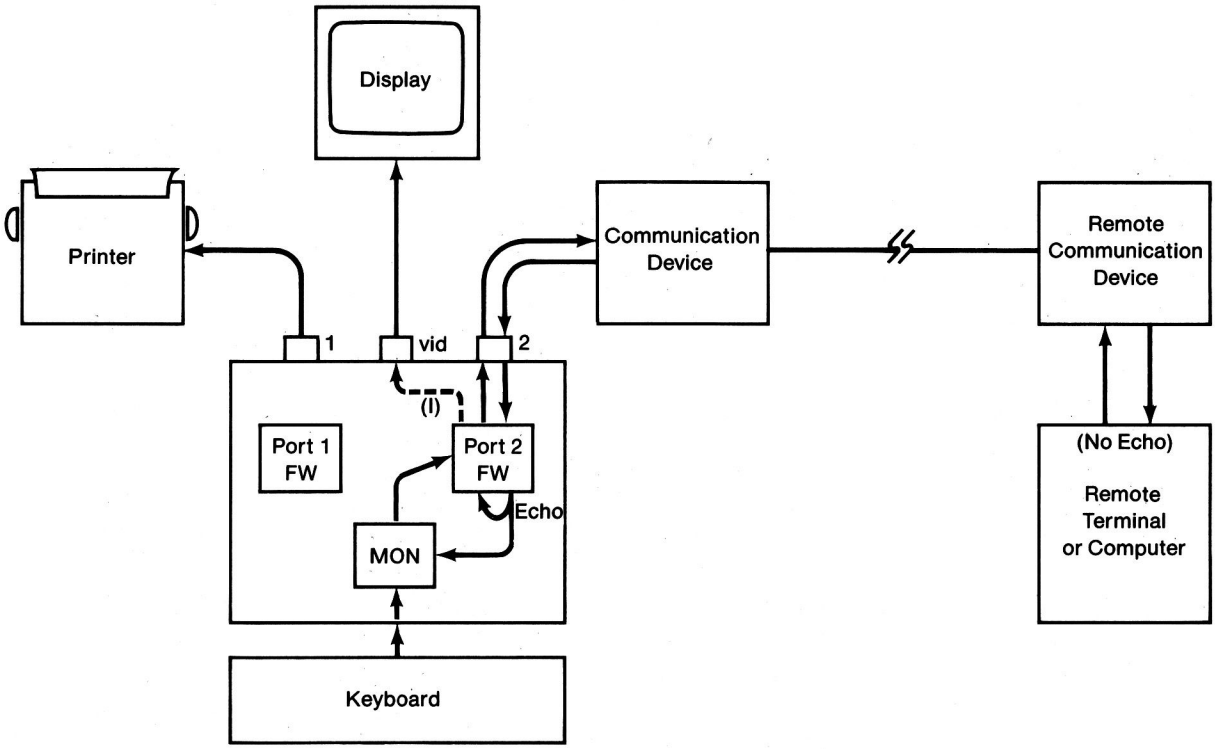


Figure 8-6
Effect of IN#2, PR#2, and T command, full-duplex host

Warning If the Apple IIc echoes input to output and the other computer does too, then the first subsequent keypress will echo back and forth endlessly and lock up the Apple IIc. This will require a Control-Reset to get out.

If you echo input to output when using an information service, the host will end up seeing the echo of what it sent you as though you had typed it.

In this arrangement, the local output hook is not available for using the printer or other device. To display keyboard and port 2 input on the screen, issue Control-A I.

Terminal mode

Terminal mode makes the Apple IIc act like a dumb terminal—one that just sends and receives information, but does not process it. Input and output flow through special serial I/O buffers on page $08 of auxiliary memory. Applesoft firmware and system software cannot see or interpret the data: only the serial port 2 firmware deals with it.

In most terminal mode setups, the firmware will not display port 2 input unless you use the Control-A I command.

Warning When using terminal mode, $0800–$08FF of auxiliary RAM is used for buffering. Any data stored there will be overwritten when terminal mode is enabled.

Control-A T turns on terminal mode, and Control-A Q turns it off.

The remote device can go into terminal mode, and then turn off the local Apple IIc's terminal mode with the Control-R command. If it then issues Control-A PR#2, local output will go to the remote device. The remote keyboard and display then become the input and output devices of the local Apple IIc processor. This is remote mode.

In remote mode, the local Apple IIc does not use the serial I/O buffers (as it does in terminal mode); therefore, local firmware and system software detect and interpret all input and output data. So, for example, if you type CATALOG at the remote device keyboard, the local Apple IIc will execute the command and list the disk catalog on the remote device's display. (In terminal mode, the local computer would simply display the word CATALOG on its screen.)

The remote device can turn the local Apple IIc's terminal mode back on with Control-T. Control-A T issued at the remote device only turns on the remote device's terminal mode, unless the command character there has already been changed to something else.

Chapter 9

Mouse and Game Input

This chapter describes the Apple IIc's mouse port and hand controller (game) input capabilities. The mouse and hand controllers use the same 9-pin connector on the back panel; the firmware uses the port as directed by keyboard or program commands.

Mouse input

Table 9-1 summarizes the mouse port's characteristics and guides you to other information in this part of the chapter.

Warning If you want your programs that use the mouse on the Apple IIe and other Apple II series computers to work with the Apple IIc, always use the I/O firmware entry points listed in Tables 9-4 and 9-5, rather than dealing directly with the mouse hardware and RAM locations.

The mouse back panel connector is described in Chapter 11.

Table 9-1
Mouse input port characteristics

Port number	Mouse input port 4.
BASIC commands	Turn on mouse: PRINT CHR$(4)"PR#4":PRINT CHR$(1) Turn off mouse interrupts: PRINT"PR#4":PRINT CHR$(0) Turn on graphics character set: See "MouseText" in Chapter 5.
Initial characteristics	After a reset, all mouse interrupts are off and the rising edge of X0 and Y0 are selected for interrupts.
Hardware page locations	See Table 9-2.
Monitor firmware routines	None.
I/O firmware entry points	See Tables 9-3 and 9-4.
Use of screen holes	See Table 9-5.

Memory expansion

The memory expansion version of the Apple IIc places the mouse at input port 7 and the memory expansion card at port 4. Thus, all "PR4" entries become "PR7" entries.

Mouse connector signals

The mouse uses the same DB-9 connector as the hand controllers. However, the interpretation of the signals arriving on the pins differs depending on the commands and signals received. Figure 11-37 shows the names of the pin assignments when a mouse is connected.

Mouse operating modes

This section tells what the mouse operating modes are for. Later sections of this chapter describe how to set the various mouse operating modes.

Your program should call the ServeMouse routine to determine the source of an interrupt as soon as it receives one, in all the interrupt modes except transparent mode.

Transparent mode

In transparent mode, your program must read screen holes to check for mouse movement. An interrupt routine in the Apple IIc firmware updates mouse position counters each time the mouse is moved, then returns control to the main program task. The findings of the interrupt routine are placed in the screen holes for your program to find. Table 9-5 lists the screen holes with the information that your program should look for.

This is the only mouse mode available to BASIC programs.

Movement interrupt mode

On the Apple IIc, a signal called *VBlInt* can interrupt the processor whenever a video vertical blanking signal occurs. This can make it easier for your programs to smoothly move the mouse cursor in response to mouse movements.

In movement interrupt mode, the mouse firmware arms VBlInt whenever the mouse is moved at least one count in any direction. When VBlInt occurs, program control passes to the vector address contained at locations $03FE and $03FF; the interrupt handler in your program can then update the cursor smoothly to its next screen position.

Your program's interrupt handler must first call ServeMouse (Table 9-3) to see if the mouse caused the interrupt. It should then call ReadMouse to get mouse status and its current X-Y position. The routine can also change the mouse mode and position if desired.

"MouseText" in Chapter 5 contains recommendations for using MouseText characters with a mouse.

The maximum amount of mouse movement that can occur between successive VBlInt interrupts is limited to the distance someone can move a mouse in one-sixtieth of a second.

Button interrupt mode

The Apple IIc mouse-button hardware location does not generate interrupts. However, a program can simulate mouse-button interrupts by polling the button whenever VBlInt occurs, and acting on the interrupt whenever the button state has changed. This lets your program provide fast response to the mouse movement without too much program overhead.

Movement/button interrupt mode

The movement/button interrupt mode is a combination of the two modes just described. It provides the best response possible without constant polling of the mouse position and button. Your program can effectively process a main task concurrently with cursor and menu updating, as well as menu-selected command processing.

Vertical blanking active modes

The vertical blanking active modes are the same as the four just described except that they allow VBL interrupts to be sent to the user.

Mouse soft switches

The soft switches assigned to the mouse interface are shown in Table 9-2. On power-up or reset, the hardware selects the rising edge of X0 and Y0 (mouse movement signals) and masks out all mouse interrupts.

Appendix E explains how the firmware handles interrupts.

Warning Table 9-2 is included here for your information only. You should use the built-in firmware to access the mouse; doing so is much easier than writing your own mouse interrupt handler and guarantees compatibility with all other Apple II-series computers.

Table 9-2
Mouse soft switches

Name	Action	Hex	Function
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*
RdIOUDis	R7	$C07E	Read IOUDis switch (1 = off)†
DisXY	R/W	$C058	Disable (mask) X0 and Y0 movement interrupts‡
EnbXY	R/W	$C059	Enable (allow) X0 and Y0 movement interrupts‡
RdXYMsk	R7	$C040	Read status of X0/Y0 interrupt mask (1 = mask on)
RstXY	R	$C048	Reset X0/Y0 interrupt flags
X0Edge	R/W	$C05	Select rising edge of X0 for interrupt‡
X0Edge	R/W	$C05D	Select falling edge of X0 for interrupt‡
RdX0Edge	R7	$C042	Read status of X0 edge selector (1 = falling)
RstXInt	R	$C015	Reset mouse X0 interrupt flag

Table 9-2 (continued)
Mouse soft switches

Name	Action	Hex	Function
Y0Edge	R/W	$C05E	Select rising edge of Y0 for interrupt‡
Y0Edge	R/W	$C05F	Select falling edge of Y0 for interrupt‡
RdY0Edge	R7	$C043	Read status of Y0 edge selector (1 = falling)
RstYInt	R	$C017	Reset mouse Y0 interrupt flag
DisVBl	R/W	$C05A	Disable (mask) VBL interrupts‡
EnVBl	R/W	$C05B	Enable (allow) VBL interrupts‡
RdVBlMsk	R7	$C041	Read status of VBL interrupt mask (1 = mask on)
RstVBl	R	$C019	Read and then reset VBlInt flag
PTrig	R/W	$C070	Reset VBlInt flag; trigger paddle timer
RdBtn0	R7	$C061	Read first mouse button status (1 = pressed)§
Rd63		R7	$C063 Read second mouse button status (0 = pressed)¶
MouX1	R7	$C066	Read status of X1 (mouse X direction) (1 = high)
MouY1	R7	$C067	Read status of Y1 (mouse Y direction) (1 = high)

* When IOUDis is on, $C058–$C05F do not affect mouse, and $C05E and $C05F become DHiRes (Table 5-8).
† Read or write to $C07x also resets VBlInt and triggers paddle timers.
‡ These work only if IOUDis is off.
§ This location is also the Open Apple key (Table 4-1).
¶ This is also the location of the Shift-key mod (Appendix F).

Mouse firmware sets interrupts in response to mode settings under program control. The vertical blanking interrupt (VBlInt) is armed if the mouse button is pushed or moves at least a count of 1 in the X0 or Y0 coordinates. Read $C070 to reset the VBL interrupt. Because a VBL occurs every sixtieth of a second, that is the maximum time that can elapse before the resulting interrupt can be acknowledged and acted upon.

Software can also select which edge of X0 and Y0 information will cause the XInt or YInt.

When an interrupt has occurred, you can read the direction of the mouse's X1 movement by reading address $C066 bit 7, and Y1 movement by reading address $C067 bit 7.

A program can read the status of the soft switches by reading one of the locations $C040–$C043 and then testing data bit 7. The soft switches are described in Table 9-2.

The section on mouse input in Chapter 11 explains what X0, Y0, X1, Y1 are and what they mean with respect to mouse movement.

If you write your own mouse interrupt handler, it should enable the main bank-switched memory, set up its own IRQ vectors at addresses $FFFE and $FFFF, keep track of video modes and the alternate stack, and check for the interrupt source in the same manner as the mouse firmware listed in Appendix I, beginning at address $C400.

Important The listing in Appendix I provides source code only for the memory expansion version of the Apple IIc. Mouse code starts at $C700 in the new ROM. There are instructions for obtaining listings for the original and UniDisk 3.5 versions in Appendix I.

UniDisk 3.5 The 32K ROM includes a new feature for programs that need to use mouse interrupts for their own purposes. If your program sets bit 7 of the mouse port mode byte at $07FC ($C7FF in the memory expansion IIc) to 1, mouse movement interrupts will be passed to the interrupt handler of your program. VBL interrupts will still be handled by the Apple IIc's firmware. You should use this feature only if the mouse firmware can't keep up with your needs.

I/O firmware support for mouse input

Memory expansion The memory expansion version of the Apple IIc places the mouse at $C700 and the memory expansion card at $C400. This means that the mouse is supported on page $C7 in the new Apple IIc, so change all $C4 and $40 addresses to $C7 and $70.

The Apple IIc supports the mouse with firmware starting at address $C400. This firmware is necessary because the mouse requires fast, transparent interrupt processing to work effectively.

In assembly language you can use direct firmware support for sophisticated mouse applications. To enable the mouse, first load a mode byte into the accumulator (and $C4 in X, $40 in Y) and then do a JSR to the firmware routine called *SetMouse* (Table 9-3). Valid mode bytes are the following:

$00	Turns mouse off
$01	Sets transparent mode
$03	Sets movement interrupt mod
$05	Sets button interrupt mode
$07	Sets movement or button interrupt mode
$08	Turns mouse off, VBlInt active
$09	Sets transparent mode, VBlInt active
$0B	Sets movement interrupt mode, VBlInt active
$0D	Sets button interrupt mode, VBlInt active
$0F	Sets movement or button interrupt mode, VBlInt active

The firmware then initializes the mouse. To read the current position and status of the mouse, first load $C4 into the X register, load $40 into the Y register, save processor status, disable interrupts, and then JSR to the firmware routine called *ReadMouse* (Table 9-3), which stores the information in the port 4 screen holes (Table 9-5).

Table 9-3 lists the mouse port firmware routine offsets. Each address contains the low byte of the entry point of the routine described. The calling setup for all routines (except ServeMouse) is the same: the X register must contain $C4, and the Y register must contain $40. When the routine has finished, the A, X, and Y register contents are undefined.

Memory expansion

The memory expansion version of the Apple IIc places the mouse at $C700 and the memory expansion card at $C400. Thus, all mouse firmware routines start at a $C7XX address, instead of $C4XX.

Table 9-3
Mouse firmware routines

Location	Offset for	Description
$C412	SetMouse	Sets the mouse mode to the value in the accumulator. Input: A register contains mode (see $07FC, Table 9-5) ($07FF in new Apple IIc). Output: Carry bit = 0 means mode was legal; carry bit = 1 means mode was not legal.
$C413	ServeMouse	Services mouse interrupt if needed. Input: X, Y, A registers—doesn't matter. Output: Carry bit = 0 means mouse caused the interrupt; carry bit = 1 means something else caused it. This routine updates $077C ($077F in new Apple IIc) to show which event caused the interrupt (values in Table 9-5).
$C414	ReadMouse	Updates screen holes to show current mouse X-Y position and button status; clears VBlInt, button and movement interrupt bits in the status byte. Doesn't reenable interrupts until after retrieving position values. Output: Carry bit = 0.
$C415	ClearMouse	Sets the mouse position to 0, though not necessarily within clamping boundaries; leaves button and interrupt bits in status byte unchanged. Output: Carry bit = 0.
$C416	PosMouse	Sets the mouse coordinates to new values. Input: X and Y screen holes contain new X and Y positions. Output: Carry bit = 0.

Table 9-3 (continued)
Mouse firmware routines

Location	Offset for	Description
$C417	ClampMouse	Sets new clamping boundaries (see Table 9-5). Does not affect mouse position or update mouse position screen holes; use ReadMouse to do that. Input: A register = 0 means set new X boundaries; A register = 1 means set new Y boundaries. Output: Carry bit = 0.
$C418	HomeMouse	Sets the internal mouse position to the upper-left corner of the clamping window. Does not update mouse position screen holes; use ReadMouse to do that.
$C419	InitMouse	Sets startup internal values; does not update mouse-position screen holes. Output: Carry bit = 0.

Here is a sample sequence of events and calls:

1. Four screen holes contain the mouse's X and Y coordinates, and one contains the status of the last mouse movement (Table 9-5).
2. Call InitMouse.
3. Inhibit interrupts, set up the boundaries you want, then call ClampMouse.
4. Use PosMouse, HomeMouse, or ClearMouse to position the mouse where you want it.
5. Put the mouse mode (see address $07FC in Table 9-5) that you want to use in the accumulator, then call SetMouse (use address $07FF for the new Apple IIc).
6. If you have set one of the interrupt modes, then when an interrupt arrives, call ServeMouse to determine the source of the interrupt.
7. Disable interrupts and call ReadMouse. Retrieve the position values, then reenable interrupts.

Pascal support

Table 9-4 lists the locations and values of the I/O firmware protocol that Pascal 1.1 and later versions use. However, Pascal must use a special attach driver to support the mouse.

Memory expansion

The memory expansion version of the Apple IIc places the mouse at $C700 and the memory expansion card at $C400. Thus, all mouse firmware routines start at a $C7XX address, instead of $C4XX.

Table 9-4
Mouse port I/O firmware protocol

Address	Value	Description
$C405	$38	Pascal ID byte
$C407	$18	Pascal ID byte
$C40B	$01	Generic signature byte of firmware cards
$C40C	$20	2 = X-Y pointing device; 0 = identification code
$C40D		Initialization routine (not implemented; returns error code)
$C40E		Standard read routine (not implemented; returns error code)
$C40F		Standard write routine (not implemented; returns error code)
$C410		Standard status routine (not implemented; returns error code)
$C411	$00	Optional routines follow
$C4FB	$D6	A mouse identification byte

BASIC and assembly-language support

Memory expansion

The memory expansion version of the Apple IIc places the mouse at $C700 and the memory expansion card at $C400. This means that all "PR4" or "IN4" calls change to "PR7" or "IN7" calls.

In BASIC you must turn the mouse on by printing PR#4 and then CHR$(1) before you can get input from the mouse. This sets transparent mode. After that, reenable video output with PR#3 and take subsequent input from the mouse by issuing IN#4. The first input statement after that (INPUT X,Y,S) initializes and enables the mouse and returns a three-element string

```
+xxxx,+yyyy,+st
```

representing the x-coordinate, y-coordinate, and status digits.

The coordinates will be integers between 0 and +1023. These are called the clamping boundaries of the mouse.

The sign preceding the status digits is normally positive; it becomes negative when you press a key on the keyboard.

The first digit, s, of the status is 0. The second digit, t, of the status is 1 if the mouse button is still pressed, 2 if it was just pressed, 3 if it was just released, and 4 if it is still released.

To disable the mouse, use these statements:

```
PRINT CHR$(4)"PR#4"
PRINT CHR$(0)
PRINT CHR(4)"PR#3"
```

Important Change all 4's to 7's for the memory expansion version.

Screen holes

Table 9-5 lists the screen holes that the mouse firmware uses. Note that the mouse firmware reserves port 5 screen holes for its own use. Also, the auxiliary page counterparts of the port 4 addresses are reserved for startup values.

Important Some screen holes are different for the Apple IIe mouse. Refer to Appendix F.

Table 9-5
Mouse port screen hole locations

Scratch area

Location	Description
$0478	Low byte of clamping minimum
$04F8	Low byte of clamping maximum
$0578	High byte of clamping mimimum
$05F8	High byte of clamping maximum

Port 4 screen holes

Location	Description
$047C	Low byte of X coordinate
$04FC	Low byte of Y coordinate
$057C	High byte of X coordinate
$05FC	High byte of Y coordinate
$067C	Reserved
$06FC	Reserved
$077C	Status byte

Bit	1 Equals
7	Button down
6	Button was down on last read and still down
5	Movement since last read
4	Reserved
3	Interrupt from VBlInt
2	Interrupt from button
1	Interrupt from movement
0	Reserved

$07FC Mode byte (current mode; mask out bits 4–7 when testing)

Bit	1 Equals
7-4	Reserved
3	VBlInt active
2	VBL interrupt on button
1	VBL interrupt on movement
0	Mouse active

Port 5 screen holes

Reserved

Memory expansion The screen hole addresses for the mouse in the Apple IIc that supports the memory expansion card all end with F, instead of C; this is because the mouse has moved to slot 7 from slot 4. For example, the low byte of the X coordinate is stored at $047C in the original and UniDisk 3.5 IIc's, while it is stored at $047F in the memory expansion IIc.

Using the mouse as a hand controller

You can use the mouse as if it were a set of hand controllers or an X-Y pointing device in port 4. If you turn the mouse on, the Monitor hand controller (game paddle) routines take input from the mouse. This is possible because the mouse and the hand controllers all use the same back panel connector.

You can run a BASIC program that uses the Pdl function to read from the mouse by doing the following, either from the keyboard or from a program:

1. Start up the system with the BASIC program that uses paddles.
2. Type `PR#4` and press Return to turn on the mouse.
3. Press Control-A Return to initialize the mouse.
4. Type `PR#0` and press Return to restore output to the screen.
5. Run the program.

Play the game using the mouse instead of the paddles.

Important Many copy-protected games do not work with a mouse. In addition, many games don't use built-in firmware for the paddles.

Game input

The Apple IIc supports game paddles, joysticks, and other hand controllers connected to the DB-9 connector on its back panel. Table 9-6 is a summary of game input characteristics.

Table 9-6
Game input characteristics

Port number	None.
Commands	None.
Initial characteristics	Game inputs cannot be disabled.

Hardware page locations

$C061	Switch input 0 and Open Apple.
$C062	Switch input 1 and Solid Apple.
$C063	Mouse button (sense is opposite that of $C061 to distinguish it from paddle button).
$C064	Analog input (paddle) 0.
$C065	Analog input (paddle) 1.
$C070	Trigger paddle timer.

Monitor firmware routines

Location	Name	Description
$FB1E	PRead	Reads a paddle position.

I/O firmware entry points

None.

Use of screen holes

None.

The hand controller connector signals

Several inputs are available to programs or devices from the 9-pin D-type miniature connector on the back of the Apple IIc: two 1-bit inputs, or *switches,* and two analog inputs.

Complete electrical specifications of these inputs are given in Chapter 11; Table 11-22 shows the connector pin numbers.

When you connect a pair of hand controllers to the 9-pin connector, the rotary controllers use two analog inputs, and the pushbuttons use two 1-bit inputs. However, you can also use these inputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick.

Switch inputs (Sw0 and Sw1)

The two 1-bit inputs can be connected to the output of another electronic device that meets the electrical requirements described in Chapter 11, 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 can 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 $C061, $C062, and $C063 (decimal locations 49249 through 49251), as shown in Table 9-6. Switch 0 and switch 1 are permanently connected to Open Apple and Solid Apple on the keyboard; these are the ones connected to the buttons on the hand controllers. Location $C063 is a second address for the mouse button, so that a program can distinguish it from an Open Apple keypress. When the mouse button is pressed, $C063 (bit 7) goes from 1 to 0, and $C061 (bit 7) goes from 0 to 1.

Analog inputs (Pdl0 and Pdl1)

The two analog inputs are designed for use with 150-KΩ 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.

A program must first reset the timing circuits before it can read the analog inputs. Accessing memory location $C070 does this. As soon as you reset the timing circuits, the high bits of the bytes at locations $C064 through $C067 are set to 1. If you PEEK at them from BASIC (locations 49252 through 49255), the values will be 128 or greater. Within about three 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 amount of time each of the 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.

Monitor support for game input

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 PRead. BASIC and other high-level languages also include convenient means of reading the analog inputs—refer to your language manuals. You can read and reread the same paddle at arbitrarily short intervals. However, you must wait at least three milliseconds between reading one paddle and reading a different paddle.

The Monitor routine PRead (at address $FB1E) places in the Y register a number between $00 and $FF that represents the position of a hand controller. You pass the number of the hand controller in the X register.

Warning If the hand controller number you furnish in the X register does not equal 0 or 1, strange things may happen.

The paddle and vertical blanking both use $C070. Disable interrupts before calling PRead if you are reading the paddles and using VBL interrupts.

Chapter 10

Using the Monitor

The System Monitor is a set of subroutines in the Apple IIc firmware that provides a standard interface to the built-in I/O devices described in Chapter 1. Many of the I/O subroutines described in Chapters 3 through 9 are part of the System Monitor.

DOS (but not ProDOS) and the BASIC interpreters (Appendix E) use these subroutines by direct calls to their starting locations. You can call the standard subroutines from your programs in the same fashion. The starting addresses for all of the standard subroutines are listed in Appendix C.

You can perform most of the Monitor functions directly from the keyboard. This chapter tells you how to use the Monitor

- to look at one or more memory locations
- to change the contents of any location
- to write small programs in machine language to be executed directly by the Apple IIc
- to move and compare blocks of memory
- to invoke other programs from the Monitor

Invoking the Monitor

The System Monitor starts at memory location $FF69 (–151). To invoke the Monitor, you make a CALL –151 statement to this location from the keyboard or from a BASIC program. When the Monitor is running, its prompting character, an asterisk (*), appears on the left side of the display screen, followed by a cursor.

The positive and negative decimal equivalents of Monitor locations are listed in Appendix C. In addition, Appendix H contains conversion tables from one numbering system to another. Appendix E gives further details on how to use Apple IIc firmware from BASIC programs.

To use the Monitor, you type commands at the keyboard. When you have finished using the Monitor, you return to the BASIC language you were previously using by pressing Control-Reset, by pressing Control-C and then Return, or by typing 3D0G, which executes the resident program—usually Applesoft—whose address is stored in a jump instruction at location $03D0.

Important If ProDOS (or DOS) is connected via the standard I/O links (Chapter 3), then you can issue commands to it from the Monitor. Under this arrangement, errors will return control to BASIC rather than to the Monitor.

If you want to have Control-Reset return you to the Monitor, load the values $69, $FF, and $5A (decimal 105, 255, and 90) into the three locations starting at address $03F2 (decimal 1010, the reset-vector address and the power-up byte).

Syntax of Monitor commands

To give a command to the Monitor, you type a line on the keyboard, then press Return. The Monitor accepts the line using the standard I/O subroutine GetLn described in Chapter 3. A Monitor command can be up to 255 characters in length, ending with a carriage return. It can include three kinds of information: addresses, data values, and command characters. You type addresses and data values in hexadecimal notation.

When the command you type calls for an address, the Monitor accepts any group of hexadecimal digits. If there are fewer than four digits in the group, it adds leading 0's; if there are more than four hexadecimal digits, the Monitor uses only the last four digits. It follows a similar procedure when the command syntax calls for two-digit data values.

Each command you type consists of one command character, usually the first letter of the command name. The Monitor recognizes 22 different command characters. Some of them are punctuation marks, some are letters (uppercase or lowercase), and some are control characters.

❖ *Note:* Although the Monitor recognizes and interprets them, control characters typed on an input line do not appear on the screen.

This chapter contains examples of Monitor command use. Some of the data values displayed by your Apple IIc may differ from the values printed in these examples, because they are variables stored in RAM.

Monitor memory commands

When you use the Monitor to examine and change the contents of memory, it keeps track of the address of the last location whose value you inquired about and the address of the location that is next to have its value changed. These are called the *last opened location* and the *next changeable location.*

Warning Because locations $C000 through $C0FF contain special hardware circuits, issuing any command that reads or writes on this page can have unpredictable, and perhaps disastrous, results.

Examining memory contents

When you type the address of a memory location and press Return, the Monitor responds with the address you typed, a dash, a space, and the value stored at that location, like this:

```
*E000
E000- 4C
*33
0033- AA
*
```

Each time the Monitor displays the value stored at a location, it saves that address as the last opened location and as the next changeable location.

Memory dump

When you type a period (.) followed by an address, and then press Return, the Monitor displays a memory dump: the data values stored at all the memory locations from the one following the last opened location to the location whose address you typed following the period. The Monitor saves the last location displayed as both the last opened location and the next changeable location. In these examples, the amount of data displayed by the Monitor depends on how much larger the address after the period is than the last opened location.

```
*20
0020- 00
*.2B
0021- 28 00 18 0F 0C 00 00
0028- A8 06 D0 07
*300
0300- 99
*.315
0301- B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03
*.32A
0316- 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20
*
```

When the Monitor performs a memory dump, it starts at the address immediately following the last opened location and displays that address and the data value stored there. It then displays the values of successive locations up to and including the location whose address you typed, but only up to eight values on a line. When it reaches a location whose address is a multiple of 8—that is, one that ends with an 8 or a 0—it displays that address as the beginning of a new line, then continues displaying more values.

After the Monitor has displayed the value at the location whose address you specified in the command, it stops the memory dump and sets that location as both the last opened location and the next changeable location. If the address specified on the input line is less than the address of the last opened location, the Monitor displays only the address and value of the location following the last opened location.

You can combine the two commands, opening a location and dumping memory, by concatenating them: type the first address, a period, and the second address. This combination of two addresses separated by a period is called a *memory range*.

```
*300.32F
0300- 99 B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20 46 03 A5 3D 4D
*30.40
0030- AA 00 FF AA 05 C2 05 C2
0038- 1B FD D0 03 3C 00 40 00
0040- 30
*E015.E025
E015- 4C ED FD
E018- A9 20 C5 24 B0 0C A9 8D
E020- A0 07 20 ED FD A9
*
```

Pressing Return by itself makes the Monitor display one line of a memory dump; that is, a memory dump from the location following the last opened location to the next multiple-of-eight boundary. The Monitor saves the address of the last location displayed as both the last opened location and the next changeable location.

```
*5
0005- 00
*Return
00 00
*Return
0008- 00 00 00 00 00 00 00 00
*32
0032- FF
*Return
AA 00 C2 05 C2
*Return
0038- 1B FD D0 03 3C 00 3F 00
*
```

Changing memory contents

The section on memory dumping showed you how to *display* values stored in the Apple IIc's memory; this section shows you how to *change* these values. You can change any location in RAM; you can change the characteristics and treatment of an output device by changing the contents of locations assigned to it; and you can change a soft switch setting by referencing its set and reset addresses.

Warning Use these commands carefully. If you change the zero-page locations used by the Interpreter or operating system (Appendix B), you may lose programs or data stored in memory.

Changing one byte

The previous commands keep track of the next changeable location; these commands make use of it. In the next example, you open location 0, then type a colon followed by a value.

```
*0
0000- 4C
*:5F
```

The contents of the next changeable location have just been changed to the value you typed, as you can see by examining that location:

```
*0
0000- 5F
*
```

You can also combine opening and changing into one operation by typing an address followed by a colon and a value. In the next example, you type the address again to verify the change.

```
*302:42
*302
0302- 42
*
```

When you change the contents of a location, the value that was contained in that location is replaced by the new value, which will remain until you or some program replaces it with another value.

- *ASCII input mode:* The Monitor has a tool to make entering values a little easier: ASCII input mode. ASCII input mode lets you enter ASCII characters as well as their hexadecimal ASCII equivalents. This means that 'A is the same as C1 and 'B is the same as C2 to the Monitor. The ASCII value for *any* character following an apostrophe is used by the Monitor. For example, to enter the string "Good morning!" at $0300 in memory, type

  ```
  *300:'G 'o 'o 'd ' 'm 'o 'r 'n 'i 'n 'g '!
  ```

  Note that each character to be placed in memory is delimited by a leading and a trailing space. The only exception to this rule is that the last character in the line is followed by a return character instead of a space.

Changing consecutive locations

You don't have to type a separate command with an address, a colon, a value, and press Return for each location you want to change. You can change the values of up to 85 consecutive locations at a time—or even more, if you omit leading 0's from the values—by typing only the initial address and colon followed by all the values separated by spaces; end with Return. The Monitor stores the consecutive values in consecutive locations, starting at the location whose address you typed. After it has processed the string of values, it takes the location following the last changed location as the next changeable location. Thus, you can continue changing consecutive locations, without typing an address on the next input line, by typing another colon and more values.

In these examples, you first change some locations, then examine them to verify the changes.

```
*300:69 01 20 ED FD 4C 03
*300
0300- 69
*Return
01 20 ED FD 4C 00 03
*10:0 1 2 3
*:4 5 6 7
*10.17
0010- 00 01 02 03 04 05 06 07
*
```

Moving data in memory

You can copy a contiguous block of data from one area in the Apple IIc's memory to another in RAM by using the Monitor's MOVE command. To move a range of memory, you must tell the Monitor both where the data is now situated in memory—the source locations—and where you want the copy to go—the destination locations.

The format of the complete MOVE command looks like this:

{destination} < *{start}* . *{end}* M

The destination is the address where you want the first of the moved data to go. The less-than symbol (<) separates the destination address from the starting and ending addresses of the block of data to be moved. The period between two addresses is the Monitor's standard notation for specifying address ranges. If the second address in the source range specification is less than the first, then only one value (that of the first location in the range) will be moved.

When you type the actual command, replace the words in braces with hexadecimal addresses, and omit the braces and spaces.

Here are some examples of memory moves. First, you examine the values stored in one range of memory, then store several values in another range of memory. The actual MOVE commands end with M.

```
*0.F
0000- 5F 00 05 07 00 00 00 00
0008- 00 00 00 00 00 00 00 00
*300:A9 8D 20 ED FD A9 45 20 DA FD 4C 00 03
*300.30C
0300- A9 8D 20 ED FD A9 45 20
0308- DA FD 4C 00 03
*0<300.30C M
*0.C
0000- A9 8D 20 ED FD A9 45 20
0008- DA FD 4C 00 03
*310<8.A M
*310.312
0310- DA FD 4C
*2<7.9 M
*0.C
0000- A9 8D 20 DA FD A9 45 20
0008- DA FD 4C 00 03
*
```

The Monitor moves a copy of the data stored in the source range of locations to the destination locations. The values in the source range are left undisturbed. The Monitor remembers the last location in the source range as the last opened location, and the first location in the source range as the next changeable location.

See "Advanced Operations" for an interesting application of this feature.

If the destination address of the MOVE command is inside the source range of addresses, then strange things happen: the locations between the beginning of the source range and the destination address are treated as a subrange and the values in this subrange are replicated throughout the source range. Try it.

Comparing data in memory

You can use the VERIFY command to compare two ranges of memory using the same format you use to move a range of memory from one place to another. In fact, the VERIFY command can be used immediately after a MOVE to make sure that the move was successful.

The VERIFY command, like the MOVE command, needs a range and a destination. The syntax of the VERIFY command is

{destination} < *{start}* . *{end}* V

The Monitor compares the values in the source locations with the values in the locations beginning at the destination address. If any values don't match, the Monitor displays the address at which the discrepancy was found and the two values that differ. In the example, you store data values in the range of locations from 0 to $0D, copy them to locations starting at $0300 with the MOVE command, and then compare them using the VERIFY command. When you use the VERIFY command after you change the value at location 6 to $E4, it detects the change.

```
*0:D7 F2 E9 F4 F4 E5 EE A0 E2 F9 A0 C3 C4 C5
*300<0.D M
*300<0.D V
*6:E4
*300<0.D V
0006-E4 (EE)
*
```

If the VERIFY command finds a discrepancy, it displays the address of the location in the source range whose value differs from its counterpart in the destination range. If there is no discrepancy, VERIFY displays nothing. The VERIFY command leaves the values in both ranges unchanged. The last opened location is the last location in the source range, and the next changeable location is the first location in the source range, just as in the MOVE command. If the ending address of the range is less than the starting address, the values of only the first locations in the ranges are compared. Like the MOVE command, the VERIFY command does unusual things if the destination address is within the source range; see "Advanced Operations" later in this chapter.

Monitor register commands

Even though the actual contents of the 65C02's internal registers are changing as you use the Monitor, you can examine the values that the registers contained at the time the Monitor gained control, either because you called it or because the program you are debugging stopped at a break (**BRK**). You can also store new register values that will be used when you execute a program from the Monitor using the GO command, described below.

Changing registers

When you call the Monitor, it stores the contents of the 65C02 registers in memory. The registers are stored in the order A, X, Y, P (processor status register), and S (stack pointer), starting at location $45. When you give the Monitor a GO command, the Monitor loads the registers from these five locations before it executes the first instruction in your program.

Examining registers

Pressing Control-E and then Return invokes the Monitor's EXAMINE command, which displays the stored register values and sets the location containing the contents of the A register as the next changeable location. After using the EXAMINE command, you can change the values in these locations by typing a colon and then typing the new values separated by spaces. In the following example, you display the registers, change the first two, and then display them again to verify the change.

```
*Control-E
M=00 A=0A X=FF Y=D8 P=B0 S=F8
*:B0 02
*Control-E
M=00 A=B0 X=02 Y=D8 P=B0 S=F8
*
```

In the EXAMINE command's display, M shows the current memory state register contents. The memory state register is location $44, and its interpretation is given in Appendix E.

Miscellaneous Monitor commands

Monitor commands discussed in this section let you do the following:

- change the video display format from normal to inverse and back
- assign input and output to various devices
- leave the Monitor and return to the currently loaded operating system (DOS 3.3 or ProDOS) or BASIC

Display inverse and normal

The COut subroutine is described in Chapter 3.

You can control the setting of the inverse-normal mask location used by the COut subroutine from the Monitor so that all the Monitor's output will be in inverse format. The INVERSE command (I) sets the mask so that all subsequent inputs and outputs are displayed in inverse format. To switch the Monitor's output back to normal format, use the NORMAL command (N).

```
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*I
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*N
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*
```

Back to BASIC

See Appendix D.

If you are using one of the Apple disk operating systems (ProDOS or DOS), press Control-Reset or type 3D0G to return to the language you were using, with your program and variables intact.

Important If you type 3D0G, make sure that the third character you type is a *zero*, not a letter *O*. The letter *G* is the Monitor's GO command.

If there is no operating system in RAM, use the BASIC command Control-B to leave the Monitor and enter the BASIC interpreter that was active when you entered the Monitor. (Normally this is Applesoft BASIC.) Any program or variables that you previously had in BASIC will be lost. If you want to reenter BASIC with your previous program and variables intact, use the CONTINUE BASIC command (Control-C).

Redirecting input and output

Chapter 3 lists the Apple IIc port numbers available.

The Control-P command diverts all output normally destined for the screen (port 0) to a device attached to one of the other ports, from 1 to 7. The format of the command is

{*port number*} Control-P

A Control-P command to port number 0 switches the stream of output characters back to the Apple IIc's video display. However, use Escape Control-Q if the enhanced video firmware is active (solid-block cursor).

Control-K controls the input stream in much the same way that Control-P controls the output stream. The format for the command is

{*port number*} Control-K

Pressing O Control-K directs the Monitor to accept input from the Apple IIc's built-in keyboard.

For more information on the way those commands work, refer to "The Standard I/O Links" in Chapter 3.

The Control-P and Control-K commands are the exact equivalents of the BASIC (but not DOS and ProDOS) commands PR# and IN#.

Hexadecimal arithmetic

You can use the Monitor as a one-byte hexadecimal addition and subtraction calculator. Just type a line in one of these formats followed by Return:

{*value*} + {*value*} Return {*value*} – {*value*} Return

The Apple IIc performs the arithmetic and displays the result, as shown in these examples.

```
*20+13
=33
*4A-C
=3E
*
```

Advanced operations

This section describes some ways of using the Monitor commands to speed up your work.

Multiple-command lines

You can put as many Monitor commands on a single line as you like, as long as you separate them with spaces, and the total number of characters in the line is less than 254. Adjacent single-letter commands such as L, S, I, and N need not be separated by spaces.

You can freely intermix all of the commands except the STORE (:) command. Because the Monitor takes all values following a colon and places them in consecutive memory locations, the last value in a STORE must be followed by a letter command before another address is encountered. You can use the NORMAL command as the required letter command in such cases; it usually has no effect and can be used anywhere.

In the following example, you display a range of memory, change it, and display it again, all with one line of commands.

```
*300.307 300:18 69 1 N 300.302
0300- 00 00 00 00 00 00 00 00
0300- 18 69 01
*
```

If the Monitor encounters a character in the input line that it does not recognize as either a hexadecimal digit or a valid command character, it executes all the commands on the input line up to that character, then stops with a beep and ignores the remainder of the input line.

Filling memory

The MOVE command can be used to replicate a pattern of values throughout a range of memory. To do this, first store the pattern in the first locations in the range

```
*300:11 22 33
*
```

Remember the number of values in the pattern: in this case, it is three. Use the number to compute addresses for the MOVE command, like this:

{start+number} < {start} . {end-number} M

This MOVE command first replicates the pattern at the locations immediately following the original pattern, then replicates that pattern following itself, and so on until it fills the entire range.

```
*303<300.32D M
*300.32F
0300- 11 22 33 11 22 33 11 22
0308- 33 11 22 33 11 22 33 11
0310- 22 33 11 22 33 11 22 33
0318- 11 22 33 11 22 33 11 22
0320- 33 11 22 33 11 22 33 11
0328- 22 33 11 22 33 11 22 33
*
```

You can use the VERIFY command to check whether a pattern repeats itself through memory. This is especially useful to verify that a given range of memory locations all contain the same value. In this example, to see the VERIFY command detect the discrepancy, you first fill the memory range from $0300 to $0320 with 0's and verify it, then change one location and verify again:

```
*300:0
*301<300.31F M
*301<300.31F V
*304:02
*301<300.31F V
0303-00 (02)
0304-02 (00)
*
```

Repeating commands

You can create a command line that repeats one or more commands over and over. You do this by beginning the part of the command line that you want to repeat with a letter command, such as N, and ending it with the sequence 34:n, where n is a hexadecimal number that specifies the position in the line of the command where you want to start repeating; for the first character in the line, n=0. The value for n must be followed with a space in order for the loop to work properly.

This trick takes advantage of the fact that the Monitor uses an index register to step through the input buffer, starting at location $0200. Each time the Monitor executes a command, it stores the value of the index at location $34; when that command is finished, the Monitor reloads the index register with the value at location $34. By making the last command change the value at location $34, you change this index so that the Monitor picks up the next command character from an earlier point in the buffer.

The only way to stop a loop like this is to press Control-Reset; that is how this example ends.

```
*N 300 302 34:0 N
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
030
*
```

Creating your own commands

The USER command (Control-Y) forces the Monitor to jump to memory location $03F8. You can put a JMP instruction there that jumps to your own machine-language program. Your program can then examine the Monitor's registers and pointers or the input buffer itself to obtain its data. For example, here is a program that displays everything on the input line after the Control-Y. The program starts at location $0300; the command line that starts with $03F8 stores a jump to $0300 at location $03F8.

```
*300:A4 34 B9 00 02 20 ED FD C8 C9 8D D0 F5 4C 69 FF
*3F8:4C 00 03
*Control-Y THIS IS A TEST
THIS IS A TEST
*
```

Machine-language programs

The main reason to program in machine language is to get more speed and sometimes to also save memory space. A program in machine language can run much faster than the same program written in high-level languages such as BASIC or Pascal, but the machine-language version usually takes a lot longer to write. There are other reasons to use machine language: you might want your program to do something that isn't included in your high-level language, or you might just enjoy the challenge of using machine language to work directly on the bits and bytes.

❖ *Note:* If you have never used machine language before, you'll need to learn the 65C02 instructions listed in Appendix A. To become proficient at programming in machine language, you'll have to spend some time at it, and study one of the books on 65C02 programming listed in the bibliography.

You can get a hexadecimal dump of your program or move it around in memory using the commands described in the previous sections. The Monitor commands in this section are intended specifically for you to use in creating, writing, and debugging machine-language programs.

Running a program

The Monitor command to start execution of your machine-language program is the GO command. When you type an address and press G, the Apple IIc starts executing machine-language instructions starting at the specified location. If you just press G, execution starts at the last opened location. The Monitor treats this program as a subroutine: it should end with an RTS (return from subroutine) instruction to transfer control back to the Monitor.

The Monitor has some special features that make it easier for you to write and debug machine-language programs, but before you get into that, here is a small machine-language program that you can run using only the simple Monitor commands already described. The program in the example merely displays the letters *A* through *Z:* you store it starting at location $0300, examine it to be sure you typed it correctly, then type 3D0G to start it running.

```
*300:A9 C1 20 ED FD 18 69 1 C9 DB D0 F6 60
*300.30C
0300- A9 C1 20 ED FD 18 69 01
0308- C9 DB D0 F6 60
*300 G
ABCDEFGHIJKLMNOPQRSTUVWXYZ
*
```

Disassembled programs

Machine-language code in hexadecimal isn't the easiest thing in the world to read and understand. To make this job a little easier, machine-language programs are usually written in assembly language and converted into machine-language code by programs called **assemblers.**

Programs like the Monitor's LIST command are called **disassemblers.** This command displays machine-language code in assembly-language form. Instead of unformatted hexadecimal gibberish, the LIST command displays each instruction on a separate line, with a three-letter instruction name, or mnemonic, and a formatted hexadecimal operand. The LIST command also converts the relative addresses used in branch instructions to absolute addresses.

The Monitor LIST command has the format

{*location*} L

The LIST command starts at the specified location and displays as much memory as it takes to make up a screenful (20 lines) of instructions, as shown in the following example:

```
*300 L
0300-       A9 C1          LDA    #$C1
0302-       20 ED FD       JSR    $FDED
0305-       18             CLC
0306-       69 01          ADC    #$01
0308-       C9 DB          CMP    #$DB
030A-       D0 F6          BNE    $0302
030C-       60             RTS
030D-       00             BRK
030E-       00             BRK
030F-       00             BRK
0310-       00             BRK
0311-       00             BRK
0312-       00             BRK
0313-       00             BRK
0314-       00             BRK
0315-       00             BRK
0316-       00             BRK
0317-       00             BRK
0318-       00             BRK
0319-       00             BRK
*
```

The first seven lines of this example are the assembly-language form of the program you typed in the previous example. The rest of the lines are BRK instructions only if this part of memory has 0's in it: other values will be disassembled as other instructions.

The Monitor saves the address that you specify in the LIST command, but not as the last opened location used by the other commands. Instead, the Monitor saves this address as the program counter, which it uses only to point to locations within programs. Whenever the Monitor performs a LIST command, it sets the program counter to point to the location immediately following the last location displayed on the screen, so that if you type another LIST command it displays another screenful of instructions, starting where the previous display left off.

The STEP and TRACE commands

Important This section applies only to the UniDisk 3.5 and memory expansion versions of the Apple IIc.

STEP and TRACE are Monitor facilities for debugging assembly-language programs. The STEP command decodes, displays, and executes one instruction at a time, and the TRACE command steps continuously through a program, stopping when a BRK instruction is executed or Solid Apple is pressed. You can press Open Apple to slow down the trace to one step per second.

Each STEP command causes the Monitor to execute the instruction in memory pointed to by the program counter. The instruction is displayed in its disassembled form, then executed. The contents of the 65C02's internal registers are displayed after the instruction is executed. After execution, the program counter is incremented to point to the next instruction in the program.

Here is an example of the STEP command, using the following program:

```
$0300:   LDX #02
$0302:   LDA $00,X
$0304:   STA $10,X
$0306:   DEX
$0307:   STA $C030
$030A:   BPL $0302
$030C:   BRK
```

To step through this program, first call the Monitor by typing `CALL -151` and pressing Return, and then from the Monitor type `300S` (to start the STEP routine at address $0300). Type `S` to advance each additional step through the program. The Monitor keeps the program counter and the last opened address separate from one another, so you can examine or change the contents of memory while you are stepping through your program. Here's what happens when you step through the program above, examining the contents of location $0012 after the third step. Note that in this example, what you type appears just after the * prompt, and the information on the next two lines—that begin without the * prompt—is what the computer displays on the screen in response.

```
*300S
0300-   A2 02    LDX #02
M=CA A=0A X=02 Y=D8 P=30 S=F8
*S
0302-   B5 00    LDA $00,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
*S
0304-   95 10    STA $10,X
M=CA A=0C X=02 Y=D8 P=30 S=F8
*12
0012- 0C
*S
0306-   CA       DEX
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
0307-   8D 30 C0 STA $C030
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
030A-   10 F6    BPL $0302
M=CA A=0C X=01 Y=D8 P=30 S=F8
*S
0302-   B5 00    LDA $00,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*S
0304-   95 10    STA $10,X
M=CA A=0B X=01 Y=D8 P=30 S=F8
*
```

The TRACE command is a continuous version of the STEP command; it stops stepping through the program only when you press Solid Apple, or when it encounters a BRK instruction in the program. Press Open Apple to slow the trace to one step per second.

Important Keep the following cautions in mind when using the STEP and TRACE Monitor commands:

- If the program ends with an RTS instruction, the TRACE routine will continue to run indefinitely until stopped with Solid Apple.
- You can't step or trace through routines that use the same zero page locations as the Monitor.

The Mini-Assembler

Important This section applies only to the UniDisk 3.5 and memory expansion versions of the Apple IIc.

Without an assembler, you have to write your machine-language program, take the hexadecimal values for the opcodes and operands, and store them in memory using the Monitor commands described earlier in this chapter.

The Mini-Assembler lets you enter machine-language programs directly from the keyboard of your Apple. ASCII characters can be entered in Mini-Assembler programs, exactly as you enter them in the Monitor.

Note that the Mini-Assembler doesn't accept labels; you must use actual values and addresses.

Starting the Mini-Assembler

To start the Mini-Assembler, first invoke the Monitor by typing `CALL -151` and pressing Return, and then from the Monitor, type ! followed by Return. The Monitor prompt character then changes from * to !.

When you finish using the Mini-Assembler, press Return from a blank line to return to the Monitor.

To enter code into memory, type the address, a colon, and the instruction. For example, after entering the Mini-Assembler, you could type

```
!300:STA C030
```

You can enter a series of instructions by typing a space, followed by the instruction, followed by Return:

```
!300:STA C030
! LDA #A0
! INX
```

Each succeeding instruction is placed in the next available memory location. As you type in instructions, each is replaced by the starting address of the instruction, the hexadecimal value(s) of the instruction, followed by mnemonics describing the instruction. For example, the sequence of instructions given above would produce the following on your screen:

```
0300-   8D 30 C0        STA $C030
0303-   A9 A0           LDA #$A0
0305-   E8              INX
```

When you're ready to execute your program, press Return to leave the Mini-Assembler and return to the Monitor. Monitor commands can't be executed directly from the Mini-Assembler.

Using the Mini-Assembler

The Mini-Assembler saves one address, that of the program counter. Before you start to type a program, you must set the program counter to point to the location where you want the Mini-Assembler to store your program. Do this by typing the address followed by a colon.

After the colon, type the mnemonic for the first instruction in your program, followed by a space and the operand of the instruction. Now press Return. The Mini-Assembler converts the line you typed into hexadecimal, stores it in memory beginning at the location of the program counter, and then disassembles it again and displays the disassembled line. It then displays a prompt on the next line.

Now the Mini-Assembler is ready to accept the second instruction in your program. To tell it that you want the next instruction to follow the first, don't type an address or a colon: just type a space and the next instruction's mnemonic and operand, then press Return. The Mini-Assembler assembles that line and waits for another.

If the line you type has an error in it, the Mini-Assembler beeps loudly and displays a caret (^) under or near the offending character in the input line. Most common errors are the result of typographical mistakes: misspelled mnemonics, missing parentheses, and so forth. The Mini-Assembler also rejects the input line if you forget the space before or after a mnemonic or include an extraneous character in a hexadecimal value or address. If the destination address of a branch instruction is out of the range of the branch (more than 127 locations distant from the address of the instruction), the Mini-Assembler flags this as an error.

- *Dollar signs:* In this manual, dollar signs ($) in addresses signify that the addresses are in hexadecimal notation. The dollar signs are ignored by the Mini-Assembler and can be omitted in programs.

```
!300:LDX #02
0300-   A2 02       LDX    #$02
! LDA $00,X
0302-   B5 00       LDA    $00,X
! STA $10,X
0304    95 10       STA    $10,X
! DEX
0306-   CA          DEX
! STA $C030
0307-   8D 30 C0    STA    $C030
! BPL $0302
030A-   10 F6       BPL    $0302
! BRK
030C-   00          BRK
!
```

To leave the Mini-Assembler and reenter the Monitor, press Return at a blank line.

Your assembly-language program is now stored in memory. You can display it with the LIST command:

```
*300L
0300-  A2 02        LDX   #$02
0302-  B5 00        LDA   $00,X
0304-  95 10        STA   $10,X
0306-  CA           DEX
0307-  8D 30 C0     STA   $C030
030A-  10 F6        BPL   $0302
030C-  00           BRK
030D-  00           BRK
030E-  00           BRK
030F-  00           BRK
0310-  00           BRK
```

```
0311-  00          BRK
0312-  00          BRK
0313-  00          BRK
0314-  00          BRK
0316-  00          BRK
0316-  00          BRK
0317-  00          BRK
0318-  00          BRK
0319-  00          BRK
*
```

Mini-Assembler instruction formats

The Apple IIc Mini-Assembler recognizes 66 mnemonics and 15 addressing formats. The mnemonics are standard, as used in the *Synertek Programming Manual* (Apple part number A2L0003), but the addressing formats are somewhat different, as shown in Table 10-1.

Table 10-1
Mini-Assembler address formats

Addressing mode	Format
Accumulator	*
Implied	*
Immediate	#${*value*}
Absolute	${*address*}
Zero page	${*address*}
Indexed zero page	${*address*},X ${*address*},Y
Indexed absolute	${*address*},X ${*address*},Y
Relative	${*address*}
Indexed indirect	(${*address*},X)
Indirect indexed	(${*address*},Y)
Absolute indirect	(${*address*})

* These instructions have no operands.

An address consists of one or more hexadecimal digits. The Mini-Assembler interprets addresses the same way the Monitor does: if an address has fewer than four digits, the Mini-Assembler adds leading 0's; if the address has more than four digits, then it uses only the last four.

There is no syntactical distinction between the absolute and zero-page addressing modes. If you give an instruction to the Mini-Assembler that can be used in both absolute and zero-page mode, the Mini-Assembler assembles that instruction in absolute mode if the operand for that instruction is greater than $FF, and it assembles it in zero-page mode if the operand is less than $0100.

Instructions in accumulator mode and implied addressing mode need no operands.

Branch instructions, which use the relative addressing mode, require the target address of the branch. The Mini-Assembler calculates the relative distance to use in the instruction automatically. If the target address is more than 127 locations distant from the instruction, the Mini-Assembler sounds a bell (beep), displays a caret (^) under the target address, and does not assemble the line.

If you give the Mini-Assembler the mnemonic for an instruction and an operand, and the addressing mode of the operand cannot be used with the instruction you entered, then the Mini-Assembler will not accept the line.

Summary of Monitor commands

Here is a summary of the Monitor commands, showing the syntax diagram for each one.

Examining memory

{*adrs*}Return	Displays the value contained in one location.
{*adrs1*}.{*adrs2*}Return	Displays the values contained in all locations between {*adrs1*} and {*adrs2*}
Return	Displays the values in up to eight locations following the last opened location.
{*adrs*}L	Lists disassembled code starting at {*adrs*} and continuing until the screen is full.

Changing the contents of memory

{*adrs*}:{*val*}{*val*}...	STORE command. Stores the values in consecutive memory locations starting at {*adrs*}.
:{*val*}{*val*}...	Stores values in memory starting at the next changeable location.

Moving and comparing

{*dest*}<{*start*}.{*end*}M	MOVE command. Copies the values in the range {*start*}.{*end*} into the range beginning at {*dest*}.
{*dest*}<{*start*}.{*end*}V	VERIFY command. Compares the values in the range {*start*}.{*end*} to those in the range beginning at {*dest*}.

The Register command

Control-E	EXAMINE command. Displays the locations where the contents of the 65C02's registers are stored and opens them for changing.

Miscellaneous Monitor commands

I	INVERSE command. Sets inverse display mode.
N	NORMAL command. Sets normal display mode.
Control-B	BASIC command. Enters the language currently active (normally Applesoft).
Control-C	CONTINUE BASIC command. Returns to the language currently active (normally Applesoft).
{*val*}+{*val*}	Adds the two values and prints the hexadecimal result.
{*val*}–{*val*}	Subtracts the second value from the first and prints the result.
{*port*}Control-P	Redirects output to the device connected to port number {*port*}. If {*port*}=0, sends output to the video display. Use only when the enhanced video firmware is not active (checkerboard cursor).
Escape Control-Q	Redirects output to video display when enhanced video firmware is active (solid block cursor).
{*port*}Control-K	Takes input from the device connected to port number {*port*}. If {*port*}=0, accepts input from the keyboard.
Control-Y	USER command. Jumps to the machine-language subroutine at location $03F8.

Running and listing programs

{*adrs*}G	Transfers control to the machine-language program beginning at {*adrs*}.
{*adrs*}L	Disassembles and displays 20 instructions starting at {*adrs*}. Subsequent L's display 20 more instructions each.



Chapter 11

Hardware Implementation

Most of this manual describes functions—what the Apple IIc does. This chapter, on the other hand, describes objects—the pieces of hardware the Apple IIc uses to carry out its functions. If you are designing a device to connect to the Apple IIc back panel, or if you just want to know more about how the Apple IIc is built, you should study this chapter.

Environmental specifications

The Apple IIc is quite sturdy when used in the way it was intended: as a transportable computer, made for use in an indoor environment. You can carry it by its handle from room to room, but for longer trips you should use its carrying case or some other protective container (such as an attaché case). Table 11-1 defines the conditions under which the Apple IIc is designed to function properly.

Table 11-1
Environmental specifications

Operating temperature	10° to 40° C (50° to 104° F)
Relative humidity	20% to 95%

You should treat the Apple IIc with the same kind of care as any other electrical appliance; protect it from physical abuse, and be careful not to bump it against furniture when you move it around. Put it in an attache case or other protective covering if you carry it outside. You should also protect the mechanical keyboard and the electrical connectors inside the case from spilled liquids, particularly those with dissolved contaminants, such as soups, fruit juices, and carbonated soft drinks.

In normal operation (with the handle locked in its down position), enough air flows through the openings in the case to keep the insides from getting too hot. If you do overheat your Apple IIc—for example, by blocking the upper or lower ventilation openings—the first symptom will be erratic operation, such as unexpectedly changed data. (The memory devices in the Apple IIc are especially sensitive to heat.) Letting the machine cool down by turning it off for a while and unblocking the vents before using it again will bring it back to normal operation. The only exception to this is if you have gotten your Apple IIc *too* hot and physically damaged some internal component.

Disks are another heat-sensitive element of the system. If the built-in drive becomes too hot, a disk within can warp or even melt. A melted or warped disk can't be used again.

Power requirements

The electrical power used by the Apple IIc—and everything that draws power from it—is limited by the capacities of the computer's power supply and internal voltage converter. This section describes these limits for the USA external power supply. Appendix G describes them for models built for other countries. The internal voltage converter is the same on all models.

The external power supply

If you purchased your Apple IIc outside the USA, consult Appendix G for external power supply characteristics.

The external power supply operates on normal household AC power and provides DC power to the Apple IIc internal converter. The basic specifications of the external power supply are listed in Table 11-2. The Apple IIc external power supply's cord must be plugged into a three-wire 115-volt (nominal) outlet. A two-wire outlet is not properly grounded—using it will damage the external power supply and perhaps the Apple IIc as well. The line voltage must be in the range given in Table 11-2.

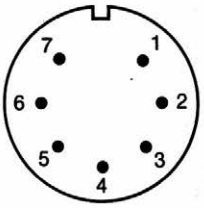
Warning Important safety instructions: This product is equipped with a three-wire grounding-type plug—a plug having a third (grounding) pin. This plug will only fit into a grounding-type AC outlet. This is a safety feature.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

Do not defeat the purpose of the grounding-type plug.

Table 11-2
Power supply specifications

Line voltage	105 to 129 VAC, 60 Hz
Maximum power consumption	25 W
Supply voltage	+15 VDC (nominal)
Supply current	1.2 A (nominal)



Pin	Signal
1	Not connected
2, 3	Signal ground
4	Shield ground
5, 6	+15 VDC
7	Not connected

Figure 11-1
External power connector

The external power connector

The external power supply is attached to the internal converter by means of a 7-pin DIN connector. The connector pins are identified in Figure 11-1 and Table 11-3.

Table 11-3
External power connector signals

Pin	Signal	Description
1, 7		Not connected
2, 3	Ground	Common electrical ground
4	Chassis	Chassis ground
5, 6	+15V	+15-volt DC input to converter

The internal converter

The internal converter in the Apple IIc operates with a supply voltage from 9 to 20 volts DC as provided by the external power supply or its equivalent. The internal converter provides enough low-voltage electrical power for the built-in electronics plus an external disk drive attached via the 19-pin connector. The basic specifications of the internal converter are listed in Table 11-4. Minus 5 volts is derived from the –12 volts (nominal) provided by the voltage converter.

Table 11-4
Internal converter specifications

Input voltage	+9 to 20 VDC
Maximum power consumption	25 W
Supply voltages	+5V ±5% +12V ±10% –12V ±10%
Maximum supply currents	+5V: 1.5 A +12V: 0.6 A continuous 0.9 A intermittent 1.5 A surge (for < 100 ms) –12V: 100 mA (–5V: 50 mA)
Maximum case temperature	60° C (140° F)

The Apple IIc uses a switching-type internal voltage converter as a power supply. It is small and lightweight, and it generates less heat than other types of voltage converters.

The voltage converter works by using the DC voltage input to power a variable-frequency oscillator. The oscillator drives a small transformer with several separate windings to produce the different voltages required. A circuit compares the voltage of the +5-volt supply with a reference voltage and feeds an error signal back to the oscillator circuit. The oscillator circuit uses the error signal to control the duty cycle of its oscillation and keep the output voltages in their normal ranges.

The converter includes circuitry to protect itself and the other electronic parts of the Apple IIc by limiting all three output voltages whenever it detects one of the following malfunctions:

- any supply voltage short-circuited to ground
- any output voltage outside the normal range

Whenever one of these malfunctions occurs, the protection circuit varies the duty cycle of the oscillator, and all the output voltages drop to 0 if they cannot be brought back into their normal range.

Apple IIc overall block diagram

Figure 11-2 is an overall block diagram of the Apple IIc. The following sections contain more detailed diagrams of the major parts of the machine. A full set of schematic diagrams of the Apple IIc appears later in this chapter.

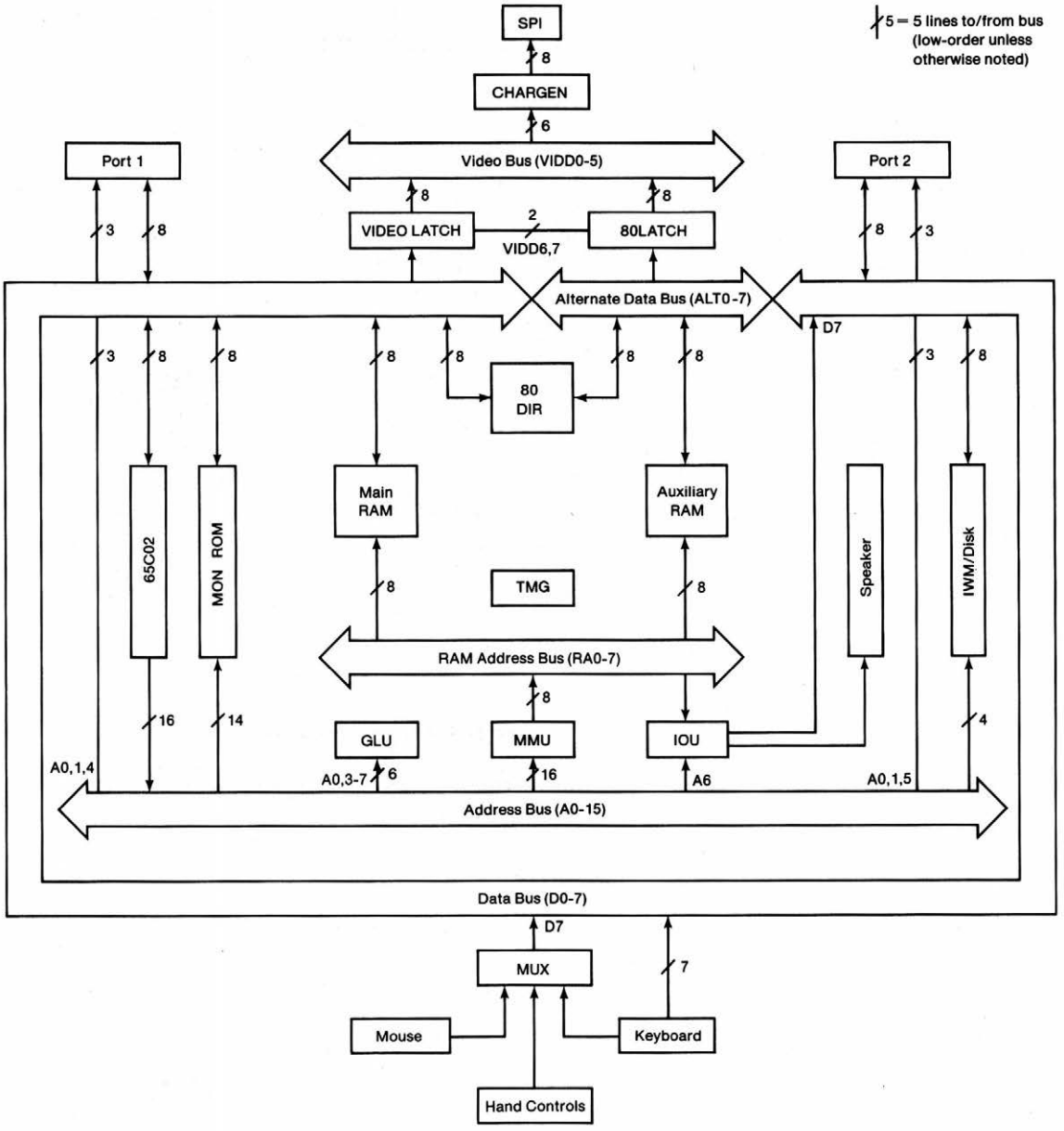


Figure 11-2
Apple IIc block diagram

The 65C02 microprocessor

CMOS (complementary metal-oxide semiconductor) is a way of making integrated circuits that require less power to operate than other technologies such as NMOS (negative-doped metal-oxide semiconductor), used by the 6502.

The Apple IIc uses a **CMOS** 6502 (designated as 65C02) microprocessor as its central processing unit (CPU). The 65C02 in the Apple IIc runs at a clock rate of 1.023 MHz and performs up to 500,000 8-bit operations per second.

❖ *Note:* The clock rate is not a very good criterion for comparing different types of microprocessors. The 65C02 has a simpler instruction cycle than most other microprocessors and it uses instruction pipelining for faster processing. The speed of the 65C02 with a 1-MHz clock is equivalent to many other types of microprocessors with clock rates up to 5 MHz.

These instructions are described in Appendix A.

In addition to requiring less power than earlier NMOS 6502 processors, the 65C02 in the Apple IIc has 27 new instructions. However, programs that use these additional instructions are not backward compatible with other Apple II series computers that are not equipped with a CMOS 6502.

65C02 block diagram

Figure 11-3 is a block diagram of the 65C02 microprocessor. Table 11-5 contains the general specifications of this chip. The 65C02 has a 16-bit address bus, giving it an address space of 64K bytes. The Apple IIc uses special techniques to address a total of more than 64K (see Chapter 2).

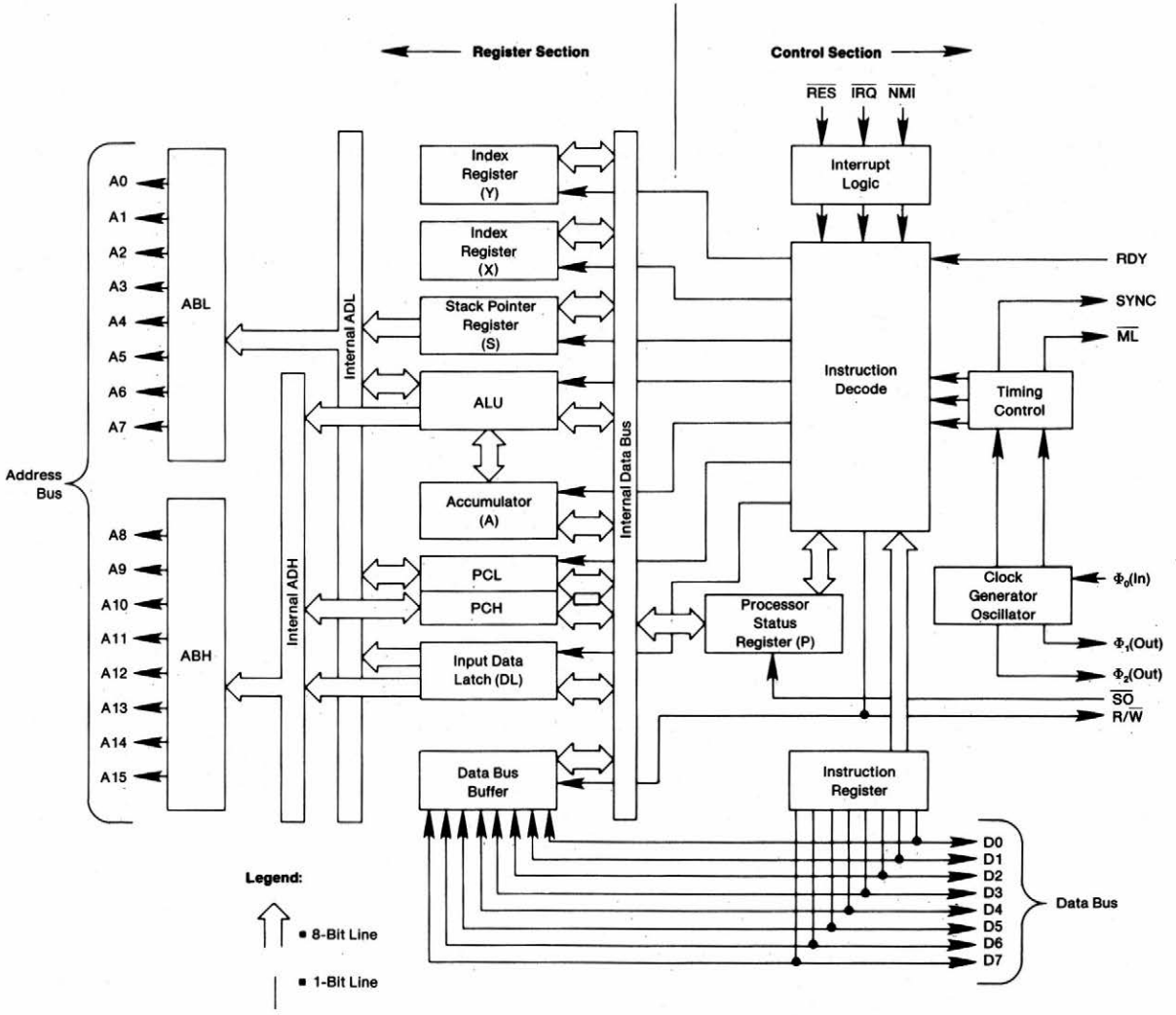


Figure 11-3
65C02 block diagram (copyright © 1982 by NCR Corporation; used by permission)

Table 11-5
65C02 microprocessor specifications

Type	65C02
Register complement	8-bit accumulator (A) 8-bit index registers (X,Y) 8-bit stack pointer (S) 8-bit processor status (P) 16-bit program counter (PC)
Data bus	8 bits wide
Address bus	16 bits wide
Address range	65,536 (64K)
Interrupts	IRQ (maskable) NMI (nonmaskable) BRK (programmed)
Operating voltage	+5V (±5%)
Power dissipation	5 mW (at 1 MHz)

65C02 timing

The Apple IIc's operation is controlled by a set of synchronous timing signals, sometimes called *clock signals*. The Apple IIc uses a 14.318-MHz master timing signal, called *14M,* to produce all the other timing signals. These timing signals perform two major tasks: controlling the computing functions, and generating the video display. The timing signals directly involved with the 65C02's operation are described in this section. Other timing signals are described later in this chapter.

The relationships of the main 65C02 timing signals are diagrammed in Figure 11-4, and the signals are listed in Table 11-6. The 65C02 clock signals are ø1 and ø0, complementary signals at a frequency of 1.0227 MHz. The Apple IIc signal ø0 is similar to the signal ø2 in Appendix A (it isn't identical—it's a tiny bit early).

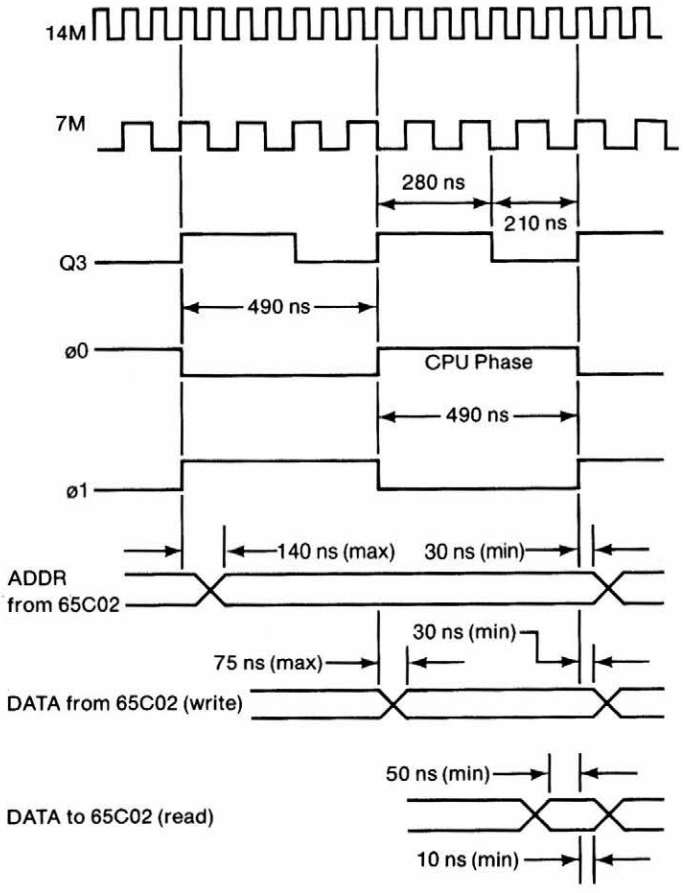


Figure 11-4
65C02 timing signals

Table 11-6
65C02 timing signal descriptions

Signal	Description
14M	Master oscillator, 14.318 MHz; also 80-column dot clock
VID7M	Intermediate timing signal and 40-column dot clock
Q3	Intermediate timing signal, 2.045 MHz with asymmetrical duty cycle
ø0	Phase 0 of 65C02 clock, 1.0227 MHz; complement of ø1
ø1	Phase 1 of 65C02 clock, 1.0227 MHz; complement of ø0

The 65C02's operations are related to the clock signals in a simple way: internal during ø1, external during ø0. The 65C02 puts an address on the address bus during ø1. This address is valid not later than 110 nanoseconds after ø1 goes high and remains valid through all of ø0. The 65C02 reads or writes data during ø0. If the 65C02 is writing, the read/write signal is low during ø0 and the 65C02 puts data on the data bus. The data are valid not later than 75 nanoseconds after ø0 goes high. If the 65C02 is reading, the read/write signal remains high. Data on the data bus must be valid no later than 50 nanoseconds before the end of ø0.

More information about the 65C02 and its instruction set is in Appendix A.

The custom integrated circuits

Most of the circuitry that controls memory and I/O addressing in the Apple IIc is in five custom integrated circuits:

- the memory management unit (MMU)
- the input-output unit (IOU)
- the timing generator (TMG)
- the general logic unit (GLU)
- the disk controller unit, also known as the Integrated Woz Machine (IWM)

The soft switches that control the various I/O and addressing modes of the Apple IIc are addressable flags inside the MMU, IOU, and GLU. The functions of the MMU and IOU are not as independent as their names suggest; working together, they generate all the addressing signals. For example, the MMU generates the RAM address signals for the CPU, while the IOU generates similar RAM address signals for the video display and most I/O hardware addresses.

The memory management unit (MMU)

The circuitry inside the MMU implements these soft switches:

- Page 2 display (Page2) (described in Chapter 5)
- high-resolution mode (HiRes) (Chapter 5)
- store to 80-column display (80Store) (Chapter 5)
- select bank 2 (Bank2) (Chapter 2)

- □ enable bank-switched RAM (EnlCRAM) (Chapter 2)
- □ read auxiliary memory (RAMRd) (Chapter 2)
- □ write auxiliary memory (RAMWrt) (Chapter 2)
- □ auxiliary stack and zero page (AltZP) (Chapter 2)
- □ reset mouse Y interrupt (RstYInt) (Chapter 9)
- □ reset mouse X interrupt (RstXInt) (Chapter 9)

These switches are available on MMU pin 21, which is connected to bit 7 on the data bus. Figure 11-5 shows the MMU pinouts; Table 11-7 describes the signals.

Important A signal name followed by an asterisk is active low—that is, it is true when the signal is at a TTL high (+5V) level.

The 64K dynamic RAMs used in the Apple IIc use a multiplexed address, as described later in this chapter. The MMU generates this multiplexed address for memory reading and writing by the 65C02 CPU.

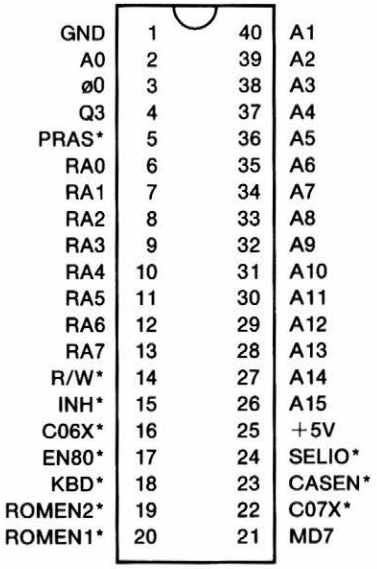


Figure 11-5
MMU pinouts

Table 11-7
MMU signal descriptions

Pin	Signal	Description
1	GND	Power and signal common
2	A0	65C02 address input
3	ø0	Clock phase 0 input
4	Q3	Timing signal input
5	PRAS*	Memory row-address strobe
6–13	RA0–RA7	Multiplexed address output
14	R/W*	65C02 read-write control signal
15	INH*	Inhibits main memory (tied to +5V)
16	C06X*	Causes $C06x outputs to go to 0 during ø0
17	EN80*	Enables auxiliary RAM
18	KBD*	Enables keyboard data bits 0-6
19	ROMEN2*	Enables ROM (tied to ROMEN1*)
20	ROMEN1*	Enables ROM (tied to ROMEN2*)
21	MD7	State of MMU flags on data bus bit 7
22	C07X	Causes $C07x outputs to go to 0 during ø0
23	CASEN*	Enables main RAM
24	SELIO*	Goes to 0 during ø0 for any access to $C0 page except $C08x, Bx, Cx, or Fx
25	+5V	Power
26–40	A15–A1	65C02 address input

The input/output unit (IOU)

Input/output unit (IOU) implements the following soft switches, all described in Chapters 2 and 3:

- Page 2 display (Page2)
- high-resolution mode (HiRes)
- text mode (TEXT)
- mixed mode (MIXED)
- 80-column display (80Col)
- character-set select (AltChar)
- any-key-down (AKD)
- mouse movement (X0, Y0)
- vertical blanking interrupt (VBlInt)

These switches are available on IOU pin 9, which is connected to bit 7 on the data bus. Figure 11-6 shows the IOU pinouts; Table 11-8 describes the signals.

The 64K dynamic RAMs used in the Apple IIc require a multiplexed address, as described later in this chapter. The IOU generates this multiplexed address during clock phase 1 for the data transfers required for display and memory refresh. The way this address is generated is described under "The Video Counters" in this chapter.

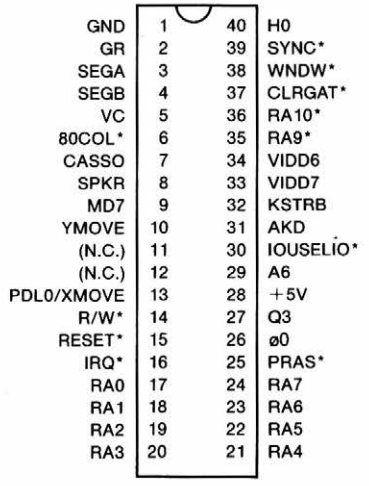


Figure 11-6
IOU pinouts

Table 11-8
IOU signal descriptions

Pin	Signal	Description
1	GND	Power and signal common
2	GR	Graphics mode enable
3	SEGA	In text mode, works with VC (see pin 5) and SEGB to determine character row address
4	SEGB	In text mode, works with VC (see pin 5) and SEGA; in graphics mode, selects high resolution when low, low resolution when high
5	VC	Displays vertical counter bit: in text mode, SEGA, SEGB, and VC determine which of the eight rows of a character's dot pattern to display; in low resolution, selects upper or lower block defined by a byte

Table 11-8 (continued)
IOU signal descriptions

Pin	Signal	Description
6	80COL*	80-column video enable
7	CASSO	Reserved
8	SPKR	Speaker output signal
9	MD7	Internal IOU flags for data bus (bit 7)
10	YMOVE	Detects mouse movement along Y axis
11	N.C.	Not used
12	N.C.	Not used
13	PDL0/XMOVE	Detects mouse movement along X axis
14	R/W*	65C02 read-write control signal
15	RESET*	Power on and reset output
16	IRQ*	Maskable interrupt line to 65C02
17-24	RA0–RA7	Video refresh multiplexed RAM address (phase 1)
25	PRAS*	Row-address strobe (phase 0)
26	ø0	Master clock phase 0
27	Q3	Intermediate timing signal
28	+5V	Power
29	A6	Address bit 6 from 65C02
30	IOUSELIO*	Derived from the SELIO* output for MMU pin 24
31	AKD	Any-key-down signal
32	KSTRB	Keyboard strobe signal
33,34	VIDD7,VIDD6	Video display data bits
35,36	RA9*,RA10*	Video display control bits
37	CLRGAT*	Color-burst gate (enable)
38	WNDW*	Displays blanking signal
39	SYNC*	Displays synchronization signal
40	H0	Displays horizontal timing signal (low bit of character counter)

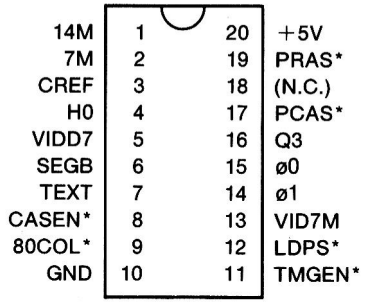


Figure 11-7
TMG pinouts

The timing generator (TMG)

A custom timing generator chip (TMG) generates several timing and control signals in the Apple IIc. The TMG pinouts are shown in Figure 11-7; the signals are listed in Table 11-9.

Table 11-9
TMG signal descriptions

Pin	Signal	Description
1	14M	14.318-MHz master timing signal input
2	7M	7.159-MHz timing signal
3	CREF	3.5795-MHz color reference timing signal
4	H0	Horizontal video timing signal
5	VIDD7	Video data bit 7
6	SEGB	Video timing signal
7	TEXT	Video display text-modes enable
8	CASEN*	RAM enable (CAS enable)
9	80COL*	Enables 80-column display mode
10	GND	Power and signal common
11	TMGEN*	Enables master timing
12	LDPS*	Video shift-register load enable
13	VID7M	Video dot clock enable, 7 MHz or continuous 0
14	ø1	Phase 1 system clock
15	ø0	Phase 0 system clock
16	Q3	Intermediate timing and strobe signal
17	PCAS*	RAM column-address strobe
18	N.C.	Reserved for testing
19	PRAS*	RAM row-address strobe
20	+5V	Power

The general logic unit (GLU)

The general logic unit is a single chip that contains the miscellaneous logic required for the system. It provides

- all RAM read/write timing
- double high-resolution enable/disable
- soft-switch status registers
- write command registers
- IOU control for mouse interrupts
- double high-resolution soft switches

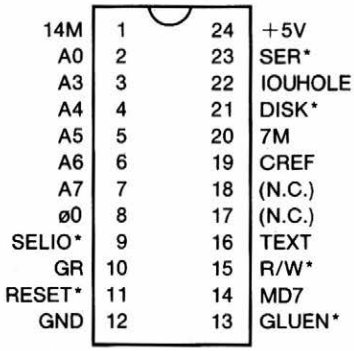


Figure 11-8
GLU pinouts

The GLU's pin assignments are shown in Figure 11-8 and its signals are listed in Table 11-10.

Table 11-10
GLU signal descriptions

Pin	Signal	Description
1	14M	Master clock (14.318 MHz)
2,3–7	A0,A3–A7	Address lines to select least significant byte of addresses on C0 page
8	ø0	Phase 0 of 1.0227-MHz processor sync clock
9	SELIO*	Device select for selecting most significant byte of the address
10	GR	Graphics mode select line
11	RESET*	Master reset for system; resets GLU
12	GND	Ground reference and negative supply
13	GLUEN*	Enables GLU
14	MD7	Indicates status of MMU flags on data bus bit 7
15	R/W*	Read/write qualifier input from processor
16	TEXT	Signal used to generate video timing in double high-resolution or not-graphics
17,18	N.C.	Not used
19	CREF	Color reference signal
20	7M	7-MHz clock output
21	DISK*	Disk controller device select output
22	IOUHOLE	Controls IOUSELIO
23	SER*	Serial controller device select output
24	+5V	+5 volt supply

The disk controller unit (IWM)

The IWM (for Integrated Woz Machine) is a disk controller that includes, on a single chip, all the capabilities of the disk controller card originally designed by Steve Wozniak in 1977.

For further information on **GCR**, refer to "Disk I/O."

Right after reset, the IWM is an integrated **GCR** (group code recording) disk drive controller. It also has a status register, mode register, and multiple operating modes. It provides both synchronous and asynchronous modes, and a fast mode with a data rate twice that of normal disk I/O speeds. Figure 11-9 shows the IWM pin assignments; Table 11-11 describes the IWM signals.

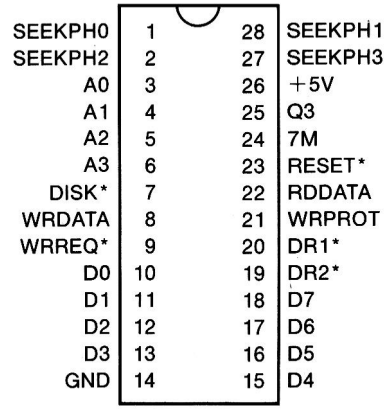


Figure 11-9
IWM pinouts

Table 11-11
IWM signal descriptions

Pin	Signal	Description
1	SEEKPH0	Stepper motor control phase 0, one of four programmable disk drive motor phase outputs.
2	SEEKPH2	Stepper motor control phase 2.
3	A0	The data input to the state bit selected by A1 to A3.
4–6	A1–A3	These three inputs select one of the eight bits in the state register to be updated.
7	DISK*	Device enable. The falling edge of DISK* latches information on A1 to A3. The rising edge of either Q3 or DISK* qualifies write register data.
8	WRDATA	The serial data output. Each 1-bit causes a transition on this output.
9	WRREQ*	This signal is a programmable buffered output line.
10–13	D0–D3	D0 to D7 make up the bidirectional data bus.
14	GND	Ground reference and negative supply.
15–18	D4–D7	The remaining bits of the bidirectional data bus.
19	DR2*	Drive 2 select.

Table 11-11 (continued)
IWM signal descriptions

Pin	Signal	Description
20	DR1*	Drive 1 select.
21	WRPROT	Write-protect input; this can be polled via bit 7 of the status register.
22	RDDATA	Serial data input line. The IWM synchronizes the falling transition of each pulse.
23	RESET*	IWM reset: places all IWM outputs in their inactive state and sets all state and mode register bits to 0.
24	7M	7-MHz clock input.
25	Q3	A 2.0-MHz clock input used to qualify the timing of the serial data being written or read.
26	+5V	The +5 volt supply.
27	SEEKPH3	Stepper motor control phase 3.
28	SEEKPH1	Stepper motor control phase 1.

Memory addressing

The 65C02 microprocessor can directly address 65,536 locations. The Apple IIc uses this entire address space, and then some: some areas in memory are used for more than one function. The following sections describe the memory devices used in the Apple IIc and the way they are addressed. Input and output also use portions of the memory address space; refer to Chapter 2 for information.

Figure 11-10 illustrates the Apple IIc's overall memory bus organization and memory selection signals.

❖ *Note:* Some Apple IIc's have ROMs with 27xx designations, some have 23xx. They are functionally equivalent.

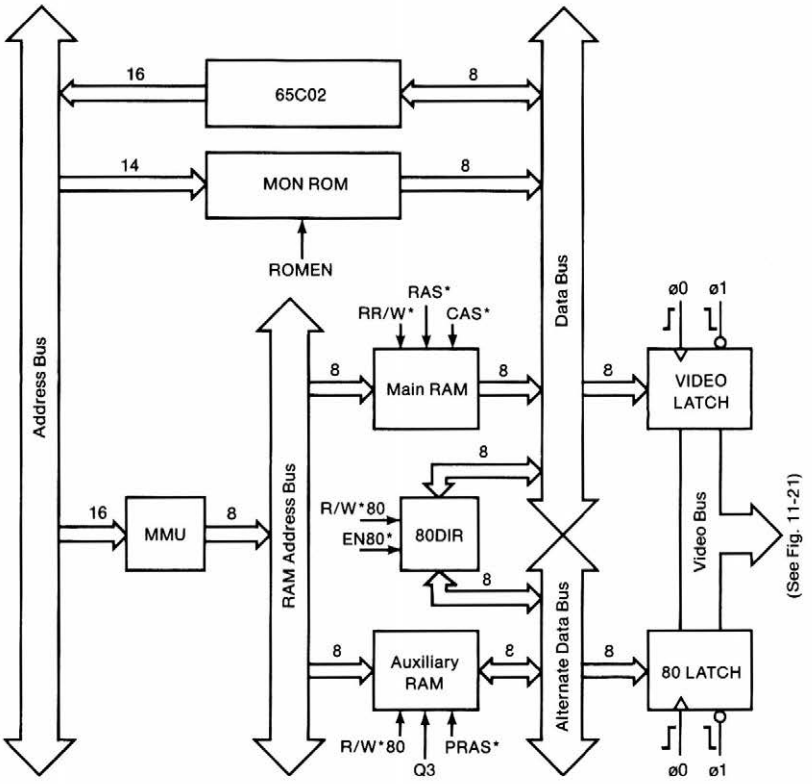


Figure 11-10
Memory bus organization

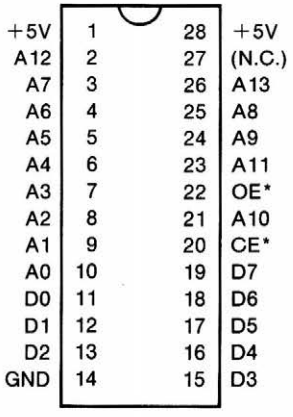


Figure 11-11
23128 ROM pinouts (in type 23256 ROM, pin #27 is A14)

ROM addressing

In the Apple IIc the following programs are permanently stored in a type 23128 16K-by-8-bit ROM (Figure 11-11):

- Applesoft editor and interpreter
- Monitor
- enhanced video firmware

UniDisk 3.5 The version of the Apple IIc that supports the UniDisk 3.5 uses a 23256 32K-by-8-bit ROM. It needs the extra space for the Protocol Converter, Mini-Assembler, and other added functions that it supports.

Memory expansion The Apple IIc that supports the memory expansion card also uses the 23256 ROM IC.

The ROM is enabled by two signals called *ROMEN1* and *ROMEN2*. (In the Apple IIc, ROMEN1 and ROMEN2 are electrically connected.) The segment of the ROM enabled by ROMEN1 occupies the memory address space from $C100 through $DFFF. The address space from $C300 through $C3FF and much of $C800 through $CFFF contains the enhanced video firmware.

These ROM address allocations are approximately true (some space sharing takes place):

- ROM addresses $C000 through $C0FF are never available.
- ROM addresses $C100 and $C200 are entry points to firmware for serial ports 1 and 2, respectively.
- ROM address $C400 is the entry point to mouse interface support.
- ROM addresses $C500 through $C5FF are reserved.
- ROM address $C600 is the entry point to firmware for the built-in and external disk drives. The built-in drive is considered slot 6 drive 1 or its equivalent. The external drive is considered slot 6 drive 2.
- ROM addresses starting at $C700 support (from the Monitor) the external drive as if it were slot 7 drive 1, for external-drive startup only.
- Addresses $D000 through $F7FF contain the Applesoft BASIC interpreter; addresses $F800 through $FFFF contain the Monitor firmware.

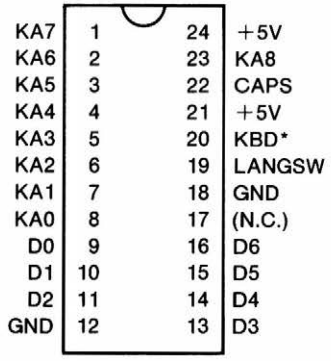


Figure 11-12
2316 ROM pinouts

Memory expansion

The Apple IIc that supports the memory expansion card has a ROM map that is different from that given for the original and UniDisk 3.5 IIc. The memory expansion ROM map is provided in Appendix I.

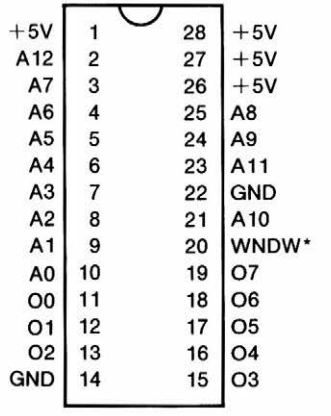


Figure 11-13
2364 pinouts

The other ROMs in the Apple IIc are a type 2316 ROM (Figure 11-12) used for the keyboard character decoder, and a type 2364 ROM (Figure 11-13) used for character sets for the video display. This 2364 ROM is rather large because it includes a section of straight-through bit-mapping for the graphics modes. This way, graphics display video can pass through the same circuits as text without additional switching circuitry.

RAM addressing

The RAM (programmable) memory in the Apple IIc is used both for program and data storage and for the video display. The areas in RAM that are used for the display are accessed both by the 65C02 microprocessor and by the video display circuits. In some computers, this dual access results in addressing conflicts (cycle stealing) that can cause temporary dropouts in the video display. This problem does not occur in the Apple IIc, thanks to the way the microprocessor and the video circuits share the memory.

The memory circuits in the Apple IIc take advantage of the two-phase system clock to interleave the microprocessor memory accesses and the display memory accesses so that they never interfere with each other. The microprocessor reads or writes to RAM only during ø0, and the display circuits read data only during ø1.

Dynamic RAM refreshment

The image on a video display is not permanent; it fades rapidly and must be refreshed periodically. To refresh the video display, the Apple IIc reads the data in the active display page and sends them to the display. To prevent visible flicker in the display, and to conform to standard practice for broadcast video, the Apple IIc refreshes the display 60 times per second.

The dynamic RAM devices used in the Apple IIc also need a kind of refreshment, because the data are stored in the form of electric charges that diminish with time and must be replenished. The Apple IIc is designed so that refreshing the display also refreshes the dynamic RAMs. The next few paragraphs explain how this is done.

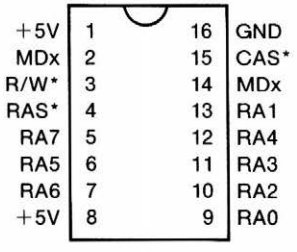


Figure 11-14
64K RAM pinouts

The job of refreshing the dynamic RAM devices is minimized by the structure of the devices themselves. The individual data cells in each RAM device are arranged in a rectangular array of rows and columns. When the device is addressed, the part of the address that specifies a row is presented first, followed by the address of the column. Splitting information into parts that follow each other in time is called *multiplexing*. Because only half of the address is needed at one time, multiplexing the address reduces the number of pins needed for connecting the RAMs (Figure 11-14).

Memory expansion

In the Apple IIc that supports the memory expansion card, the 16 64Kx1 RAM ICs used for the original and UniDisk 3.5 IIc's are replaced by 4 64Kx4 ICs.

Table 11-12
RAM address multiplexing

Mux'd address	Row address	Column address
RA0	A0	A9
RA1	A1	A6
RA2	A2	A10
RA3	A3	A11
RA4	A4	A12
RA5	A5	A13
RA6	A7	A14
RA7	A8	A15

Different manufacturers' 64K RAMs have cell arrays of either 128 rows by 512 columns or 256 rows by 256 columns. Only the row portion of the address is used in refreshing the RAMs.

Now consider how the display is refreshed. As described later in this chapter, the display circuitry generates a sequence of 8,192 memory addresses in high-resolution mode; in text and low-resolution modes, this sequence is the 1,024 display-page addresses repeated 8 times. The display address cycles through this sequence 60 times a second, or once every 17 milliseconds. The way the low-order address lines are assigned to the RAMs, the row address cycles through all 256 possible values once every 2 milliseconds (see Table 11-12). This more than satisfies the refresh requirements of the dynamic RAMs.

Dynamic RAM timing

The Apple IIc's microprocessor clock runs at a speed of 1.023 MHz, but the interleaving of CPU and display cycles means that the RAM is being accessed at a 2-MHz rate, or a cycle time of just under 500 nanoseconds. Data for the CPU are strobed by the falling edge of ø0, and display data are strobed by the falling edge of ø1, as shown in Figure 11-15.

The RAM timing looks complicated because the RAM address is multiplexed, as described previously. The MMU takes care of multiplexing the address for the CPU cycle, and the IOU performs the same function for the display cycle. The multiplexed address is sent to the RAM ICs over the lines RA0–RA7 (Table 11-13). Along with the other timing signals, the TMG generates two signals that control the RAM addressing: row-address strobe (RAS) and column-address strobe (CAS).

Table 11-13
RAM timing signals

Signal	Description
ø0	Clock phase 0 (CPU phase)
ø1	Clock phase 1 (display phase)
RAS	Row-address strobe
CAS	Column-address strobe
Q3	Alternate RAM/column-address strobe
RA0–RA7	Multiplexed address bus
MD0–MD7	Internal data bus

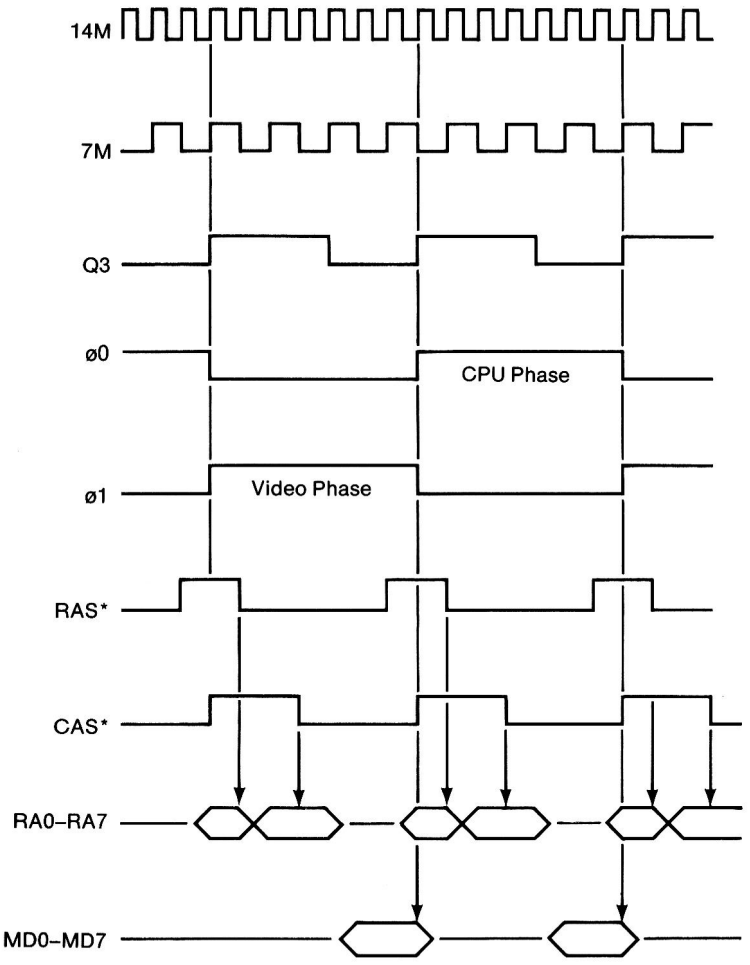


Figure 11-15
RAM timing signals

The keyboard

The Apple IIc's keyboard is a matrix of key switches connected to an AY-3600-type keyboard decoder via a ribbon cable and a 26-pin connector (Figure 11-16). The AY-3600 scans the array of keys over and over to detect any keys pressed. The scanning rate is set by the external resistor-capacitor network made up of C46 and R6. The debounce time is also set externally, by C45.

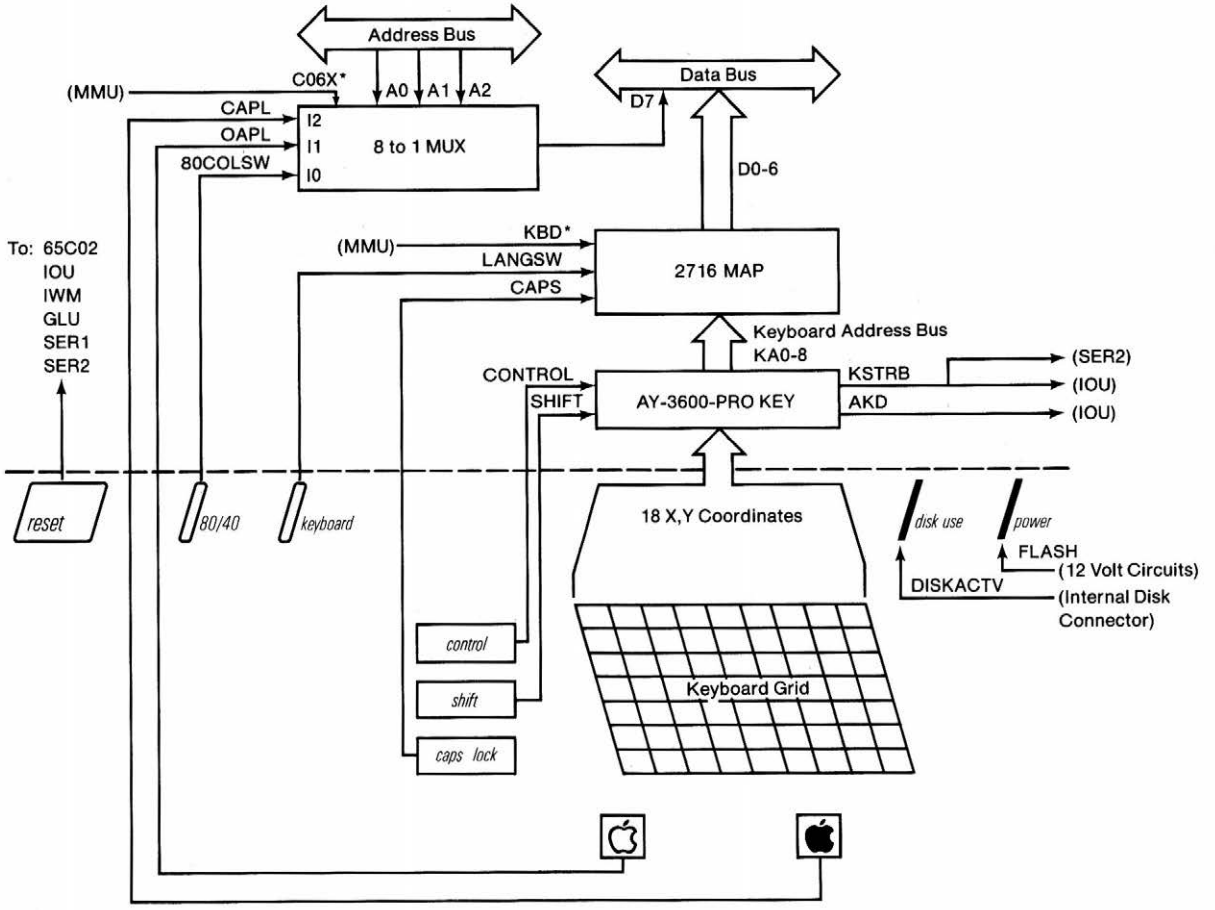


Figure 11-16
Keyboard circuit diagram

The AY-3600's outputs include five bits of key code plus separate lines for Control, Shift, any-key-down, and keyboard strobe. The any-key-down and keyboard-strobe lines are connected to the IOU, which addresses them as soft switches. The key-code line and Control and Shift are inputs to a separate 2316 ROM. The ROM translates them to the character codes that are enabled onto the data bus by signals named *KBD** and *ENKBD**. The KBD* signal is enabled by the MMU whenever a program reads location $C000, as described in Chapter 2.

Figure 11-17 illustrates the events that occur when a key is pressed, when the keypress is detected by a program, and when a key is pressed and held for more than about a second.

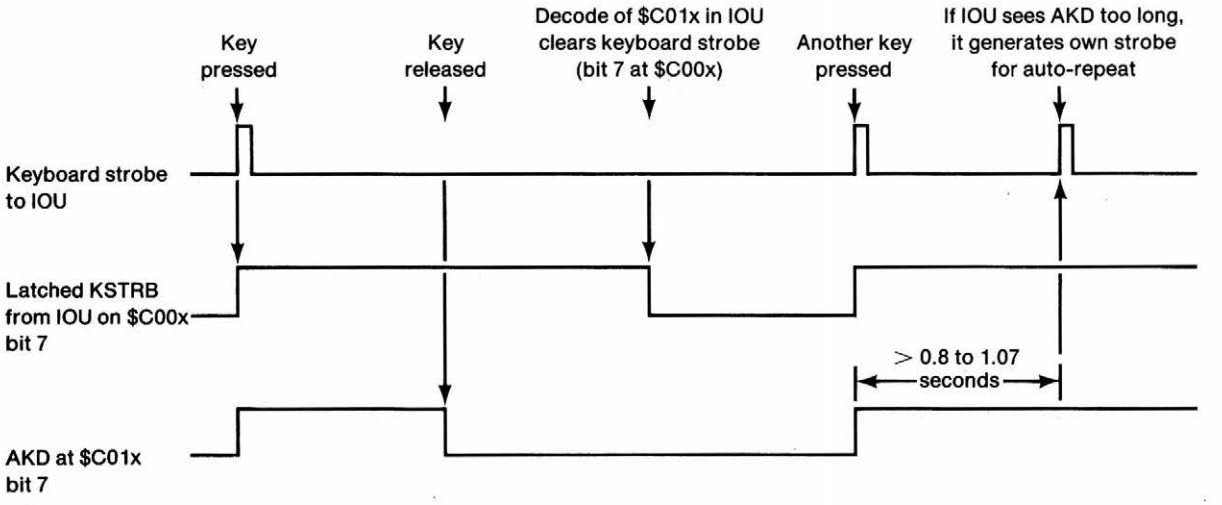


Figure 11-17
Keyboard signals

The speaker

The Apple IIc's built-in loudspeaker is controlled by a single bit of output from the input/output unit (IOU), amplified by a hybrid circuit (Figure 11-18).

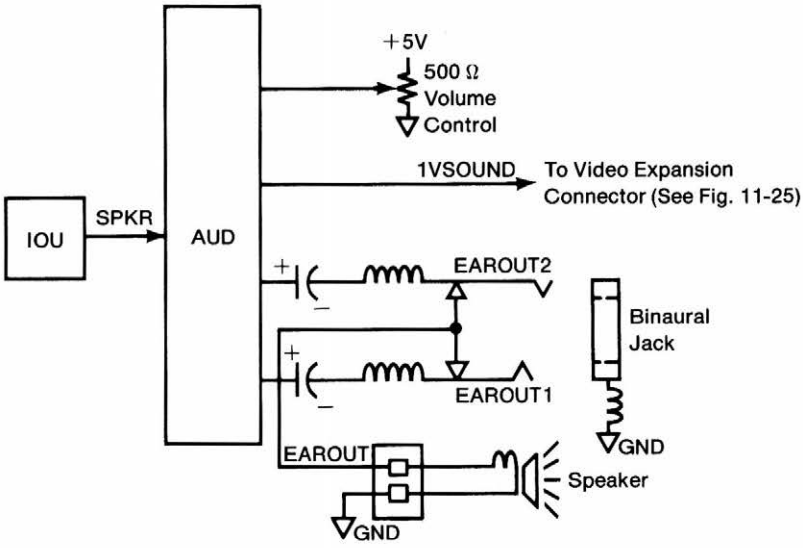


Figure 11-18
Speaker circuit diagram

Volume control

AUD is an audio-amplifier hybrid circuit.

There is a 500-ohm variable resistor feeding anywhere from 0 to 5 volts to pin 5 of **AUD** to control the speaker volume. This potentiometer controls the volume of both the built-in speaker and whatever is plugged into the output jack.

Output jack

Next to the volume control, along the lower-left side of the Apple IIc case, there is a 3.5-mm audio output jack. Although speaker output is monaural, the jack accommodates stereo headphone plugs (as well as monaural), providing sound to both channels. Inserting a headphone plug into the jack disconnects the built-in speaker.

The video display

The Apple IIc produces a video signal that creates a display on a standard video monitor or, if you add an RF modulator, on a black-and-white or color television set. The video signal is a composite made up of the data that are being displayed plus the horizontal and vertical synchronization signals that the video monitor uses to arrange the lines of display data on the screen.

❖ *Note:* Apple IIc computers manufactured for sale in the USA generate a video signal that is compatible with the standards set by the NTSC (National Television Standards Committee). Apple IIc's used in European countries require an external adapter to provide video that is compatible with the standard used there, which is called *PAL* (for phase alternating lines). References to the PAL standard are found in the bibliography at the end of this manual. This manual describes only the NTSC version of the video circuits.

The display portion of the video signal is a time-varying voltage generated from a stream of data bits, where a 1 corresponds to a voltage that generates a bright dot, and a 0 to a dark dot. The display bit stream is generated in bursts that correspond to the horizontal lines of dots on the video screen. The signal named *WNDW** is low during these bursts.

During the time intervals between bursts of data, nothing is displayed on the screen. During these intervals, called the *blanking intervals,* the display is blank and the WNDW* signal is high. The synchronization signals, called *sync* for short, are produced by making the signal named *SYNC** low during portions of the blanking intervals. The sync pulses are at a voltage equivalent to blacker-than-black video and don't show on the screen.

The video counters

The address and timing signals that control the generation of the video display are all derived from a chain of counters inside the IOU. Only a few of these counter signals are accessible from outside the IOU, but they are all important in understanding the operation of the display generation process, particularly the display memory addressing described in the next section.

The horizontal counter is made up of seven stages: H0, H1, H2, H3, H4, H5, and HPE*. The input to the horizontal counter is the 1-MHz signal that controls the reading of data being displayed. The complete cycle of the horizontal counter consists of 65 states. The six bits H0 through H5 count from 0 to 64, then start over at 0. Whenever this happens, HPE* forces another count with H0 through H5 held at 0, extending the total count to 65.

The IOU uses the 40 horizontal count values from 25 through 64 in generating the low-order part of the display data address. The IOU uses the count values from 0 to 24 to generate the horizontal blanking, the horizontal sync pulse, and the color-burst gate.

When the horizontal count gets to 65, it signals the end of a line by triggering the vertical counter. The vertical counter has nine stages: VA, VB, VC, V0, V1, V2, V3, V4, and V5. When the vertical count reaches 262, the IOU resets it and starts counting again from 0. Only the first 192 scanning lines are actually displayed; the IOU uses the vertical counts from 192 to 262 to generate the vertical blanking and sync pulse. Nothing is displayed during the vertical blanking interval. (The vertical line count is 262 rather than the standard 262.5 because, unlike normal television, the Apple IIc's video display is not interlaced.)

Display memory addressing

As described in Chapter 5, data bytes are not stored in memory in the same sequence in which they appear on the display. You can get an idea of the way the display data are stored by using the Monitor to set the display to graphics mode, then storing data starting at the beginning of the display page at hexadecimal $0400 and watching the effect on the display. If you do this, you should use the graphics display instead of text to avoid confusion: the text display is also used for Monitor input and output.

If you want your program to display data by storing them directly into the display memory, you must first transform the display coordinates into the appropriate memory addresses, as shown in Chapter 2. The descriptions that follow will help you understand how this address transformation is done and why it is necessary.

The address transformation that folds three rows of 40 display bytes into 128 contiguous memory locations is the same for all display modes, so it is described first. The differences among the different display modes are described later in this chapter.

Display address mapping

Consider the simplest display on the Apple IIc, the 40-column text mode. To address 40 columns requires 6 bits, and to address 24 rows requires another 5 bits, for a total of 11 address bits. Addressing the display this way would involve 2048 (2 to the 11th power) bytes of memory to display a mere 960 characters. The 80-column text mode would require 4096 bytes to display 1920 characters. The leftover chunks of memory that were not displayed could be used for storing other data, but not easily, because they would not be contiguous.

Instead of using the horizontal and vertical counts to address memory directly, the circuitry inside the IOU transforms them into the new address signals described below. The transformed display address must meet the following criteria:

- map the 960 bytes of 40-column text into only 1024 bytes
- scan the low-order address to refresh the dynamic RAMs
- continue to refresh the RAMs during video blanking

The requirements for RAM refreshing are discussed earlier in this chapter under "RAM addressing."

The transformation involves only horizontal counts H3, H4, and H5, and vertical counts V3 and V4. Vertical count bits VA, VB, and VC address the lines making up the characters, and are not involved in the address transformation. The remaining low-order count bits, H0, H1, H2, V0, V1, and V2 are used directly, and are not involved in the transformation.

The IOU performs an addition that reduces the five significant count bits to four new signals S0, S1, S2, and S3, where S stands for sum. Figure 11-19 is a diagram showing the addition in binary form, with V3 appearing as the carry in and H5 appearing as its complement H5*. A constant value of 1 appears as the low-order bit of the addend. The carry bit generated with the sum is not used.

If this transformation seems terribly obscure, try it with actual values. For example, for the upper-left corner of the display, the vertical count is 0 and the horizontal count is 24: H0, H1, H2, and H5 are 0's, and H3 and H4 are 1's. The value of the sum is 0, so the memory location for the first character on the display is the first location in the display page, as you might expect.

Table 11-14
Display memory addressing

Memory address bit	Display address bit
A0	H0
A1	H1
A2	H2
A3	S0
A4	S1
A5	S2
A6	S3
A7	V0
A8	V1
A9	V2
A10	*
A11	*
A12	*
A13	*
A14	*
A15	GND

* For these address bits, see Table 11-15.

			V3	Carry in
H5*	V3	H4	H3	Augend
V4	H5*	V4	1	Addend
S3	S2	S1	S0	Sum

Figure 11-19
Display address transformation

Horizontal bits H0, H1, and H2 and sum bits S0, S1, and S2 make up the transformed horizontal address (A0 through A6 in Table 11-14). As the horizontal count increases from 24 to 63, the value of the sum (S3 S2 S1 S0) increases from 0 to 4 and the transformed address goes from 0 to 39, relative to the beginning of the display page.

The low-order three bits of the vertical row counter are V0, V1, and V2. These bits control address bits A7, A8, and A9, as shown in Table 11-14, so that rows 0 through 7 start on 128-byte boundaries. When the vertical row counter reaches 8, V0, V1, and V2 are 0 again, and V3 changes to 1. If you do the addition in Figure 11-19 with H equal to 24 (the horizontal count for the first column displayed) and V equal to 8, the sum is 5 and the horizontal address is 40: the first character in row 8 is stored in the memory location 40 bytes from the beginning of the display page.

Table 11-14 shows how the signals from the video counters are assigned to the address lines. H0, H1, and H2 are horizontal-count bits, and V0, V1, and V2 are vertical-count bits. S0, S1, S2, and S3 are the folded address bits described above. Table 11-15 shows memory address bits for the display modes.

Table 11-15
Memory address bits for display modes

Address bit	Text and low resolution	High resolution and double high resolution
A10	80STORE+PAGE2'	VA
A11	80STORE'.PAGE2	VB
A12	0	VC
A13	0	80STORE+PAGE2'
A14	0	80STORE'.PAGE2

Note: Period (.) means logical AND; prime (') means logical NOT.

Figure 11-20 shows how groups of three 40-character rows are stored in blocks of 120 contiguous bytes starting on 128-byte address boundaries. This diagram is another way of describing the display mapping shown in Figure 5-5. Notice that the three rows in each block of 120 bytes are not adjacent on the display.

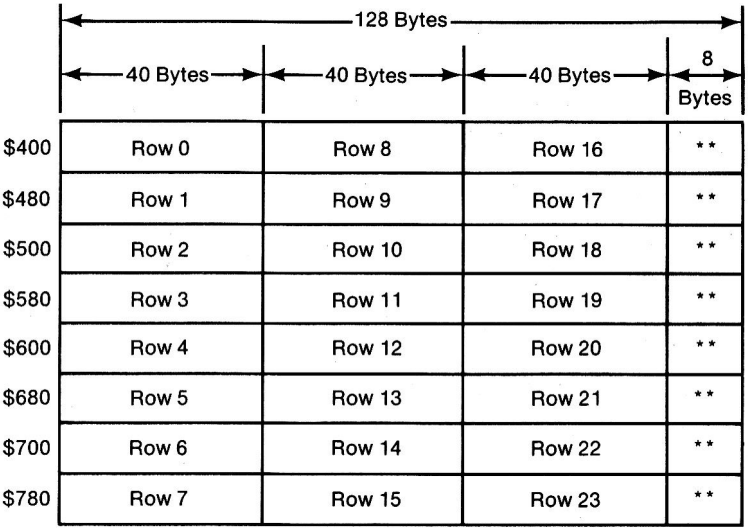


Figure 11-20
40-column text display memory (memory locations marked with a double asterisk ** are screen holes, described in Chapter 2)

Video display modes

The different display modes all use the address-mapping scheme described later in this chapter, but they use different-sized memory areas in different locations. This section describes the addressing schemes and the methods of generating the actual video signals for the different display modes. Figure 11-21 illustrates the video display circuits discussed in this section.

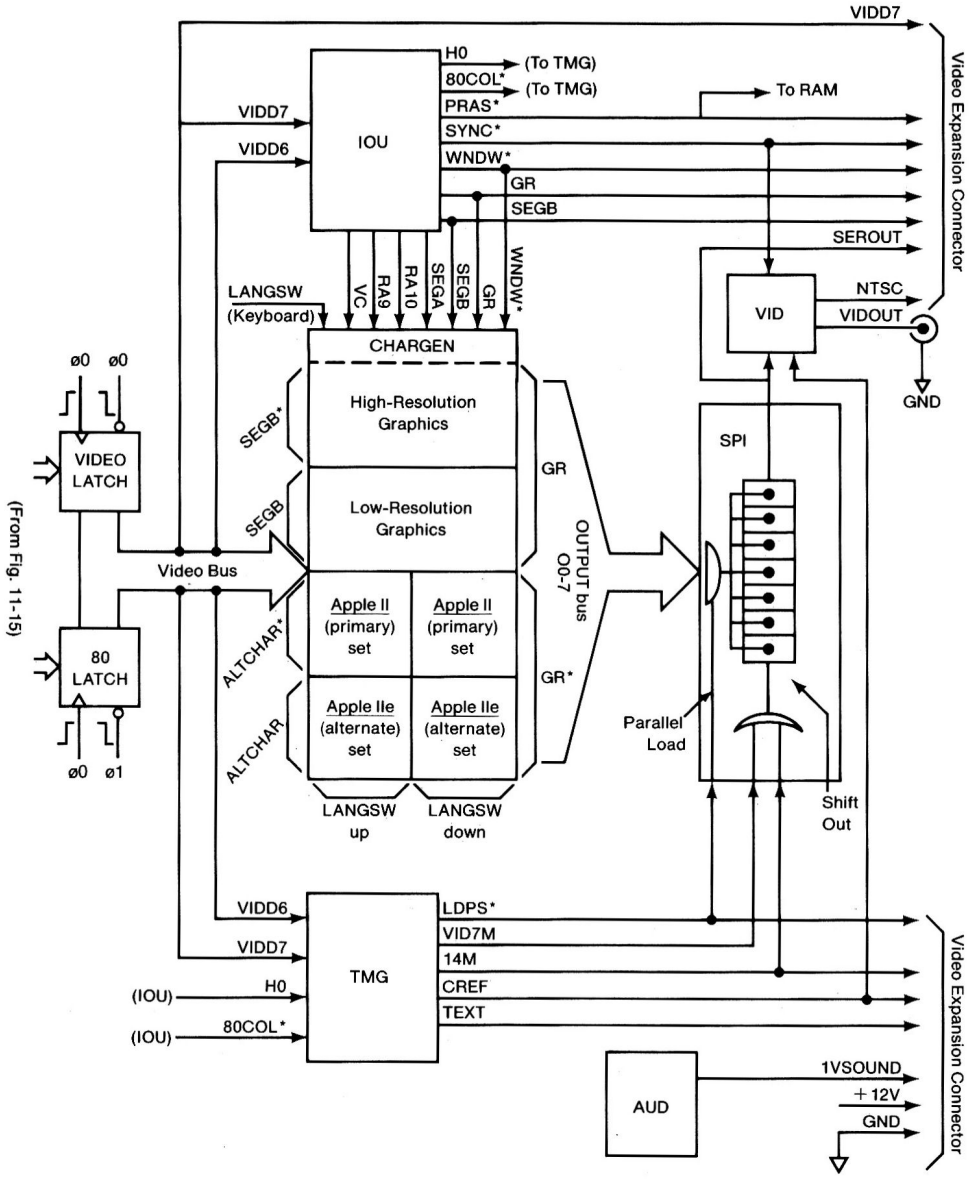


Figure 11-21
Video display circuits

Text displays

The text and low-resolution graphics pages begin at memory locations $0400 and $0800. Table 11-15 shows how the display-mode signals control the address bits to produce these addresses. Address bits A10 and A11 are controlled by the settings of Page2 and 80Store, the display-page and 80-column-video soft switches. Address bits A12, A13, and A14 are set to 0. Notice that 80Store active inhibits Page2: there is only one display page in 80-column mode.

The low-order six bits of each data byte reach the character generator directly, via the video data bus VID0–VID5. The two high-order bits are modified by the IOU to select between the primary and alternate character sets and are sent to the character generator on lines RA9 and RA10.

The data for each row of characters are read eight times, once for each of the eight lines of dots making up the row of characters. The data bits are sent to the character generator along with VA, VB, and VC, the low-order bits from the vertical counter. For each character being displayed, the character generator puts out one of eight stored bit patterns selected by the three-bit number made up of VA, VB, and VC.

The bit patterns from the character generator are loaded into the 74166 parallel-to-serial shift register and output as a serial bit stream that goes to the video output circuit (Figure 11-21). The shift register is controlled by signals named LDPS* (for load parallel-to-serial shifter) and VID7M (for video 7 MHz). In 40-column mode, LDPS* strobes the output of the character generator into the shift register once each microsecond, and VID7M shifts the bits out at 7 MHz (Figure 11-22).

The addressing for the 80-column display is exactly the same as for the 40-column display: the 40 columns of display memory in auxiliary memory are addressed in parallel with the 40 columns in main memory. The data from these two memories reach the video data bus (lines VID0–VID7) via separate 74LS374 three-state buffers. These buffers are loaded simultaneously (at the rising edge of ø0), but their outputs are sent to the character generator alternately by the falling edge of ø0 and ø1. In 80-column mode, LDPS* loads data from the character generator into the shift register twice during each microsecond, once during ø0 and once during ø1, and VID7M remains low, enabling the clock continuously at 14M (Figure 11-23).

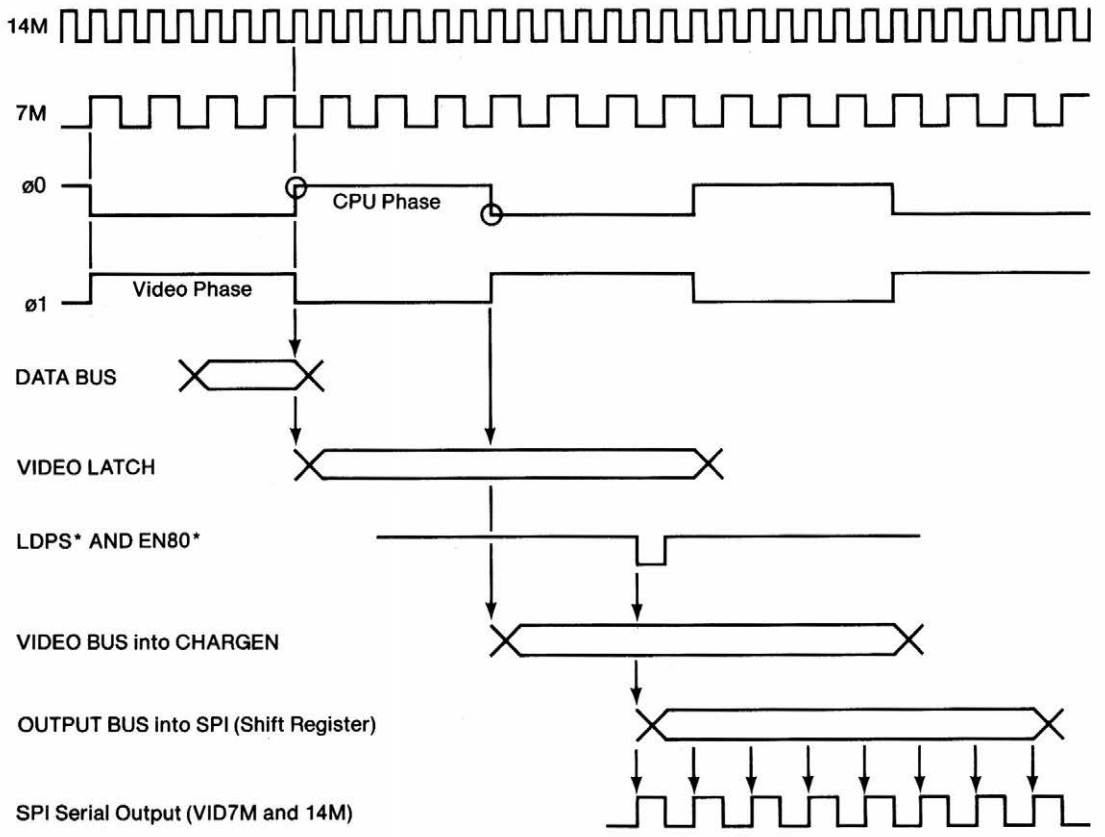


Figure 11-22
7-MHz video timing signals: 40-column, low-resolution, and high-resolution display

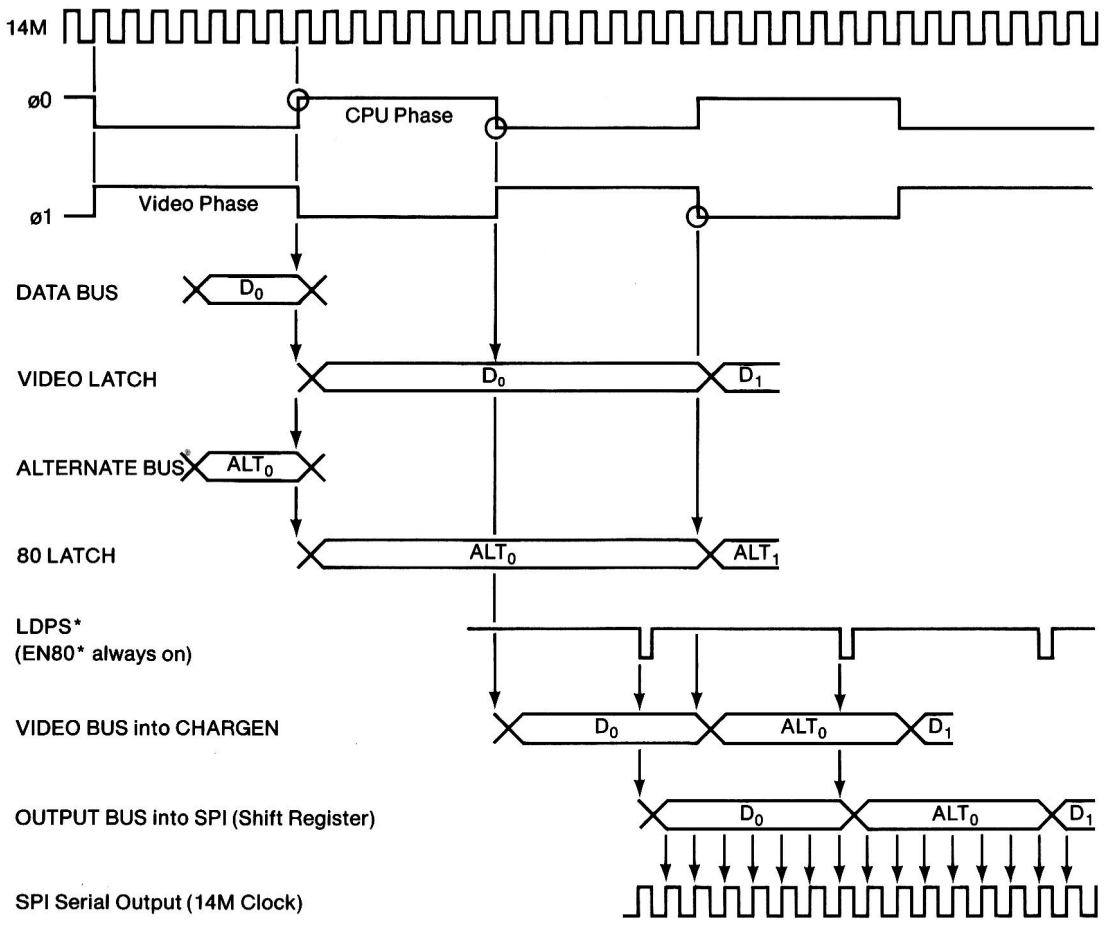


Figure 11-23
14-MHz video timing signals: 80-column and double high-resolution display

Low-resolution display

In the graphics modes, VA and VB are not used by the character generator, so the IOU uses lines SEGA and SEGB to transmit H0 and HIRES*, as shown in Table 11-16.

Table 11-16
Character-generator control signals

Display mode	SEGA	SEGB	SEGC
Text	VA	VB	VC
Graphics	H0	HIRES*	VC

The low-resolution graphics display uses VC to divide the eight display lines corresponding to a row of characters into two groups of four lines each. Each row of data bytes is addressed eight times, the same as in text mode, but each byte is interpreted as two nibbles. Each nibble selects 1 of 16 colors. During the upper four of the eight display lines, VC is low and the low-order nibble determines the color. During the lower four display lines, VC is high and the high-order nibble determines the color.

The bit patterns that produce the low-resolution colors are read from the character-generator ROM in the same way the bit patterns for characters are produced in text mode. The 74166 parallel-to-serial shift register converts the bit patterns to a serial bit stream for the video circuits (Figure 11-21).

The video signal generated by the Apple IIc includes a short burst of 3.58-MHz signal that is used by an NTSC color monitor or color TV set to generate a reference 3.58-MHz color signal. The Apple IIc's video signal produces color by interacting with this 3.58-MHz signal inside the monitor or TV set. Different bit patterns produce different colors by changing the duty cycles and delays of the bit stream relative to the 3.58-MHz color signal. To produce the small delays required for so many different colors, the shift register runs at 14 MHz and shifts out 14 bits during each cycle of the 1-MHz data clock. To generate a stream of 14 bits from each 8-bit pattern read from the ROM, the output of the shift register is connected back to the register's serial input to repeat the same 8 bits; the last 2 bits are ignored the second time around.

Each bit pattern is output for the same amount of time as a character: 1.02 microseconds. Because that is exactly enough time for three and a half cycles of the 3.58-MHz color signal, the phase relationship between the bit patterns and the signal changes by a half cycle for each successive pattern. To compensate for this, the character generator puts out one of two different bit patterns for each nibble, depending on the state of H0, the low-order bit of the horizontal counter.

High-resolution display

The high-resolution graphics pages begin at memory locations $2000 and $4000 (decimal 8192 and 16384). These page addresses are selected by address bits A13 and A14. In high-resolution mode, these address bits are controlled by PAGE2 and 80STORE, the signals controlled by the display-page (Page2) and 80-column-video (80Col) soft switches. As in text mode, 80STORE inhibits addressing of the second page because there is only one page of 80-column text available for mixed mode.

In high-resolution graphics mode, the display data are still stored in blocks like the one shown in Figure 11-20, but there are eight of these blocks. As Tables 11-14 and 11-15 show, vertical counts VA, VB, and VC are used for address bits A10, A11, and A12, which address eight blocks of 1024 bytes each. Remember that in the display, VA, VB, and VC count adjacent horizontal lines in groups of eight. This addressing scheme maps each of those lines into a different 1024-byte block.

It might help to think of this scheme as a kind of eight-way multiplexer: it's as if eight text displays were combined to produce a single high-resolution display, with each text display providing one line of dots in turn, instead of a row of characters.

The high-resolution bit patterns are produced by the character-generator ROM. In this mode, the bit patterns simply reproduce the seven bits of display data. The low-order six bits of data reach the ROM via the video data bus VID0–VID5. The IOU sends the other two data bits to the ROM via RA9 and RA10.

The high-resolution colors described in Chapter 2 are produced by the interaction between the video signal the bit patterns generate and the 3.58-MHz color signal generated inside the monitor or TV set. The high-resolution bit patterns are always shifted out at 7 MHz, so each dot corresponds to a half-cycle of the 3.58-MHz color signal. Any part of the video signal that produces a single white dot between two black dots, or vice versa, is effectively a short burst of 3.58 MHz and is therefore displayed as color. In other words, a bit pattern consisting of alternating 1's and 0's gets displayed as a line of color. The high-resolution graphics subroutines produce the appropriate bit patterns by masking the data bits with alternating 1's and 0's.

To produce different colors, the bit patterns must have different phase relationships to the 3.58-MHz color signal. If alternating 1s and 0's produce a certain color, say green, then reversing the pattern to 0's and 1's will produce the complementary color, purple. As in the low-resolution mode, each bit pattern corresponds to three and a half cycles of the color signal, so the phase relationship between the data bits and the color signal changes by a half cycle for each successive byte of data. Here, however, the bit patterns produced by the hardware are the same for adjacent bytes; the color compensation is performed by the high-resolution software, which uses different color masks for data being displayed in even and odd columns.

To produce other colors, bit patterns must have other timing relationships to the 3.58-MHz color signal. In high-resolution mode, the Apple IIc produces two more colors by delaying the output of the shift register by half a dot (70 ns), depending on the high-order bit of the data byte being displayed. (The high-order bit doesn't actually get displayed as a dot, because at 7 MHz there is only time to shift out seven of the eight bits.)

As each byte of data is sent from the character generator to the shift register, high-order data bit D7 is also sent to the TMG. If D7 is off, the TMG transmits shift-register timing signals LDPS* and VID7M normally. If D7 is on, the TMG delays LDPS* and VID7M by 70 nanoseconds, the time corresponding to half a dot. The bit pattern that formerly produced green now produces orange; the pattern for purple now produces blue.

- *A note about timing:* For 80-column text, the shift register is clocked at twice normal speed. When 80-column text is used with graphics in mixed mode, the TMG controls shift-register timing signals LDPS* and VID7M so that the graphics portion of the display works correctly even when the text window is in 80-column mode.

Double high-resolution display

Double high-resolution graphics mode displays two bytes in the time normally required for one, but it uses high-resolution graphics Pages 1 and 1X instead of text and low-resolution Pages 1 and 1X.

❖ *Note:* There is a second pair of bytes, HRP2 and HRP2X, which can be used to display a second double high-resolution page.

Double high-resolution graphics mode displays each pair of data bytes as 14 adjacent dots, 7 from each byte. The high-order bit (color-select bit) of each byte is ignored. The auxiliary-memory byte is displayed first, so data from auxiliary memory appear in columns 0–6, 14–20, and so on, up to columns 547–552. Data from main memory appear in columns 7–13, 21–27, and so on, up to 553–559.

As in 80-column text, there are twice as many dots across the display screen, so the dots are only half as wide. On a TV set or low-bandwidth (less than 14 MHz) monitor, single dots are dimmer than normal.

RGB stands for red, green, and blue, and identifies a type of color monitor that uses independent inputs for the three primary colors.

❖ *Note:* Except for some expensive **RGB**-type color monitors, any video monitor with a bandwidth as high as 14 MHz will be a monochrome monitor. Monochrome means one color: a monochrome video monitor can have a screen color of white, green, orange, or any other single color.

The main memory and auxiliary memory are connected to the address bus in parallel, so both are activated during the display cycle. The rising edge of ø0 clocks a byte of main memory data into the video latch, and a byte of auxiliary memory data into the 80 latch (Figure 11-21).

Phi 1 enables output from the (auxiliary) 80 latch, and ø0 enables output from the (main) video latch. Output from both latches goes to CHARGEN, where GR and SEGB* select high-resolution graphics. LDPS operates at 2 MHz in this mode, alternately gating the auxiliary byte and main byte into the parallel-to-serial shift register. VID7M is active (kept true) for double high-resolution display mode, so when it is ANDed with 14M, the result is still 14M. The 14M serial clock signal gates shift register output to VID, the video display hybrid circuit, for output to the display device.

For further information about double high-resolution graphics display, refer to the Bibliography.

Video output signals

VID is a video-amplifier hybrid circuit.

The stream of video data generated by the display circuits described above goes to a hybrid circuit (**VID**) that adjusts the signals to the proper amplitudes and conditions the color burst.

The resulting video signal is an NTSC-compatible composite-video signal that can be displayed on a standard video monitor. The signal is similar to the EIA (Electronic Industries Association) standard positive composite video. This signal is available in two places in the Apple IIc (Figure 11-24):

- at the video output connector on the back of the Apple IIc
- at the video expansion connector (pin 12) on the back panel (Table 11-17)

Monitor output

The sleeve of the video output connector at the center of the Apple IIc back panel is connected to ground and the tip is connected to the video output through a resistor network that attenuates it to about 1 volt and matches its impedance to 75 ohms. This arrangement is suitable for most video monitors.

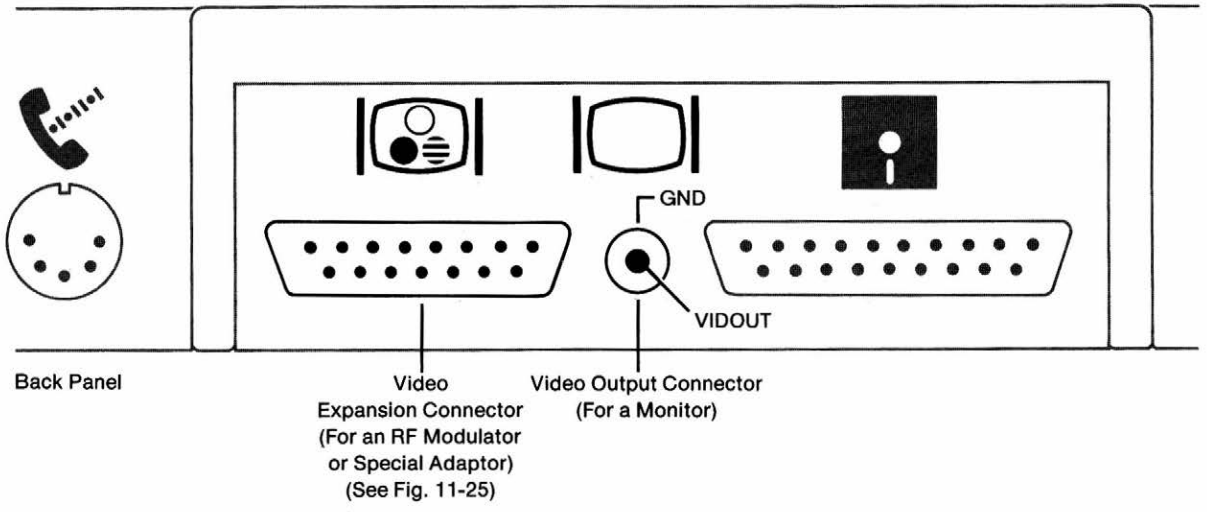


Figure 11-24
Video output back panel connectors

Video expansion output

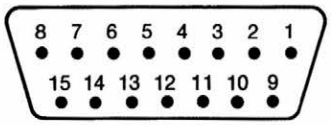
The back panel of the Apple IIc has a DB-15 connector for sophisticated video interfaces external to the computer. Figure 11-25 shows the pin assignments for this connector; Table 11-17 describes the signals. In Table 11-17, the column labeled *Deriv* indicates what clock signals the video signals are derived from. LDPS, CREF, and PRAS have a maximum delay of 30 ns from the appropriate 14-MHz rising edge. SEROUT is clocked out of a 74LS166 by the rising edge of 14M and has a maximum delay of 35 ns. VIDD7 is driven from a 74LS374 and has a maximum delay of 28 ns from the rising and (if 80-column) falling edges of ø1.

To align CREF so it is in the same phase at the beginning of every line, certain clock signals must be stretched. This stretch is for one 7M cycle (140 ns), and occurs at the end of each video line. All timing signals except 14M, 7M, and CREF are stretched.

Warning The maximum allowable current drain of +12V regulated power at the video expansion connector is 300 milliamps. If the external device draws more than this, it can damage the computer or cause the power supply to shut down.

Warning The signals at the DB-15 on the Apple IIc are not the same as those at the DB-15 on the Apple III. Do not attempt to plug a cable intended for one into the other.

Several of these signals, such as 14 MHz, must be buffered within about 4 inches (10 cm) of the back panel connector—preferably inside a container directly connected to the back panel. For technical information, contact Apple Technical Support.



Pin	Signal	Pin	Signal
1	TEXT	9	PRAS*
2	14M	10	GR
3	SYNC*	11	SEROUT*
4	SEGB	12	NTSC
5	1VSOUND	13	GND
6	LDPS*	14	VIDD7
7	WNDW*	15	CREF
8	+12V		

Figure 11-25
Video expansion connector pinouts

Table 11-17
Video expansion connector signals

Pin	Deriv	Signal	Description
1	ø0	TEXT	Video text signal from TMG; set to inverse of GR, except in double high-resolution mode
2		14M	14-MHz master timing signal from the system oscillator
3	Q3	SYNC*	Displays horizontal and vertical synchronization signal from IOU pin 39
4	PRAS	SEGB	Displays vertical counter bit from IOU pin 4; in text mode indicates second low-order vertical counter; in graphics mode indicates low-resolution
5		1VSOUND	One-volt sound signal from pin 5 of the audio hybrid circuit (AUD)
6	14M	LDPS*	Video shift-register load enable from pin 12 of TMG
7	PRAS	WNDW*	Active area display blanking; includes both horizontal and vertical blanking

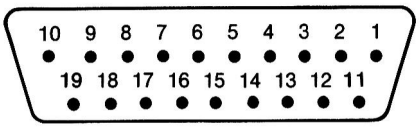
Table 11-17 (continued)
Video expansion connector signals

Pin	Deriv	Signal	Description
8		+12V	Regulated +12 volts DC; can drive 300 mA
9	14M	PRAS*	RAM row-address strobe from TMG pin 19
10	PRAS	GR	Graphics mode enable from IOU pin 2
11	14M	SEROUT*	Serialized character-generator output from pin 1 of the 74LS166 shift register
12		NTSC	Composite NTSC video signal from VID hybrid chip
13		GND	Ground reference and supply
14	ø0	VIDD7	From 74LS374 video latch; causes half-dot shift if high
15	14M	CREF	Color reference signal from TMG pin 3; 3.58 MHz

Disk I/O

Disk I/O—for both the built-in and the external drive—is supported by the IWM disk controller unit. The external drive is attached via a DB-19 connector. Figure 11-26 shows this connector. Table 11-18 describes the pin assignments. Supply voltages come from the power supply; all other signals come from the IWM, described earlier in this chapter.

Warning The power available at this connector is for a Disk IIc or similar drive only. Do not use power from the external disk connector for any other purpose—you may damage the internal voltage converter. To derive external power for an attached device, use one of the other connectors and observe the current limits given in this manual.



Pin	Signal	Pin	Signal
1,2,3,4	GND	13	SEEKPH2
5	−12V	14	SEEKPH3
6	+5V	15	WRREQ*
7,8	+12V	16	N.C.
9	EXTINT*	17	DR2*
10	WRPROT	18	RDDATA
11	SEEKPH0	19	WRDATA
12	SEEKPH1		

Figure 11-26
Disk drive connector

Table 11-18
Disk drive connector signals

Connector pin	Signal	Description
1,2,3,4	GND	Ground reference and supply
6	+5V	+5 volt supply
7,8	+12	+12 volt supply
9	EXTINT*	External interrupt
10	WRPROT	Write-protect input
11–14	ø0–4	Motor phase 0–4 output
15	WRREQ*	Write request
17	DR1*	Drive 1 select
18	RDDATA	Read data input
19	WRDATA	Write data output

Serial I/O

The Apple IIc has built into it two 6551 asynchronous communication interface adapters (ACIA) and supporting input and output buffers for full-duplex serial communication. Figure 11-27 is a block diagram of the Apple IIc serial ports. ACIA outputs are buffered by a 1448-quad line driver. Similarly, ACIA inputs are buffered by a 1489-quad line receiver.

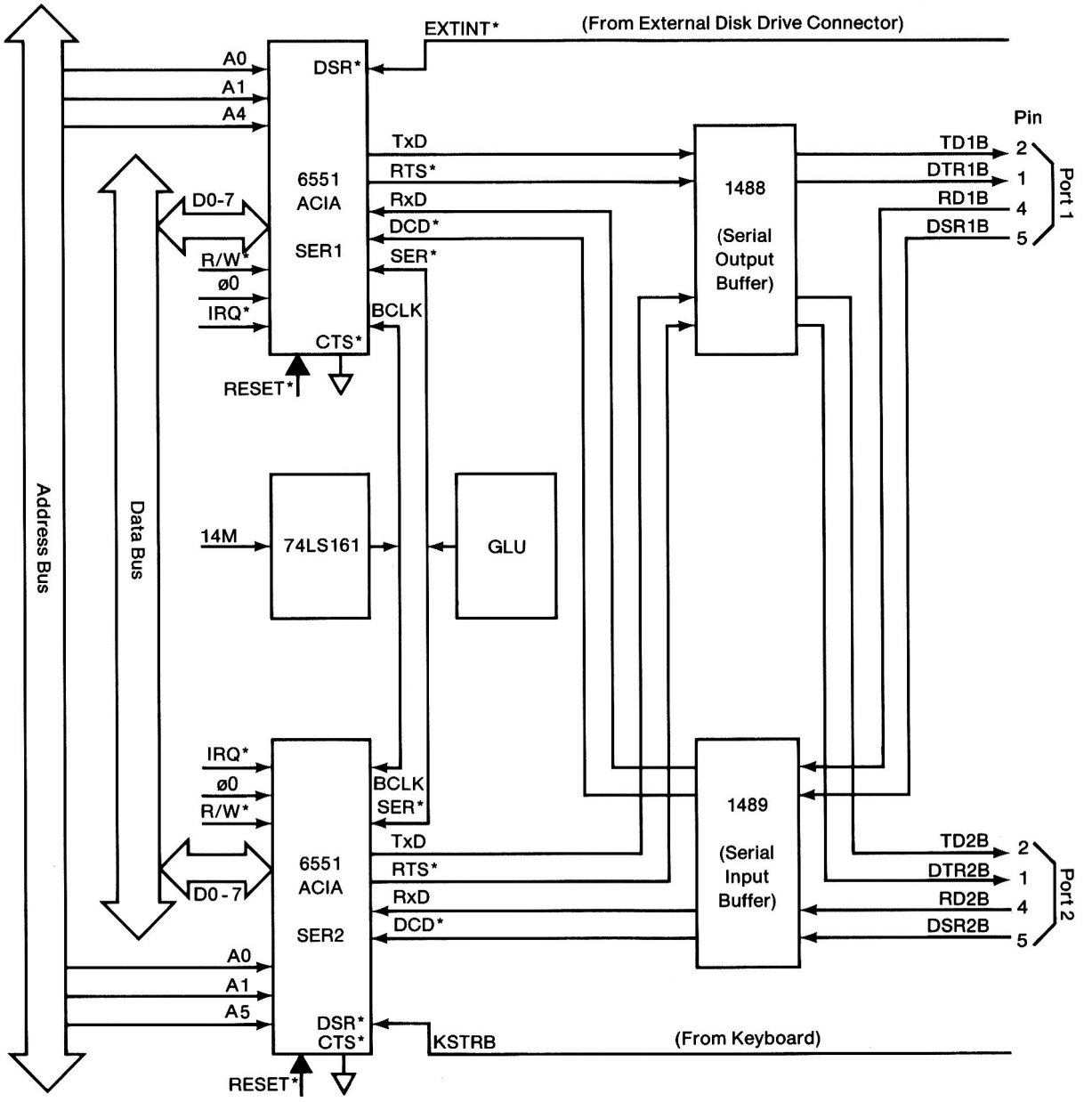


Figure 11-27
Serial port circuits

Figure 11-28 is a detailed block diagram of the 6551 ACIA. The registers are described later in this chapter.

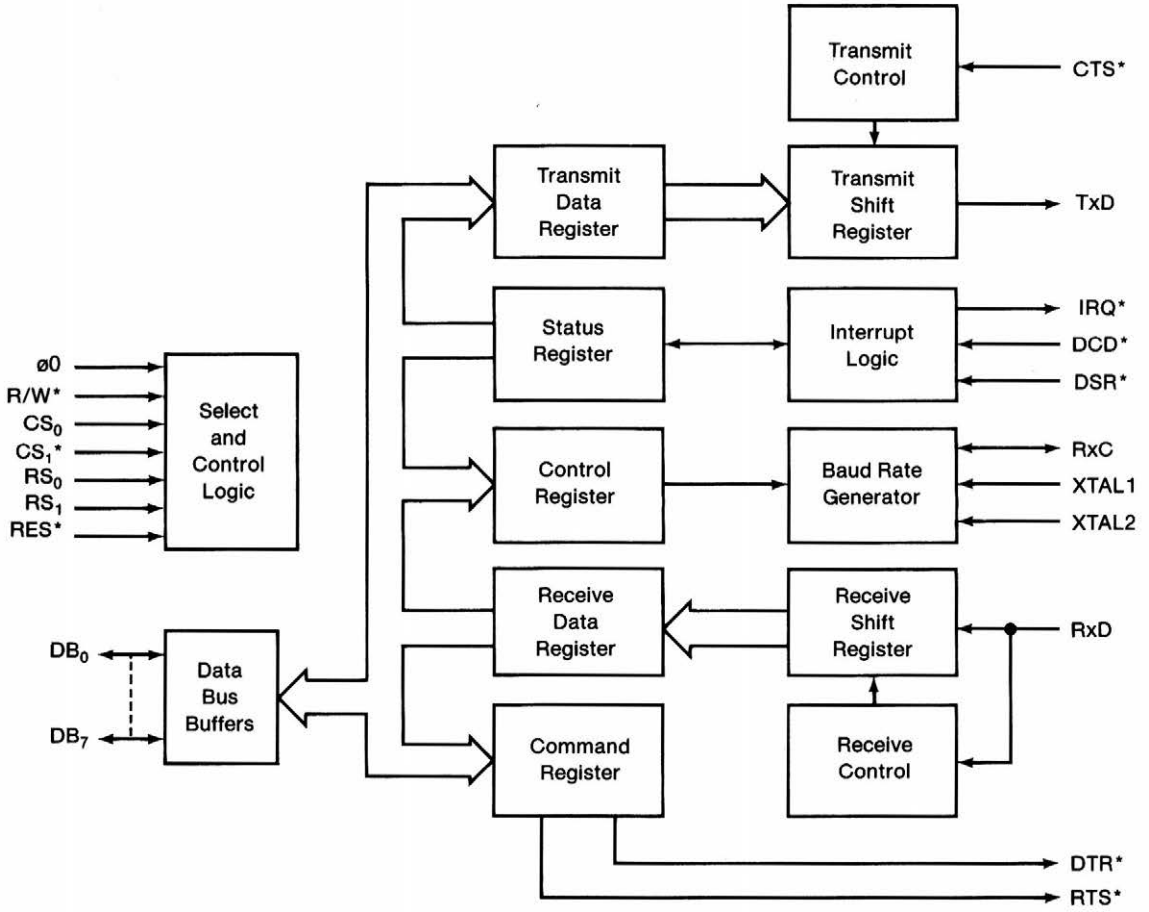


Figure 11-28
6551 ACIA block diagram (copyright © 1978 by Synertek Inc.; used by permission)

The 6551 pin assignments are shown in Figure 11-29 and described in Table 11-19. Note that the two 6551's are not used in exactly the same way—each one supports a different set of interrupts.

Port 1 reads external interrupts (EXTINT*) on its Data Set Ready (DSR) pin. This input is tied to +5V through a 3.3-KΩ pullup resistor.

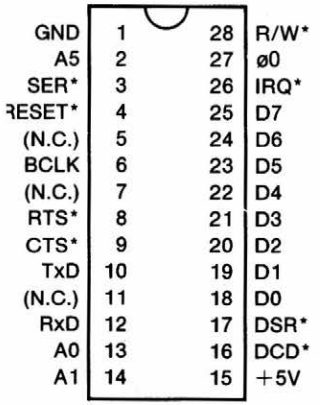
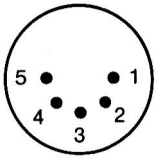


Figure 11-29
6551 pinouts

Table 11-19
6551 signal descriptions

Pin	Signal	Description
1	GND	Power and signal common ground
2	A4 A5	Address line 4 to select serial port 1 Address line 5 to select serial port 2
3	SER*	Serial device select from GLU
4	RESET*	Resets both serial ports
5	N.C.	Not connected
6	BCLK	Baud rate clock from GLU
7	N.C.	Not connected
8	RTS*	Request to Send output
9	CTS*	Clear to Send input (not used on IIc; tied to ground)
10	TXD	Transmit Data output
11	N.C.	Not connected
12	RXD	Receive Data input
13,14	A0,A1	Address lines 0 and 1
15	+5V	+5 volt supply
16	DSR	DCD* pin; used on IIc as Data Set Ready input
17	 EXTINT* KSTRB	DSR*pin; used on IIc as External interrupt (port 1 ACIA), or Keyboard strobe input (port 2 ACIA; Appendix E)
18–25	D0–D7	8-bit data bus
26	IRQ*	Interrupt Request input
27	ø0	Phase 0 clock pulse
28	R/W*	Read/write select input



Pin	Port 1	Port 2
1	DTR1B	DTR2B
2	TD1B	TD2B
3	GND	GND
4	RD1B	RD2B
5	DSR1B	DSR2B

Figure 11-30
Serial port connectors

The back panel connectors for both serial ports are 5-pin DIN jacks. The pin assignments are shown in Figure 11-30 and described in Table 11-20.

Table 11-20
Serial port connector signals

Pin	Signal	Description
1	DTR1B DTR2B	Data Terminal Ready output
2	TD1B TD2B	Transmit Data output
3	GND	Power and signal common
4	RD1B RD2B	Read Data input
5	DSR1B DSR2B	Data Set Ready input

ACIA control register

Figure 11-31 shows the bit assignments for the ACIA control register, which the hardware locates at address $C09B for serial port 1, and $C0AB for serial port 2. This register determines the number of data and stop bits the ACIA will transmit and receive, and the clock source and baud rate to use for data transfer.

The receiver clock source is derived from the Apple IIc's TMG chip; the resulting baud rates are equal to or up to two percent lower than the nominal rate. (The EIA standard allows plus or minus two percent variation.) If an Apple IIc serial port is used with a modem that is two percent above the nominal rate, framing errors can occur, especially at 1200 baud and above, when using 8 data bits. It may be necessary to select a lower baud rate for 8-bit binary data transfers.

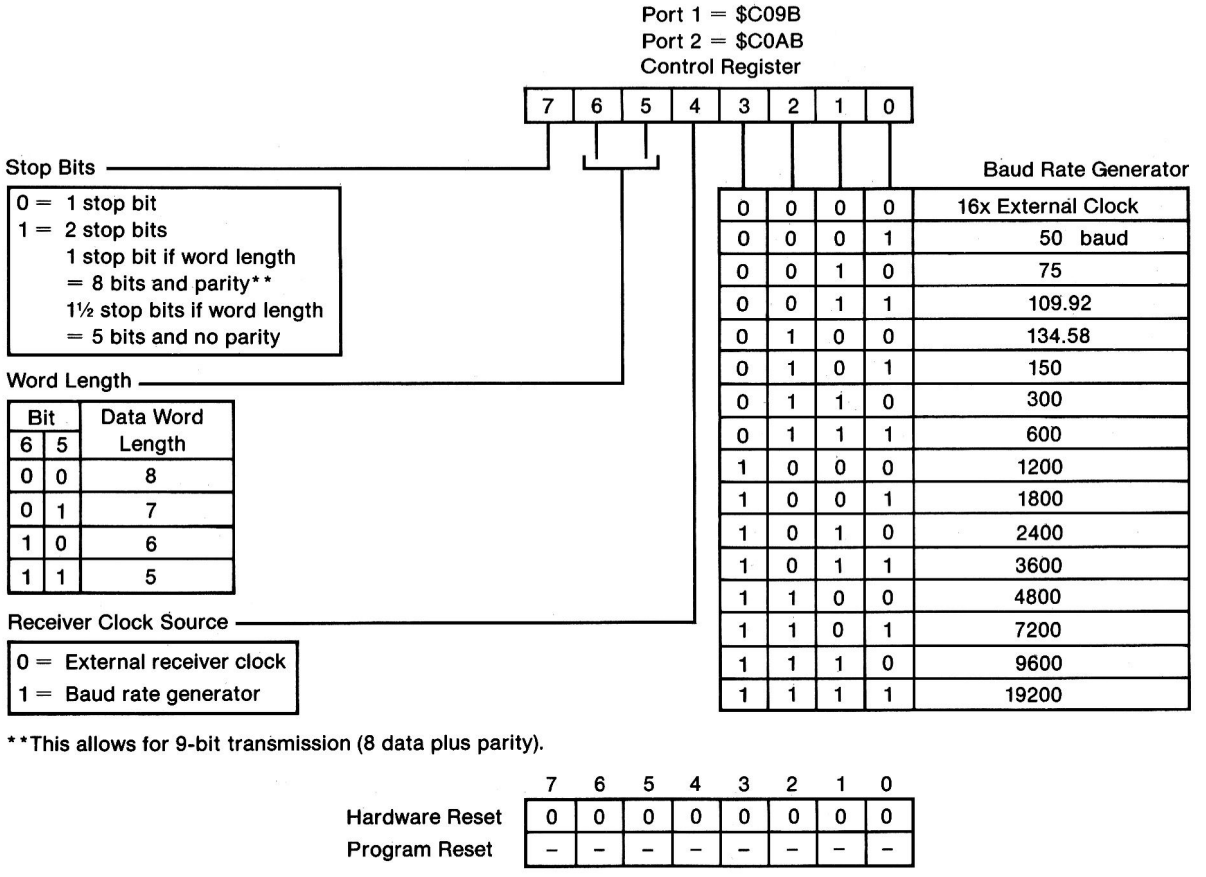
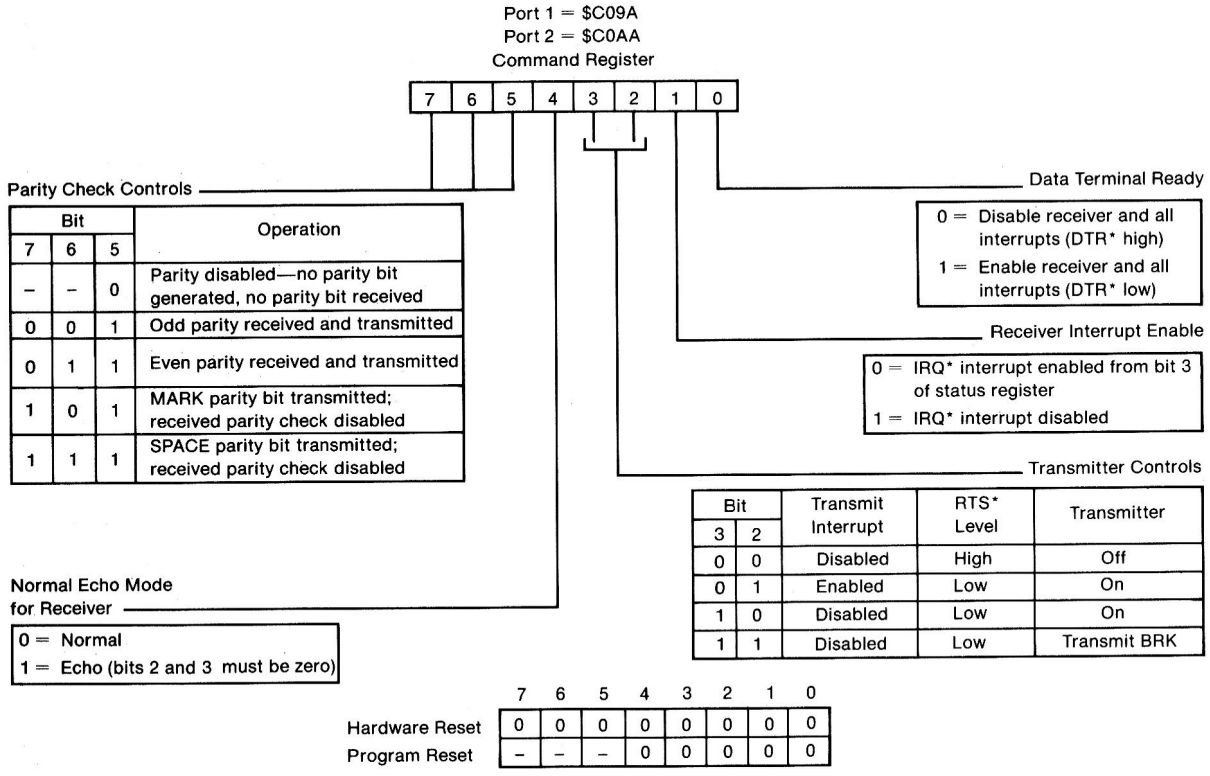


Figure 11-31
ACIA control register (copyright © 1978 by Synertek Inc.; used by permission)

ACIA command register

Figure 11-32 shows the bit assignments for the ACIA command register, which the hardware locates at address $C09A for serial port 1, and at $C0AA for serial port 2. This register controls specific transmit and receive functions: parity checking, echoing input to output, allowing transmit and receive interrupts, and setting levels for Data Terminal Ready and Request to Send.



Bit 7	Bit 6	Bit 5	Operation
–	–	0	Parity disabled—no parity bit generated, no parity bit received
0	0	1	Odd parity received and transmitted
0	1	1	Even parity received and transmitted
1	0	1	MARK parity bit transmitted; received parity check disabled
1	1	1	SPACE parity bit transmitted; received parity check disabled

Bit 3	Bit 2	Transmit Interrupt	RTS* Level	Transmitter
0	0	Disabled	High	Off
0	1	Enabled	Low	On
1	0	Disabled	Low	On
1	1	Disabled	Low	Transmit BRK

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	–	–	–	0	0	0	0	0

Figure 11-32
ACIA command register (copyright © 1978 by Synertek Inc.; used by permission)

ACIA status register

Figure 11-33 shows the bit assignments for the ACIA status register, which is hard-wired to address $C099 for serial port 1, and $C0A9 for serial port 2. This register reports the condition of the transmit/receive register, errors detected during data transfer, and the level of the Data Carrier Detect, Data Set Ready, and Interrupt Request lines.



Port 1 = $C099
Port 2 = $C0A9

Status	Set By	Cleared By
Parity error†	0 = No error 1 = Error	Self-clearing**
Framing error†	0 = No error 1 = Error	Self-clearing**
Overrun†	0 = No error 1 = Error	Self-clearing**
Receive Data, Register full	0 = Not full 1 = Full	Reading receive data register
Transmit Data, Register empty	0 = Not empty 1 = Empty	Writing to transmit data register
DCD*	0 = DCD* low 1 = DCD* high	Not resettable; reflects DCD* state
DSR*	0 = DSR* low 1 = DSR* high	Not resettable; reflects DSR* state
IRQ	0 = No interrupt 1 = Interrupt	Reading status register

† No interrupt generated for these conditions.
** Cleared automatically after a read of RDR and the next error-free receipt of data.

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	–	–	–	–	–	–	–	–

Figure 11-33
ACIA status register (copyright © 1978 by Synertek Inc.; used by permission)

ACIA transmit/receive register

Each ACIA uses the same address—$C098 for serial port 1, $C0A8 for serial port 2—as temporary storage for both transmission and reception of data.

When the register is used for transmitting data, bit 0 is the leading bit to be transmitted; unused data bits are the high-order bits, which are ignored.

When the register is used for receiving data, bit 0 is the first bit received; unused data bits are the high-order bits, which are set to 0. Parity bits never appear in the receive data register; they are stripped off after being used for external parity checking.

Mouse input

The mouse is a hand-held X-Y pointing device that can be rolled along a flat surface. It has an attached pushbutton. This section describes how mouse movement and direction can be detected and interpreted.

A mouse has a ball inside its housing that protrudes a small distance so that its turning corresponds to mouse movements across a table top. Two wheels inside the housing, set at 90-degree angles to each other, follow movements of the ball; this causes two disks to rotate. The disks have rectangular holes arranged near their edges, making them resemble circular slide mounts used with stereoscopic slide viewers.

The light from a tiny infrared emitter reaches a photoreceptor whenever one of the holes on the disk lies between them. An internal circuit in the mouse causes the resulting voltage to swing quickly to a 1 or a 0 value as soon as a certain threshold is crossed. The result is something approximating a square wave (Figure 11-34) that varies directly with the speed of mouse movement. One of these indicates the X component (X0) of mouse movement; the other, the Y component (Y0).

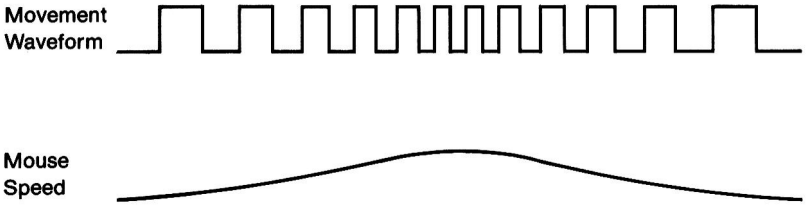


Figure 11-34
Sample mouse waveform

Under program control, either the rising edge or the falling edge of each square wave can cause an interrupt, which the firmware handles by updating a counter. However, the program needs to know whether to add or to subtract 1 from a counter; that is, it needs to know the direction of X or Y movement.

There is a second infrared emitter/photoreceptor pair almost 180 degrees opposite the first pair for each disk. These pairs are positioned in such a way that the square waves they generate are approximately a quarter-wave offset from their respective movement waves (Figure 11-35). These waveforms are called *X1* (X direction) and *Y1* (Y direction).

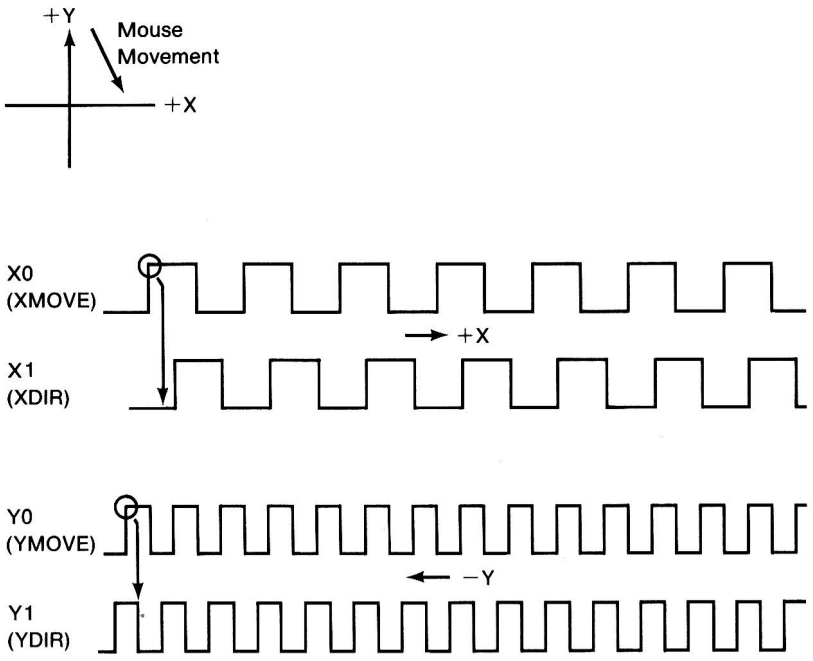
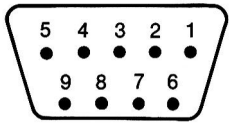


Figure 11-35
Mouse movement and direction waveforms



Pin	Signal
1	MOUSEID*
2	+5V
3	GND
4	XDIR
5	XMOVE
6	(N.C.)
7	MSW*
8	YDIR
9	YMOVE

Figure 11-36
Mouse connector

When a rising edge of X0 causes an interrupt, a mouse-driver program can immediately check whether X1 is 0 (indicating a movement to the right) or 1 (indicating a movement to the left). Similarly, the mouse driver can read Y1 immediately after a Y0 interrupt to determine whether the mouse moved up or down one count along the Y axis.

Figure 11-36 shows the pin assignments for the mouse DB-9 connector on the back panel. Table 11-21 gives the signal names and descriptions.

Table 11-21
Mouse connector signals

Pin	Signal	Description
1	MOUSEID*	Mouse identifier: when active, disables NE556 hand controller timer
2	+5V	Total current drain from this pin must not exceed 100 mA
3	GND	System ground
4	XDIR	Mouse X-direction indicator
5	XMOVE	Mouse X-movement interrupt
6	N.C.	Not connected
7	MSW*	Mouse button
8	YDIR	Mouse Y-direction indicator
9	YMOVE	Mouse Y-movement interrupt

Figure 11-37 shows the mouse and hand controller circuitry with the mouse circuits emphasized. Figure 11-38 illustrates the values of the mouse-button circuit when the button is pressed or not pressed. Pressing the button disables the NE556 by pulling the reset comparator threshold value up so that it cannot reset the flip flop. As a result the mouse-button input value remains at a TTL level.

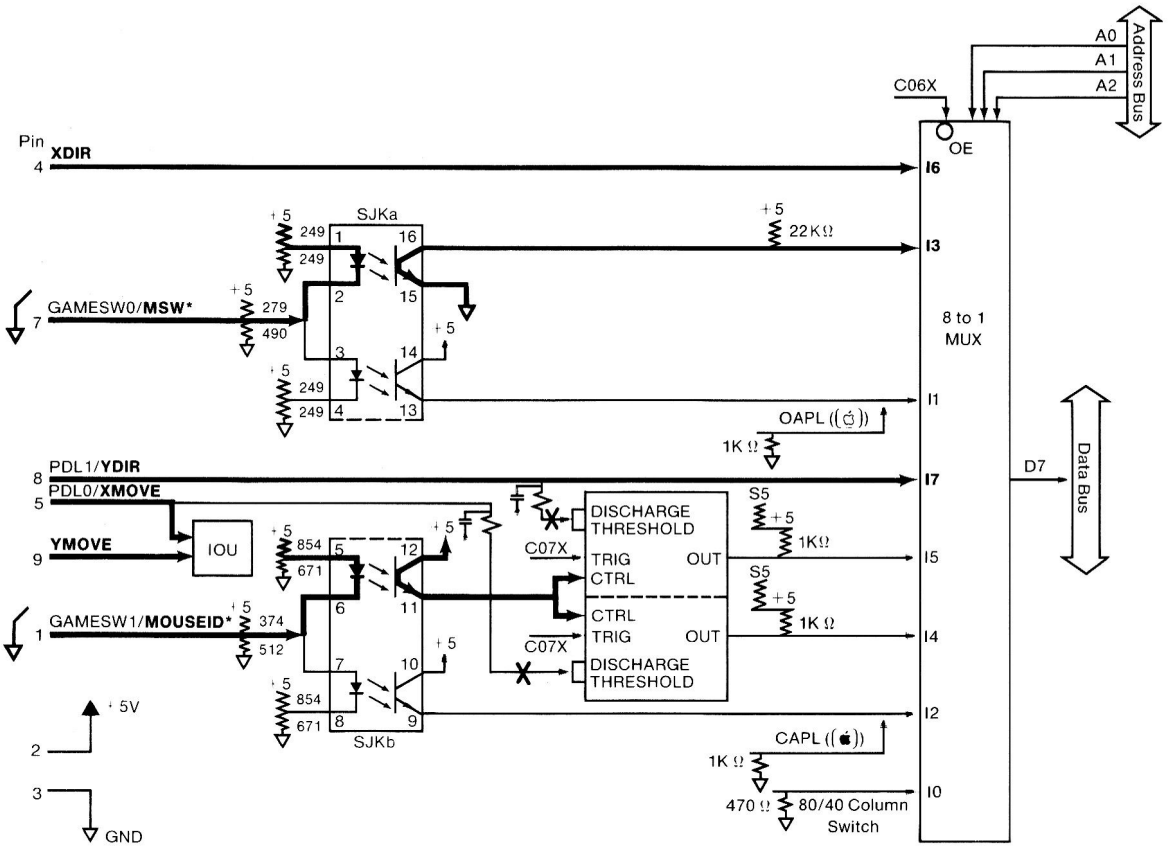


Figure 11-37
Mouse circuits

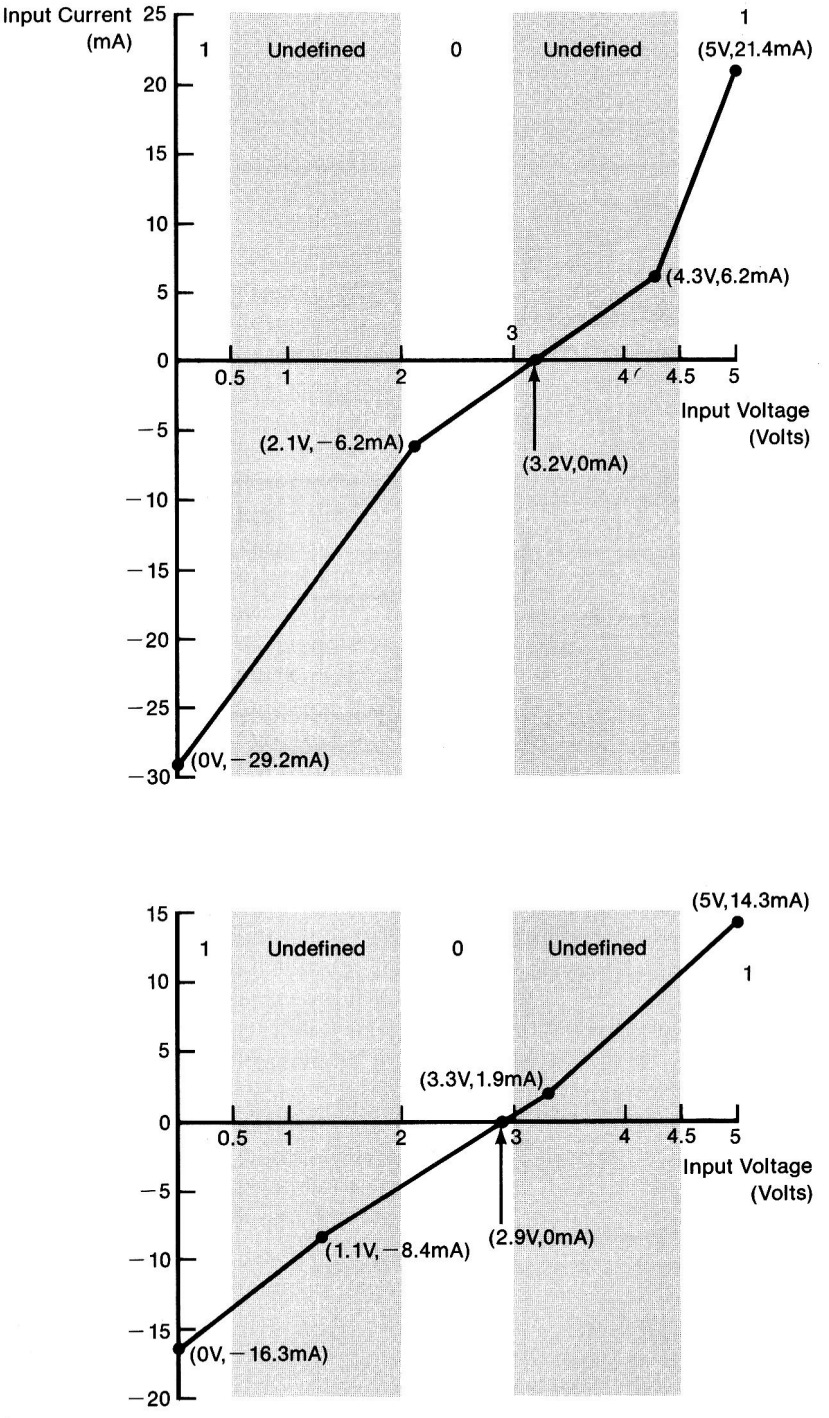
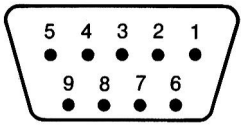


Figure 11-38
Mouse button signals



Pin	Signal
1	GAMESW1
2	+5V
3	GND
4	Not used for hand controllers
5	PDL0
6	(N.C.)
7	GAMESW0
8	PDL1
9	Not used for hand controllers

Figure 11-39
Hand controller connector

Hand controller input

Several input signals that are individually controlled via soft switches are collectively referred to as the **hand controller** (game) signals. These signals arrive in the Apple IIc via the same DB-9 connector as the one used for the mouse, but the Apple IIc interprets these signals differently.

The DB-9 connector pin assignments and signal descriptions, as used for hand controller input, appear in Figure 11-39 and Table 11-22.

Even though they are normally used for hand controllers, these signals can be used for other simple I/O applications. There are two 1-bit switch inputs, labeled *Sw0* and *Sw1,* and two analog inputs, called *paddles* and labeled *Pdl0* and *Pdl1*. Figure 11-40 shows how to connect the 1-bit switch inputs for compatibility with all other Apple II series computers.

The switch inputs are multiplexed by a 74LS251 8-to-1 multiplexer enabled by the C06X* signal from the MMU. Depending on the low-order address, the appropriate game input is connected to bit 7 of the data bus. Figure 11-41 shows the mouse and hand controller circuitry with the hand controller circuits highlighted. Figure 11-42 illustrates the values of the hand controller switch inputs when the switch is open or closed.

Table 11-22
Hand controller connector signals

Pin	Signal	Description
1	GAMESW1	Switch input 1 (sometimes called *paddle button 1*).
2	+5V	+5V power supply; total current drain from this pin must not exceed 100 mA.
3	GND	System ground.
4,9		Not used for hand controllers.
5,8	PDL0 and PDL1	Hand controller inputs; each of these must be connected to a 150-KΩ variable resistor connected to +5V.
6	N.C.	Not connected.
7	GAMESW0	Switch input 0 (sometimes called *paddle button 0*).

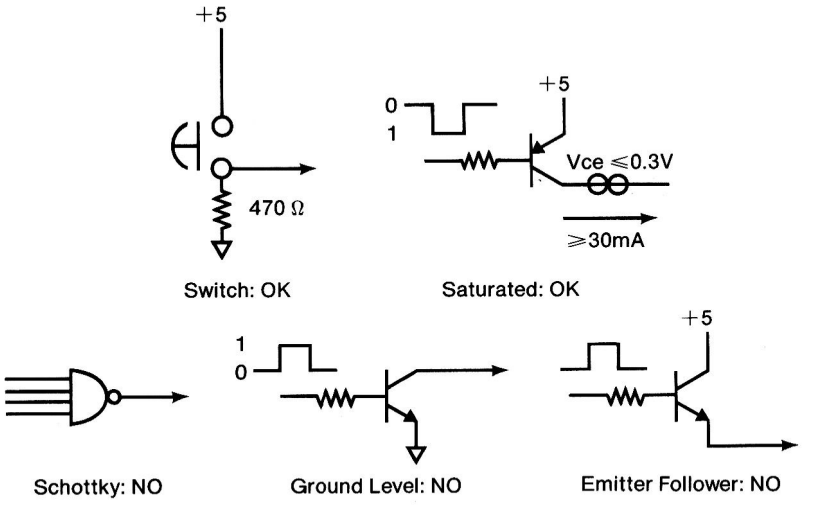


Figure 11-40
How to connect switch inputs

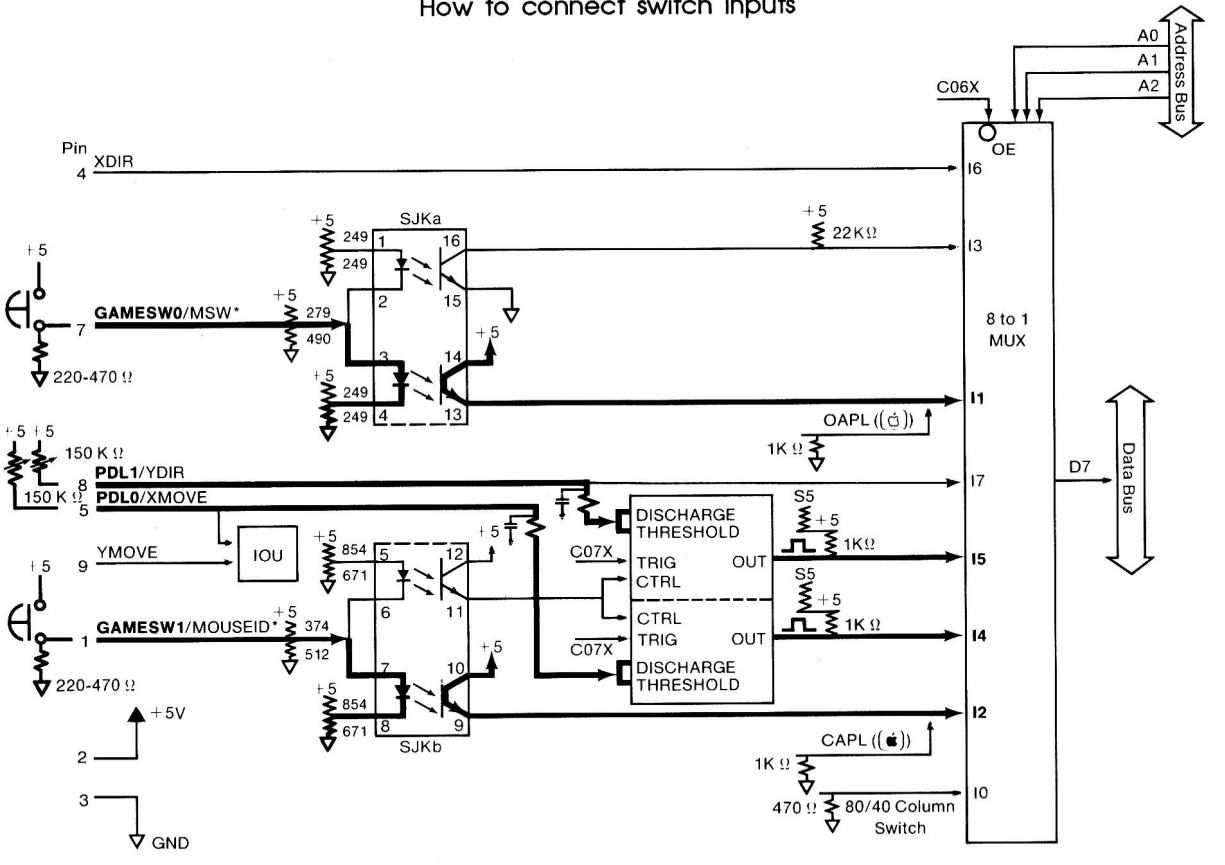


Figure 11-41
Hand controller circuits

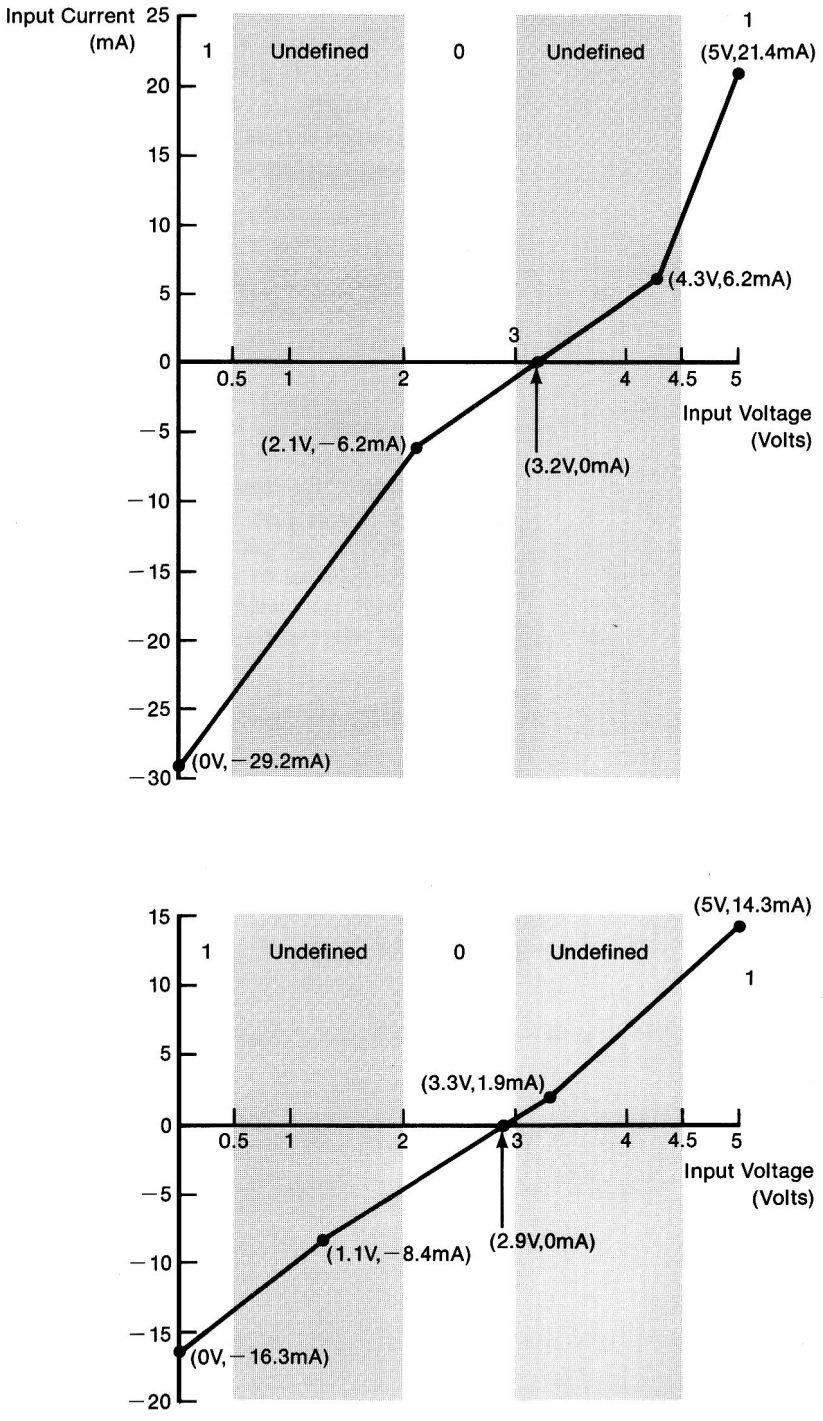


Figure 11-42
Hand controller signals

The hand controller inputs are connected to the timing inputs of an NE556 dual analog timer. Addressing $C07X sends a signal from MMU pin 22 that resets both timers and causes their outputs to go to 1 (high). A variable resistance of up to 150 KΩ connected between one of these inputs and the +5V supply controls the charging time of one of the two 0.022 microfarad capacitors.

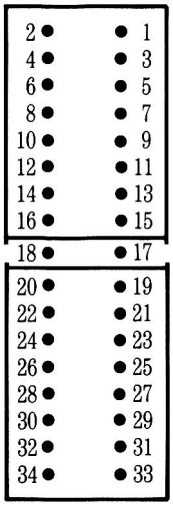
When the voltage on the capacitor passes a certain threshold, the output of the NE556 changes back to 0 (low). Programs can determine the setting of a variable resistor by resetting the timers and then counting time until the selected timer input changes from high to low. The resulting count is proportional to the resistance.

Warning The only way to ensure correct paddle values is to make sure the output of the paddle you intend to read is low before you trigger the timer. Triggering the timer starts the charging cycle for the capacitor in each paddle circuit; the cycle for one may not be completed by the time you have read the other. If you retrigger or read the other paddle too soon (that is, in less than 3 ms), you will get a false value for it.

Memory expansion card

Memory expansion card I/O is supported by an internal connector mounted on the main logic board. Figure 11-43 is a pinout diagram for this connector.

For information on the Apple IIc Memory Expansion Card, refer to the *Apple IIc Memory Expansion Card Reference.*



Pin	Signal	Pin	Signal
1	D0	18	A9
2	D1	19	A10
3	D2	20	A11
4	D3	21	A12
5	D4	22	A13
6	D5	23	A15
7	D6	24	$\overline{\text{A15}}$
8	D7	25	$\overline{\text{RESET}}$
9	GND	26	$\overline{\text{RW}}$
10	GND	27	+5V
11	A0	28	+5V
12	A1	29	PHO
13	A4	30	GND
14	A5	31	7M
15	A6	32	GND
16	A7	33	Q3
17	A8	34	+5V

Figure 11-43
Memory expansion card connector pinout diagram

Schematic diagrams

Figure 11-44, on the following pages, is a set of schematic diagrams for the Apple IIc.

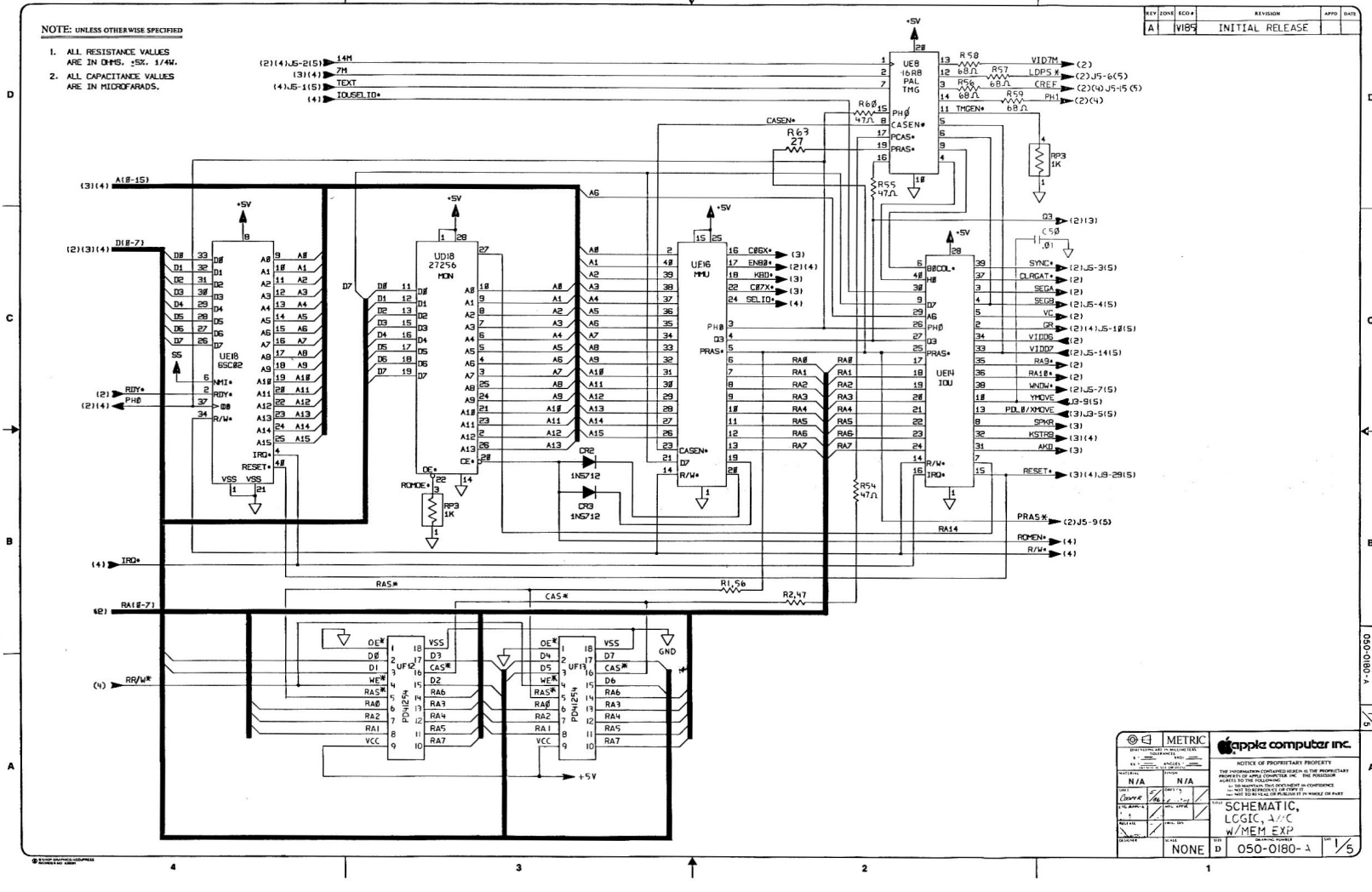


Figure 11-44a
Apple IIc schematic diagram, part 1

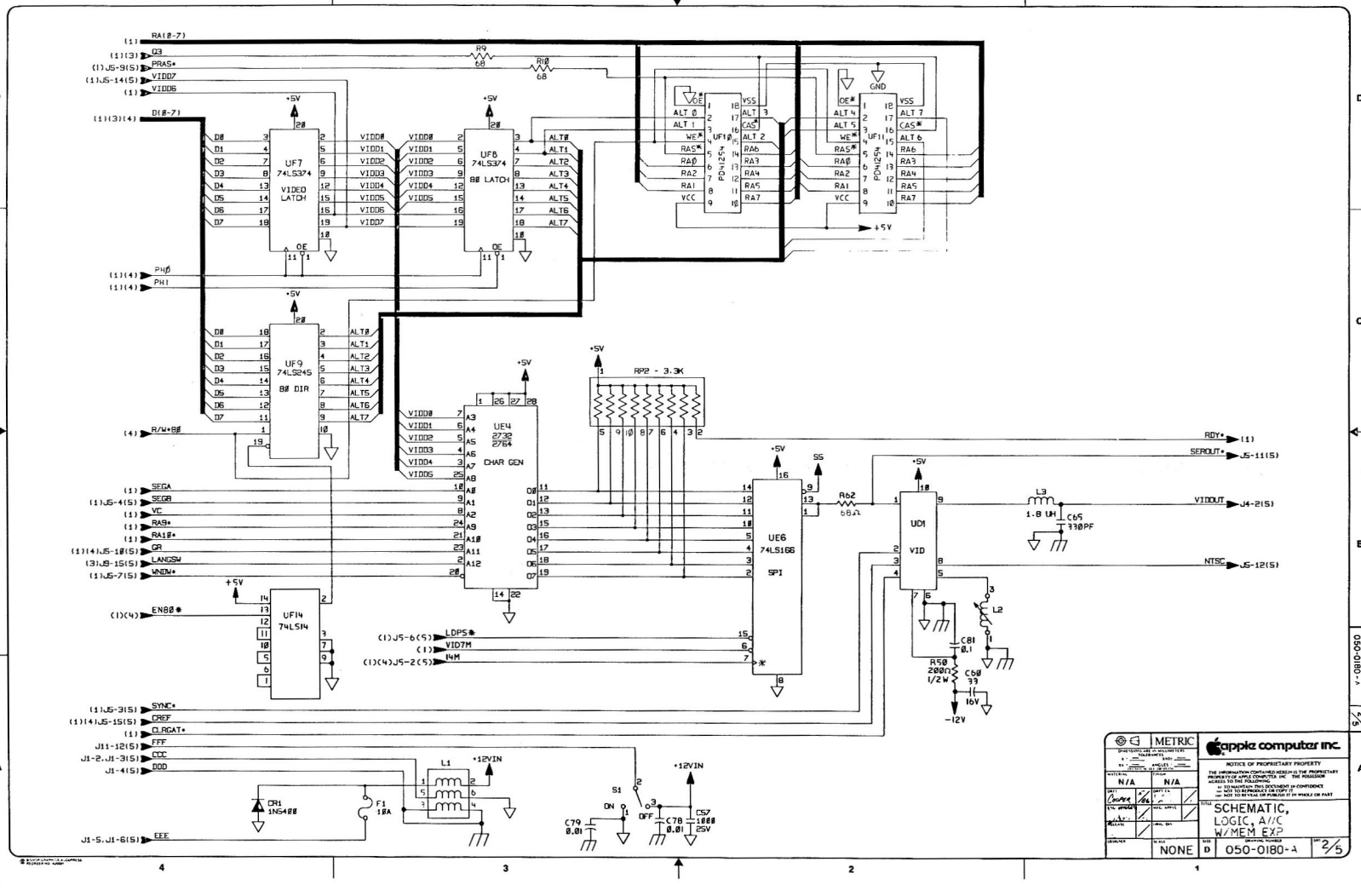


Figure 11-44b
Apple IIc schematic diagram, part 2

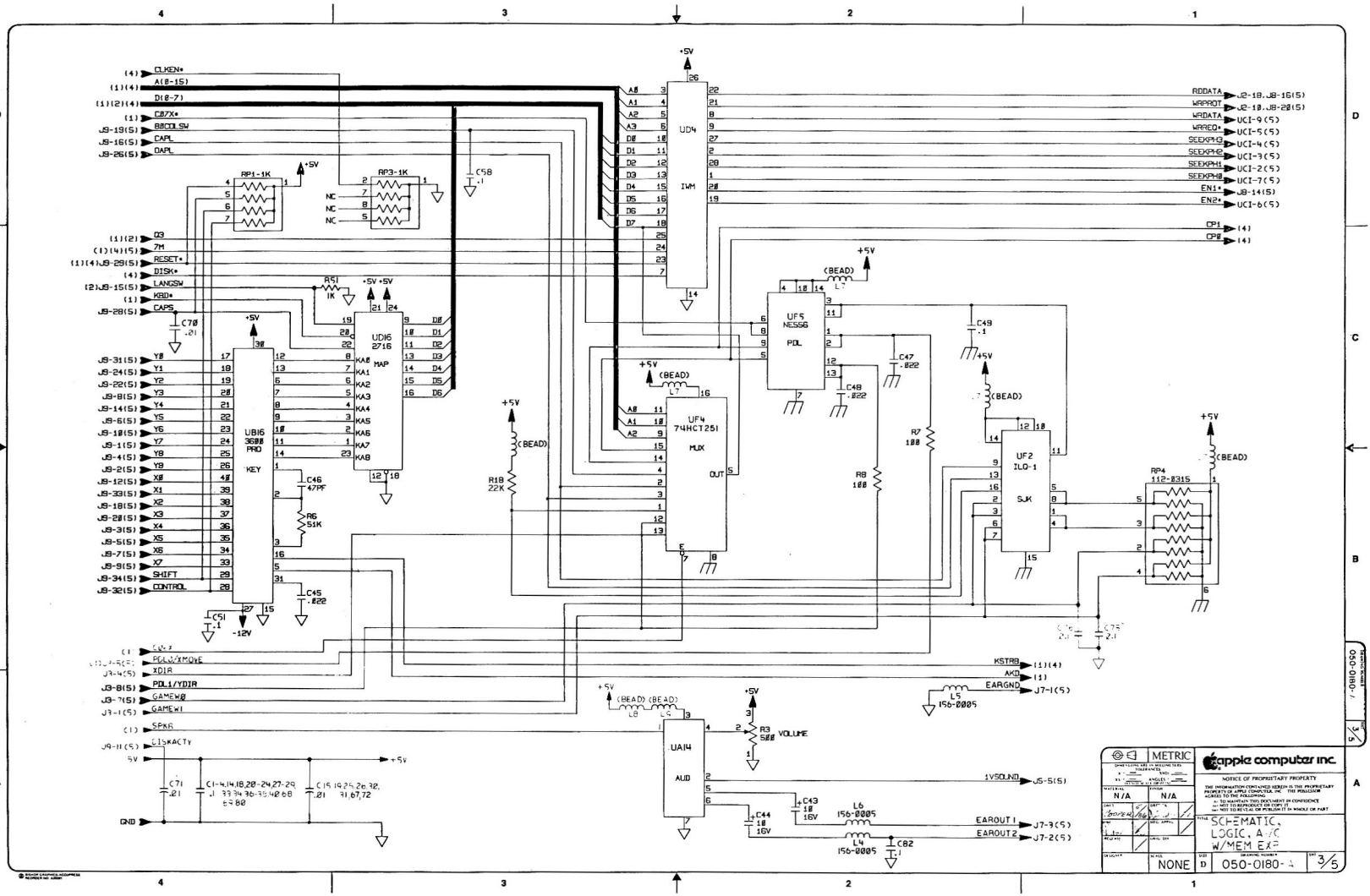


Figure 11-44c
Apple IIc schematic diagram, part 3

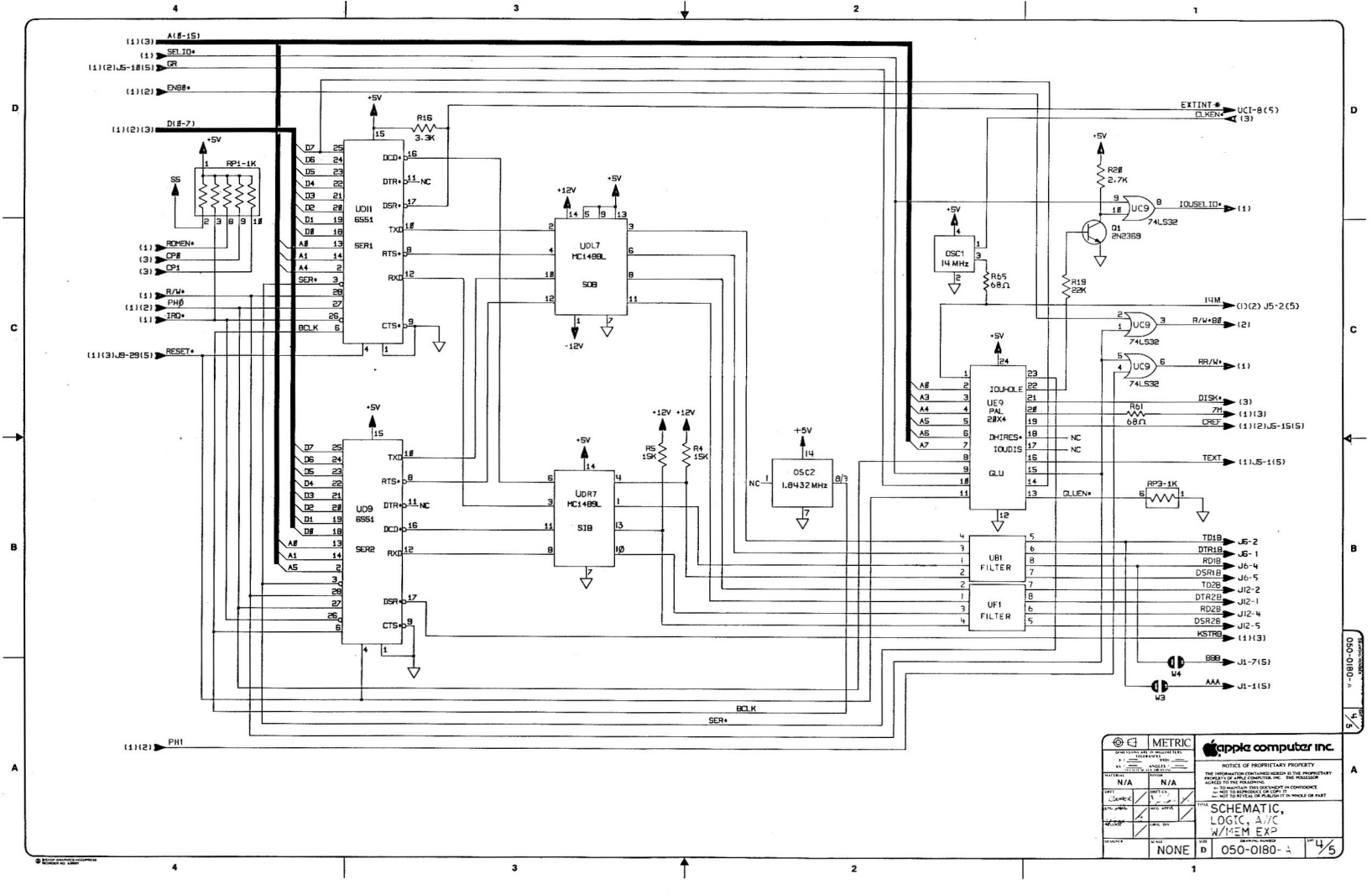


Figure 11-44d
Apple IIc schematic diagram, part 4

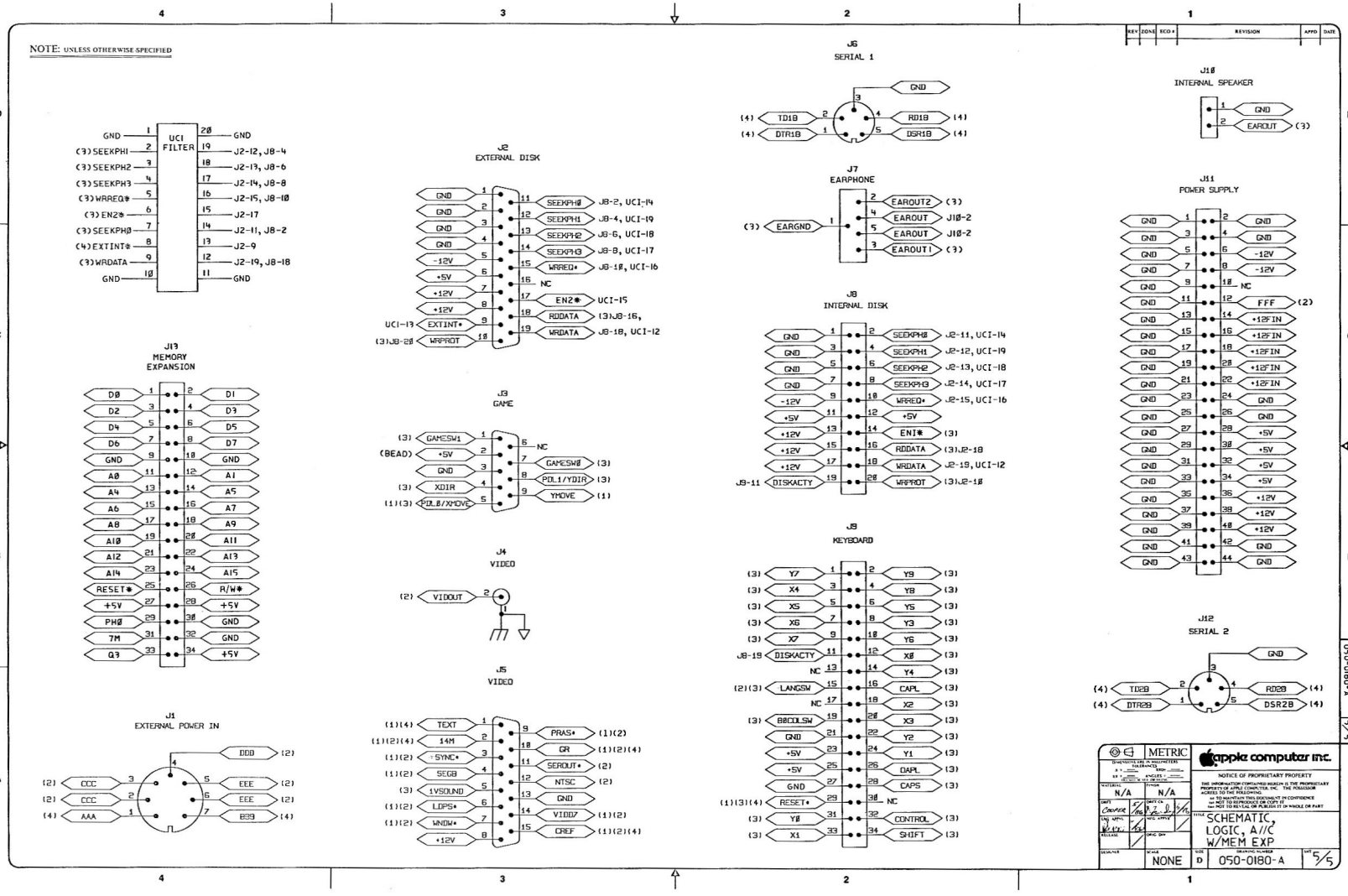


Figure 11-44e
Apple IIc schematic diagram, part 5

Appendix A

The 65C02 Microprocessor

This appendix describes the differences between the 6502 and the 65C02 microprocessors. It also contains the data sheet for the NCR 65C02 microprocessor.

In the data sheet tables, execution times are specified in numbers of cycles. One cycle for the Apple IIc equals 0.978 microseconds.

If you want to write programs that execute on all computers in the Apple II series, make sure your code uses only the subset of 65C02 instructions present on the 6502.

Differences between 6502 and 65C02

The data sheet in this chapter lists the new 65C02 instructions and addressing modes. This section supplements that information by listing the instructions whose execution times or results have changed from their 6502 counterparts.

Differing cycle times

In general, differences in execution times are significant only in time-dependent code, such as precise wait loops. Fortunately, instructions with changed execution times are few.

Table A-1 lists the 65C02 instructions whose number of instruction execution cycles is different from their number on the 6502.

Table A-1
Cycle time differences

Instruction/mode	Opcode	6502 cycles	65C02 cycles
ASL Absolute, X	1E	7	6
DEC Absolute, X	DE	7	6
INC Absolute, X	FE	7	6
JMP (Absolute)	6C	5	6
LSR Absolute, X	5E	7	6
ROL Absolute, X	3E	7	6
ROR Absolute, X	7E	7	6

Differing instruction results

The instructions that have different results from their 6502 equivalents are

- BIT (in immediate mode)
- JMP (indirect, when crossing a page boundary).

The BIT instruction when used in immediate mode (code $89) leaves processor status register bits 7 (N) and 6 (V) unchanged on the 65C02. On the 6502, all modes of the BIT instruction have the same effect on the status register: the value of memory bit 7 is placed in status bit 7, and memory bit 6 is placed in status bit 6. However, all BIT instructions on both versions of the processor set status bit 1 (Z) if the memory location being tested contains a 0.

If the JMP indirect instruction (code $6C) references an indirect address location that spans a page boundary, the 65C02 fetches the high-order byte of the effective address from the first byte of the next page, while the 6502 fetches it from the first byte of the current page. For example, JMP ($02FF) gets ADL from location $02FF on both processors. On the 65C02, ADH comes from $0300 while on the 6502, ADH comes from $0200.

Data sheet

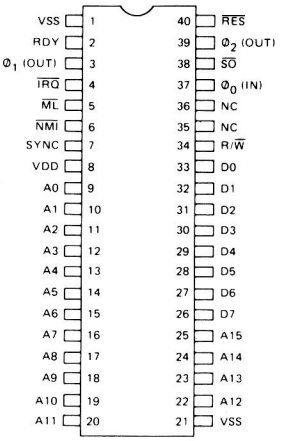
The rest of this appendix is copyright 1982, NCR Corporation, Dayton, Ohio, and is reprinted with their permission.

NCR65C02

■ GENERAL DESCRIPTION

The NCR CMOS 6502 is an 8-bit microprocessor which is software compatible with the NMOS 6502. The NCR65C02 hardware interfaces with all 6500 peripherals. The enhancements include ten additional instructions, expanded operational codes and two new addressing modes. This microprocessor has all of the advantages of CMOS technology: low power consumption, increased noise immunity and higher reliability. The CMOS 6502 is a low power high performance microprocessor with applications in the consumer, business, automotive and communications market.

■ PIN CONFIGURATION

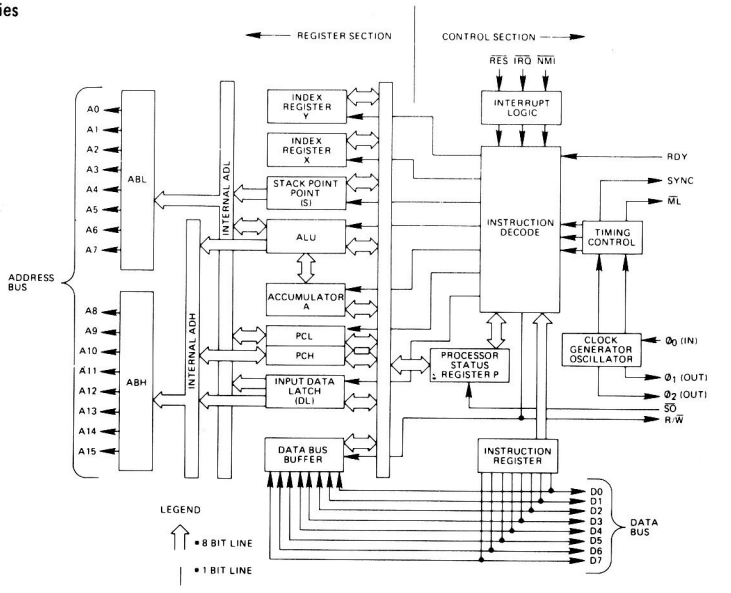


■ FEATURES

- Enhanced software performance including 27 additional OP codes encompassing ten new instructions and two additional addressing modes.
- 66 microprocessor instructions.
- 15 addressing modes.
- 178 operational codes.
- 1MHz, 2MHz operation.
- Operates at frequencies as low as 200 Hz for even lower power consumption (pseudo-static: stop during $\emptyset_2$ high).
- Compatible with NMOS 6500 series microprocessors.
- 64 K-byte addressable memory.
- Interrupt capability.
- Lower power consumption. 4mA @ 1MHz.
- +5 volt power supply.
- 8-bit bidirectional data bus.
- Bus Compatible with M6800.
- Non-maskable interrupt.
- 40 pin dual-in-line packaging.
- 8-bit parallel processing
- Decimal and binary arithmetic.
- Pipeline architecture.
- Programmable stack pointer.
- Variable length stack.
- Optional internal pullups for (RDY, IRQ, SO, NMI and RES)

* Specifications are subject to change without notice.

■ NCR65C02 BLOCK DIAGRAM



Copyright ©1982 by NCR Corporation, Dayton, Ohio, USA

NCR65C02

■ ABSOLUTE MAXIMUM RATINGS: (V_{DD} = 5.0 V ± 5%, V_{SS} = 0 V, T_A = 0° to + 70°C)

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	V_{DD}	−0.3 to +7.0	V
INPUT VOLTAGE	V_{IN}	−0.3 to +7.0	V
OPERATING TEMP.	T_A	0 to + 70	°C
STORAGE TEMP.	T_{STG}	−55 to + 150	°C

■ PIN FUNCTION

PIN	FUNCTION
A0 - A15	Address Bus
D0 - D7	Data Bus
$\overline{IRQ}$ *	Interrupt Request
RDY *	Ready
$\overline{ML}$	Memory Lock
$\overline{NMI}$ *	Non-Maskable Interrupt
SYNC	Synchronize
$\overline{RES}$ *	Reset
$\overline{SO}$ *	Set Overflow
NC	No Connection
R/$\overline{W}$	Read/Write
VDD	Power Supply (+5V)
VSS	Internal Logic Ground
$\emptyset_0$	Clock Input
$\emptyset_1$, $\emptyset_2$	Clock Output

*This pin has an optional internal pullup for a No Connect condition.

■ DC CHARACTERISTICS

	SYMBOL	MIN.	TYP.	MAX	UNIT
Input High Voltage					
$\emptyset_0$ (IN)	V_{IH}	V_{SS} + 2.4	–	V_{DD}	V
Input High Voltage					
$\overline{RES}$, $\overline{NMI}$, RDY, $\overline{IRQ}$, Data, S.O.		V_{SS} + 2.0	–	–	V
Input Low Voltage					
$\emptyset_0$ (IN)	V_{IL}	V_{SS} −0.3	–	V_{SS} + 0.4	V
$\overline{RES}$, $\overline{NMI}$, RDY, $\overline{IRQ}$, Data, S.O.		–	–	V_{SS} + 0.8	V
Input Leakage Current					
(V_{IN} = 0 to 5.25V, V_{DD} = 5.25V)	I_{IN}				
With pullups		-30	–	+30	μA
Without pullups		–	–	+1.0	μA
Three State (Off State) Input Current					
(V_{IN} = 0.4 to 2.4V, V_{CC} = 5.25V)					
Data Lines	I_{TSI}	–	–	10	μA
Output High Voltage					
(I_{OH} = −100 μAdc, V_{DD} = 4.75V					
SYNC, Data, A0-A15, R/W)	V_{OH}	V_{SS} + 2.4	–	–	V
Out Low Voltage					
(I_{OL} = 1.6mAdc, V_{DD} = 4.75V					
SYNC, Data, A0-A15, R/W)	V_{OL}	–	–	V_{SS} + 0.4	V
Supply Current f = 1MHz	I_{DD}	–	–	4	mA
Supply Current f = 2MHz	I_{DD}	–	–	8	mA
Capacitance	C				pF
(V_{IN} = 0, T_A = 25°C, f = 1MHz)					
Logic	C_{IN}	–	–	5	
Data		–	–	10	
A0-A15, R/W, SYNC	Cout	–	–	10	
$\emptyset_0$ (IN)	C$\emptyset_0$ (IN)	–	–	10	

▪ AC CHARACTERISTICS V_{DD} = 5.0V ± 5%, T_A = 0°C to 70°C, Load = 1 TTL + 130 pF

		1MHz		2MHz		3MHz		
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Unit
Delay Time, $\emptyset_0$ (IN) to $\emptyset_2$ (OUT)	t_{DLY}	–	60	–	60	20	60	nS
Delay Time, $\emptyset_1$ (OUT) to $\emptyset_2$ (OUT)	t_{DLY1}	–20	20	–20	20	–20	20	nS
Cycle Time	t_{CYC}	1.0	5000*	0.50	5000*	0.33	5000*	μS
Clock Pulse Width Low	t_{PL}	460	–	220	–	160	–	nS
Clock Pulse Width High	t_{PH}	460	–	220	–	160	–	nS
Fall Time, Rise Time	t_F, t_R	–	25	–	25	–	25	nS
Address Hold Time	t_{AH}	20	–	20	–	0	–	nS
Address Setup Time	t_{ADS}	–	225	–	140	–	110	nS
Access Time	t_{ACC}	650	–	310	–	170	–	nS
Read Data Hold Time	t_{DHR}	10	–	10	–	10	–	nS
Read Data Setup Time	t_{DSU}	100	–	60	–	60	–	nS
Write Data Delay Time	t_{MDS}	–	30	–	30	–	30	nS
Write Data Hold Time	t_{DHW}	20	–	20	–	15	–	nS
$\overline{SO}$ Setup Time	t_{SO}	100	–	100	–	100	–	nS
Processor Control Setup Time**	t_{PCS}	200	–	150	–	150	–	nS
SYNC Setup Time	t_{SYNC}	–	225	–	140	–	100	nS
$\overline{ML}$ Setup Time	t_{ML}	–	225	–	140	–	100	nS
Input Clock Rise/Fall Time	$t_{F\emptyset o}$, $t_{R\emptyset o}$	–	25	–	25	–	25	nS

*NCR65C02 can be held static with $\emptyset_2$ high.

**This parameter must only be met to guarantee that the signal will be recognized at the current clock cycle.

▪ MICROPROCESSOR OPERATIONAL ENHANCEMENTS

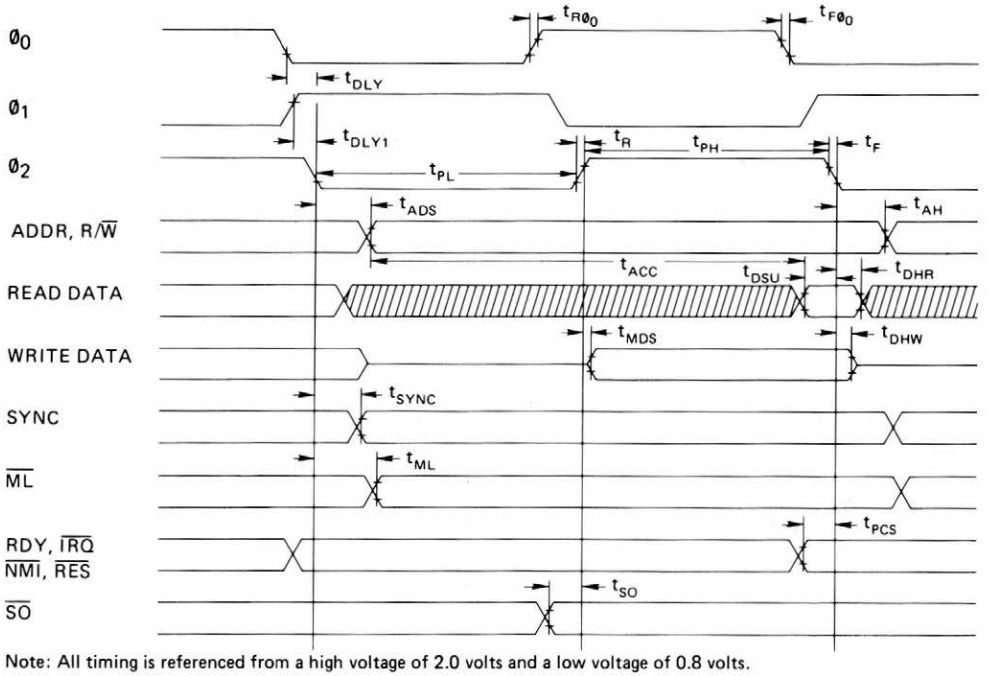
Function	NMOS 6502 Microprocessor	NCR65C02 Microprocessor
Indexed addressing across page boundary.	Extra read of invalid address.	Extra read of last instruction byte.
Execution of invalid op codes.	Some terminate only by reset. Results are undefined.	All are NOPs (reserved for future use). Op Code / Bytes / Cycles X2 / 2 / 2 X3, X7, XB, XF / 1 / 1 44 / 2 / 3 54, D4, F4 / 2 / 4 5C / 3 / 8 DC, FC / 3 / 4
Jump indirect, operand = XXFF.	Page address does not increment.	Page address increments and adds one additional cycle.
Read/modify/write instructions at effective address.	One read and two write cycles.	Two read and one write cycle.
Decimal flag.	Indeterminate after reset.	Initialized to binary mode (D=0) after reset and interrupts.
Flags after decimal operation.	Invalid N, V and Z flags.	Valid flag adds one additional cycle.
Interrupt after fetch of BRK instruction.	Interrupt vector is loaded, BRK vector is ignored.	BRK is executed, then interrupt is executed.

▪ MICROPROCESSOR HARDWARE ENHANCEMENTS

Function	NMOS 6502	NCR65C02
Assertion of Ready RDY during write operations.	Ignored.	Stops processor during $\emptyset_2$.
Unused input-only pins ($\overline{IRQ}$, $\overline{NMI}$, RDY, $\overline{RES}$, $\overline{SO}$).	Must be connected to low impedance signal to avoid noise problems.	Connected internally by a high-resistance to V_{DD} (approximately 250 K ohm.)

NCR65C02

▪ TIMING DIAGRAM



Note: All timing is referenced from a high voltage of 2.0 volts and a low voltage of 0.8 volts.

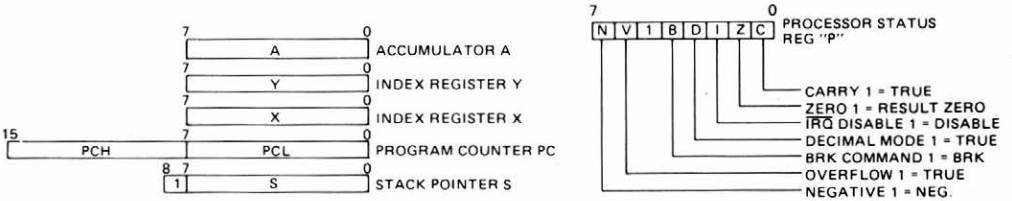
▪ NEW INSTRUCTION MNEMONICS

HEX	MNEMONIC	DESCRIPTION
80	BRA	Branch relative always [Relative]
3A	DEA	Decrement accumulator [Accum]
1A	INA	Increment accumulator [Accum]
DA	PHX	Push X on stack [Implied]
5A	PHY	Push Y on stack [Implied]
FA	PLX	Pull X from stack [Implied]
7A	PLY	Pull Y from stack [Implied]
9C	STZ	Store zero [Absolute]
9E	STZ	Store zero [ABS, X]
64	STZ	Store zero [Zero page]
74	STZ	Store zero [ZPG,X]
1C	TRB	Test and reset memory bits with accumulator [Absolute]
14	TRB	Test and reset memory bits with accumulator [Zero page]
0C	TSB	Test and set memory bits with accumulator [Absolute]
04	TSB	Test and set memory bits with accumulator [Zero page]

▪ ADDITIONAL INSTRUCTION ADDRESSING MODES

HEX	MNEMONIC	DESCRIPTION
72	ADC	Add memory to accumulator with carry [(ZPG)]
32	AND	"AND" memory with accumulator [(ZPG)]
3C	BIT	Test memory bits with accumulator [ABS, X]
34	BIT	Test memory bits with accumulator [ZPG, X]
D2	CMP	Compare memory and accumulator [(ZPG)]
52	EOR	"Exclusive Or" memory with accumulator [(ZPG)]
7C	JMP	Jump (New addressing mode) [ABS(IND,X)]
B2	LDA	Load accumulator with memory [(ZPG)]
12	ORA	"OR" memory with accumulator [(ZPG)]
F2	SBC	Subtract memory from accumulator with borrow [(ZPG)]
92	STA	Store accumulator in memory [(ZPG)]

■ MICROPROCESSOR PROGRAMMING MODEL



■ FUNCTIONAL DESCRIPTION

Timing Control
The timing control unit keeps track of the instruction cycle being monitored. The unit is set to zero each time an instruction fetch is executed and is advanced at the beginning of each phase one clock pulse for as many cycles as is required to complete the instruction. Each data transfer which takes place between the registers depends upon decoding the contents of both the instruction register and the timing control unit.

Program Counter
The 16-bit program counter provides the addresses which step the microprocessor through sequential instructions in a program.

Each time the microprocessor fetches an instruction from program memory, the lower byte of the program counter (PCL) is placed on the low-order bits of the address bus and the higher byte of the program counter (PCH) is placed on the high-order 8 bits. The counter is incremented each time an instruction or data is fetched from program memory.

Instruction Register and Decode
Instructions fetched from memory are gated onto the internal data bus. These instructions are latched into the instruction register, then decoded, along with timing and interrupt signals, to generate control signals for the various registers.

Arithmetic and Logic Unit (ALU)
All arithmetic and logic operations take place in the ALU including incrementing and decrementing internal registers (except the program counter). The ALU has no internal memory and is used only to perform logical and transient numerical operations.

Accumulator
The accumulator is a general purpose 8-bit register that stores the results of most arithmetic and logic operations, and in addition, the accumulator usually contains one of the two data words used in these operations.

Index Registers
There are two 8-bit index registers (X and Y), which may be used to count program steps or to provide an index value to be used in generating an effective address.

When executing an instruction which specifies indexed addressing, the CPU fetches the op code and the base address, and modifies the address by adding the index register to it prior to performing the desired operation. Pre- or post-indexing of indirect addresses is possible (see addressing modes).

Stack Pointer
The stack pointer is an 8-bit register used to control the addressing of the variable-length stack on page one. The stack pointer is automatically incremented and decremented under control of the microprocessor to perform stack manipulations under direction of either the program or interrupts ($\overline{NMI}$ and $\overline{IRQ}$). The stack allows simple implementation of nested subroutines and multiple level interrupts. The stack pointer should be initialized before any interrupts or stack operations occur.

Processor Status Register
The 8-bit processor status register contains seven status flags. Some of the flags are controlled by the program, others may be controlled both by the program and the CPU. The 6500 instruction set contains a number of conditional branch instructions which are designed to allow testing of these flags (see microprocessor programming model).

NCR65C02
■ ADDRESSING MODES

Fifteen addressing modes are available to the user of the NCR65C02 microprocessor. The addressing modes are described in the following paragraphs:

Implied Addressing [Implied]
In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

Accumulator Addressing [Accum]
This form of addressing is represented with a one byte instruction and implies an operation on the accumulator.

Immediate Addressing [Immediate]
With immediate addressing, the operand is contained in the second byte of the instruction; no further memory addressing is required.

Absolute Addressing [Absolute]
For absolute addressing, the second byte of the instruction specifies the eight low-order bits of the effective address, while the third byte specifies the eight high-order bits. Therefore, this addressing mode allows access to the total 64K bytes of addressable memory.

Zero Page Addressing [Zero Page]
Zero page addressing allows shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. The careful use of zero page addressing can result in significant increase in code efficiency.

Absolute Indexed Addressing [ABS, X or ABS, Y]
Absolute indexed addressing is used in conjunction with X or Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields, resulting in reduced coding and execution time.

Zero Page Indexed Addressing [ZPG, X or ZPG, Y]
Zero page absolute addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally, due to the "Zero Page" addressing nature of this mode, no carry is added to the high-order eight bits of memory, and crossing of page boundaries does not occur.

Relative Addressing [Relative]
Relative addressing is used only with branch instructions; it establishes a destination for the conditional branch. The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the program counter when the counter is set at the next instruction. The range of the offset is −128 to +127 bytes from the next instruction.

Zero Page Indexed Indirect Addressing [(IND, X)]
With zero page indexed indirect addressing (usually referred to as indirect X) the second byte of the instruction is added to the contents of the X index register; the carry is discarded. The result of this addition points to a memory location on page zero whose contents is the low-order eight bits of the effective address. The next memory location in page zero contains the high-order eight bits of the effective address. Both memory locations specifying the high- and low-order bytes of the effective address must be in page zero.

***Absolute Indexed Indirect Addressing [ABS(IND, X)] (Jump Instruction Only)**
With absolute indexed indirect addressing the contents of the second and third instruction bytes are added to the X register. The result of this addition, points to a memory location containing the lower-order eight bits of the effective address. The next memory location contains the higher-order eight bits of the effective address.

Indirect Indexed Addressing [(IND), Y]
This form of addressing is usually referred to as Indirect, Y. The second byte of the instruction points to a memory location in page zero. The contents of this memory location are added to the contents of the Y index register, the result being the low-order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high-order eight bits of the effective address.

***Zero Page Indirect Addressing [(ZPG)]**
In the zero page indirect addressing mode, the second byte of the instruction points to a memory location on page zero containing the low-order byte of the effective address. The next location on page zero contains the high-order byte of the effective address.

Absolute Indirect Addressing [(ABS)] (Jump Instruction Only)
The second byte of the instruction contains the low-order eight bits of a memory location. The high-order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low-order byte of the effective address. The next memory location contains the high-order byte of the effective address which is loaded into the 16 bit program counter.

NOTE: * = New Address Modes

▪ SIGNAL DESCRIPTION

Address Bus (A0-A15)
A0-A15 forms a 16-bit address bus for memory and I/O exchanges on the data bus. The output of each address line is TTL compatible, capable of driving one standard TTL load and 130pF.

Clocks (ϕ_0, ϕ_1, and ϕ_2)
ϕ_0 is a TTL level input that is used to generate the internal clocks in the 6502. Two full level output clocks are generated by the 6502. The ϕ_2 clock output is in phase with ϕ_0. The ϕ_1 output pin is 180° out of phase with ϕ_0. (See timing diagram.)

Data Bus (D0-D7)
The data lines (D0-D7) constitute an 8-bit bidirectional data bus used for data exchanges to and from the device and peripherals. The outputs are three-state buffers capable of driving one TTL load and 130 pF.

Interrupt Request ($\overline{IRQ}$)
This TTL compatible input requests that an interrupt sequence begin within the microprocessor. The $\overline{IRQ}$ is sampled during ϕ_2 operation; if the interrupt flag in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during ϕ_1. The program counter and processor status register are stored in the stack. The microprocessor will then set the interrupt mask flag high so that no further $\overline{IRQ}$s may occur. At the end of this cycle, the program counter low will be loaded from address FFFE, and program counter high from location FFFF, transferring program control to the memory vector located at these addresses. The RDY signal must be in the high state for any interrupt to be recognized. A 3K ohm external resistor should be used for proper wire OR operation.

Memory Lock ($\overline{ML}$)
In a multiprocessor system, the $\overline{ML}$ output indicates the need to defer the rearbitration of the next bus cycle to ensure the integrity of read-modify-write instructions. $\overline{ML}$ goes low during ASL, DEC, INC, LSR, ROL, ROR, TRB, TSB memory referencing instructions. This signal is low for the modify and write cycles.

Non-Maskable Interrupt ($\overline{NMI}$)
A negative-going edge on this input requests that a non-maskable interrupt sequence be generated within the microprocessor. The $\overline{NMI}$ is sampled during ϕ_2; the current instruction is completed and the interrupt sequence begins during ϕ_1. The program counter is loaded with the interrupt vector from locations FFFA (low byte) and FFFB (high byte), thereby transferring program control to the non-maskable interrupt routine.

Note: Since this interrupt is non-maskable, another $\overline{NMI}$ can occur before the first is finished. Care should be taken when using $\overline{NMI}$ to avoid this.

Ready (RDY)
This input allows the user to single-cycle the microprocessor on all cycles including write cycles. A negative transition to the low state, during or coincident with phase one (ϕ_1), will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two (ϕ_2) in which the ready signal is low. This feature allows microprocessor interfacing with low-speed memory as well as direct memory access (DMA).

Reset ($\overline{RES}$)
This input is used to reset the microprocessor. Reset must be held low for at least two clock cycles after V_{DD} reaches operating voltage from a power down. A positive transistion on this pin will then cause an initialization sequence to begin. Likewise, after the system has been operating, a low on this line of at least two cycles will cease microprocessing activity, followed by initialization after the positive edge on $\overline{RES}$.

When a positive edge is detected, there is an initialization sequence lasting six clock cycles. Then the interrupt mask flag is set, the decimal mode is cleared, and the program counter is loaded with the restart vector from locations FFFC (low byte) and FFFD (high byte). This is the start location for program control. This input should be high in normal operation.

Read/Write (R/$\overline{W}$)
This signal is normally in the high state indicating that the microprocessor is reading data from memory or I/O bus. In the low state the data bus has valid data from the microprocessor to be stored at the addressed memory location.

Set Overflow ($\overline{SO}$)
A negative transition on this line sets the overflow bit in the status code register. The signal is sampled on the trailing edge of ϕ_1.

Synchronize (SYNC)
This output line is provided to identify those cycles during which the microprocessor is doing an OP CODE fetch. The SYNC line goes high during ϕ_1 of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the ϕ_1 clock pulse in which SYNC went high, the processor will stop in its current state and will remain in the state until the RDY line goes high. In this manner, the SYNC signal can be used to control RDY to cause single instruction execution.

NCR65C02
▪ INSTRUCTION SET — ALPHABETICAL SEQUENCE

ADC Add Memory to Accumulator with Carry
AND "AND" Memory with Accumulator
ASL Shift One Bit Left
BCC Branch on Carry Clear
BCS Branch on Carry Set
BEQ Branch on Result Zero
BIT Test Memory Bits with Accumulator
BMI Branch on Result Minus
BNE Branch on Result not Zero
BPL Branch on Result Plus
*BRA Branch Always
BRK Force Break
BVC Branch on Overflow Clear
BVS Branch on Overflow Set
CLC Clear Carry Flag
CLD Clear Decimal Mode
CLI Clear Interrupt Disable Bit
CLV Clear Overflow Flag
CMP Compare Memory and Accumulator
CPX Compare Memory and Index X
CPY Compare Memory and Index Y
* DEA Decrement Accumulator
DEC Decrement by One
DEX Decrement Index X by One
DEY Decrement Index Y by One
EOR "Exclusive-or" Memory with Accumulator
* INA Increment Accumulator
INC Increment by One
INX Increment Index X by One
INY Increment Index Y by One
JMP Jump to New Location
JSR Jump to New Location Saving Return Address
LDA Load Accumulator with Memory
LDX Load Index X with Memory
LDY Load Index Y with Memory
LSR Shift One Bit Right
NOP No Operation
ORA "OR" Memory with Accumulator
PHA Push Accumulator on Stack
PHP Push Processor Status on Stack
* PHX Push Index X on Stack
* PHY Push Index Y on Stack
PLA Pull Accumulator from Stack
PLP Pull Processor Status from Stack
* PLX Pull Index X from Stack
* PLY Pull Index Y from Stack
ROL Rotate One Bit Left
ROR Rotate One Bit Right
RTI Return from Interrupt
RTS Return from Subroutine
SBC Subtract Memory from Accumulator with Borrow
SEC Set Carry Flag
SED Set Decimal Mode
SEI Set Interrupt Disable Bit
STA Store Accumulator in Memory
STX Store Index X in Memory
STY Store Index Y in Memory
* STZ Store Zero in Memory
TAX Transfer Accumulator to Index X
TAY Transfer Accumulator to Index Y
* TRB Test and Reset Memory Bits with Accumulator
* TSB Test and Set Memory Bits with Accumulator
TSX Transfer Stack Pointer to Index X
TXA Transfer Index X to Accumulator
TXS Transfer Index X to Stack Pointer
TYA Transfer Index Y to Accumulator

Note: * = New Instruction

▪ MICROPROCESSOR OP CODE TABLE

S D	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRK	ORA ind, X			TSB* zpg	ORA zpg	ASL zpg		PHP	ORA imm	ASL A		TSB* abs	ORA abs	ASL abs		0
1	BPL rel	ORA ind, Y	ORA*† (zpg)		TRB* zpg	ORA zpg, X	ASL zpg, X		CLC	ORA abs, Y	INA* A		TRB* abs	ORA abs, X	ASL abs, X		1
2	JSR abs	AND ind, X			BIT zpg	AND zpg	ROL zpg		PLP	AND imm	ROL A		BIT abs	AND abs	ROL abs		2
3	BMI rel	AND ind, Y	AND*† (zpg)		BIT* zpg, X	AND zpg, X	ROL zpg, X		SEC	AND abs, Y	DEA* A		BIT*† abs, X	AND abs, X	ROL abs, X		3
4	RTI	EOR ind, X				EOR zpg	LSR zpg		PHA	EOR imm	LSR A		JMP abs	EOR abs	LSR abs		4
5	BVC rel	EOR ind, Y	EOR*† (zpg)			EOR zpg, X	LSR zpg, X		CLI	EOR abs, Y	PHY*			EOR abs, X	LSR abs, X		5
6	RTS	ADC ind, X			STZ* zpg	ADC zpg	ROR zpg		PLA	ADC imm	ROR A		JMP (abs)	ADC abs	ROR abs		6
7	BVS rel	ADC ind, Y	ADC*† (zpg)		STZ* zpg, X	ADC zpg, X	ROR zpg, X		SEI	ADC abs, Y	PLY*		JMP*† abs (ind,X)	ADC abs, X	ROR abs, X		7
8	BRA* rel	STA ind, X			STY zpg	STA zpg	STX zpg		DEY	BIT* imm	TXA		STY abs	STA abs	STX abs		8
9	BCC rel	STA ind, Y	STA*† (zpg)		STY zpg, X	STA zpg, X	STX zpg, Y		TYA	STA abs, Y	TXS		STZ* abs	STA abs, X	STZ* abs, X		9
A	LDY imm	LDA ind, X	LDX imm		LDY zpg	LDA zpg	LDX zpg		TAY	LDA imm	TAX		LDY abs	LDA abs	LDX abs		A
B	BCS rel	LDA ind, Y	LDA*† (zpg)		LDY zpg, X	LDA zpg, X	LDX zpg, Y		CLV	LDA abs, Y	TSX		LDY abs, X	LDA abs, X	LDX abs, Y		B
C	CPY imm	CMP ind, X			CPY zpg	CMP zpg	DEC zpg		INY	CMP imm	DEX		CPY abs	CMP abs	DEC abs		C
D	BNE rel	CMP ind, Y	CMP*† (zpg)			CMP zpg, X	DEC zpg, X		CLD	CMP abs, Y	PHX*			CMP abs, X	DEC abs, X		D
E	CPX imm	SBC ind, X			CPX zpg	SBC zpg	INC zpg		INX	SBC imm	NOP		CPX abs	SBC abs	INC abs		E
F	BEQ rel	SBC ind, Y	SBC*† (zpg)			SBC zpg, X	INC zpg, X		SED	SBC abs, Y	PLX*			SBC abs, X	INC abs, X		F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Note: * = New OP Codes
Note: † = New Address Modes

■ OPERATIONAL CODES, EXECUTION TIME, AND MEMORY REQUIREMENTS

MNE	OPERATION		IMME-DIATE OP	n	#	ABSO-LUTE OP	n	#	ZERO PAGE OP	n	#	ACCUM OP	n	#	IM-PLIED OP	n	#	(IND, X) OP	n	#	(IND), Y OP	n	#	ZPG, X OP	n	#	ZPG, Y OP	n	#	ABS, X OP	n	#	ABS, Y OP	n	#	RELA-TIVE OP	n	#	(ABS) OP	n	#	ABS (IND, X) OP	n	#	(ZPG) OP	n	#	PROCESSOR STATUS CODES 7 6 5 4 3 2 1 0 N V . B D I Z C	MNE
ADC	A + M + C → A	(1,3)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2				7D	4	3	79	4	3									72	5	2	N V . . . . Z C	ADC
AND	A ∧ M → A	(1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2				3D	4	3	39	4	3									32	5	2	N . . . . . Z .	AND
ASL	C ← 7 … 0 ← 0	(1)				0E	6	3	06	5	2	0A	2	1										16	6	2				1E	6	3													N . . . . . Z C	ASL
BCC	Branch if C=0	(2)																																	90	2	2											. . . . .	BCC
BCS	Branch if C=1	(2)																																	B0	2	2											. . . . .	BCS
BEQ	Branch if Z=1	(2)																																	F0	2	2											. . . . .	BEQ
BIT	A ∧ M	(4,5)	89	2	2	2C	4	3	24	3	2													34	4	2				3C	4	3															M_7 M_6 . . . . Z .	BIT
BMI	Branch if N=1	(2)																																	30	2	2											. . . . .	BMI
BNE	Branch if Z=0	(2)																																	D0	2	2											. . . . .	BNE
BPL	Branch if N=0	(2)																																	10	2	2											. . . . .	BPL
BRA	Branch Always	(2)																																	80	2	2											. . . . .	BRA
BRK	Break														00	7	1																														. . 1 . . .	BRK
BVC	Branch if V=0	(2)																																	50	2	2											. . . . .	BVC
BVS	Branch if V=1	(2)																																	70	2	2											 0	BVS
CLC	0 → C														18	2	1																														 0	CLC
CLD	0 → D														D8	2	1																														. . . . 0 .	CLD
CLI	0 → I														58	2	1																														. . . . 0 .	CLI
CLV	0 → V														B8	2	1																														. 0	CLV
CMP	A - M	(1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2				DD	4	3	D9	4	3									D2	5	2	N . . . . . Z C	CMP
CPX	X - M		E0	2	2	EC	4	3	E4	3	2																																				N . . . . . Z C	CPX
CPY	Y - M		C0	2	2	CC	4	3	C4	3	2																																				N . . . . . Z C	CPY
DEA	A - 1 → A											3A	2	1																															N . . . . . Z .	DEA
DEC	M - 1 → M	(1)				CE	6	3	C6	5	2													D6	6	2				DE	6	3															N . . . . . Z .	DEC
DEX	X - 1 → X														CA	2	1																															N . . . . . Z .	DEX
DEY	Y - 1 → Y														88	2	1																															N . . . . . Z .	DEY
EOR	A ⊻ M → A		49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2				5D	4	3	59	4	3									52	5	2	N . . . . . Z .	EOR
INA	A + 1 → A											1A	2	1																															N . . . . . Z .	INA
INC	M + 1 → M	(1)				EE	6	3	E6	5	2													F6	6	2				FE	6	3															N . . . . . Z .	INC
INX	X + 1 → X														E8	2	1																															N . . . . . Z .	INX
INY	Y + 1 → Y														C8	2	1																															N . . . . . Z .	INY
JMP	Jump to new loc					4C	3	3																																6C	6	3	7C	6	3						JMP
JSR	Jump Subroutine					20	6	3																																										JSR
LDA	M → A	(1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2				BD	4	3	B9	4	3									B2	5	2	N . . . . . Z .	LDA
LDX	M → X	(1)	A2	2	2	AE	4	3	A6	3	2																B6	4	2				BE	4	3											N . . . . . Z .	LDX
LDY	M → Y	(1)	A0	2	2	AC	4	3	A4	3	2													B4	4	2				BC	4	3															N . . . . . Z .	LDY
LSR	0 → 7 … 0 → C	(1)				4E	6	3	46	5	2	4A	2	1										56	6	2				5E	6	3													0 . . . . . Z C	LSR
NOP	PC + 1 → PC														EA	2	1																															. . . . .	NOP
ORA	A ∨ M → A	(1)	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2				1D	4	3	19	4	3									12	5	2	N . . . . . Z .	ORA
PHA	A → M_S S - 1 → S														48	3	1																															. . . . .	PHA
PHP	P → M_S S - 1 → S														08	3	1																															. . . . .	PHP
PHX	X → M_S S - 1 → S														DA	3	1																															. . . . .	PHX
PHY	Y → M_S S - 1 → S														5A	3	1																															. . . . .	PHY
PLA	S + 1 → S M_S → A														68	4	1																															N . . . . Z .	PLA
PLP	S + 1 → S M_S → P														28	4	1																															N V . 1 D I Z C	PLP
PLX	S + 1 → S M_S → X														FA	4	1																															N . . . . Z .	PLX
PLY	S + 1 → S M_S → Y														7A	4	1																															N . . . . Z .	PLY
ROL	7 … 0 ← C ←	(1)				2E	6	3	26	5	2	2A	2	1										36	6	2				3E	6	3															N . . . . Z C	ROL
ROR	→ C → 7 … 0	(1)				6E	6	3	66	5	2	6A	2	1										76	6	2				7E	6	3															N . . . . Z C	ROR
RTI	Return from Inter.														40	6	1																															N V . 1 D I Z C	RTI
RTS	Return from Subr.														60	6	1																															. . . . .	RTS
SBC	A - M - $\overline{C}$ → A	(1,3)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2				FD	4	3	F9	4	3									F2	5	2	N V . . . . Z C	SBC
SEC	1 → C														38	2	1																															 1	SEC
SED	1 → D														F8	2	1																															. . . . 1 .	SED
SEI	1 → I														78	2	1																															. . . . 1 .	SEI
STA	A → M					8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2				9D	5	3	99	5	3									92	5	2	. . . . .	STA
STX	X → M					8E	4	3	86	3	2																96	4	2																			. . . . .	STX
STY	Y → M					8C	4	3	84	3	2													94	4	2																							. . . . .	STY
STZ	00 → M					9C	4	3	64	3	2													74	4	2				9E	5	3																	. . . . .	STZ
TAX	A → X														AA	2	1																															N . . . . Z .	TAX
TAY	A → Y														A8	2	1																															N . . . . Z .	TAY
TRB	$\overline{A}$ ∧ M → M	(4)				1C	6	3	14	5	2																																					. . . . . Z .	TRB
TSB	A ∨ M → M	(4)				0C	6	3	04	5	2																																					. . . . . Z .	TSB
TSX	S → X														BA	2	1																															N . . . . Z .	TSX
TXA	X → A														8A	2	1																															N . . . . Z .	TXA
TXS	X → S														9A	2	1																															. . . . .	TXS
TYA	Y → A														98	2	1																															N . . . . Z .	TYA

Notes:

1. Add 1 to "n" if page boundary is crossed.
2. Add 1 to "n" if branch occurs to same page.
   Add 2 to "n" if branch occurs to different page.
3. Add 1 to "n" if decimal mode.
4. V bit equals memory bit 6 prior to execution.
   N bit equals memory bit 7 prior to execution.

*5. The immediate addressing mode of the BIT instruction leaves bits 6 & 7 (V & N) in the Processor Status Code Register unchanged.

X Index X
Y Index Y
A Accumulator
M Memory per effective address
Ms Memory per stack pointer

\+ Add
− Subtract
∧ And
∨ Or
⊻ Exclusive or

n No. Cycles
No. Bytes
M_6 Memory bit 6
M_7 Memory bit 7

Appendix B

Memory Map

This appendix lists all important RAM and hardware locations in address order and briefly describes them. Appendix C contains a similar list for important firmware addresses.

The tables in this appendix list addresses in either two or three forms: the hexadecimal form (preceded by a dollar sign) for use in assembly language; the decimal form for use in Applesoft BASIC; and (for numbers greater than 32,767) the complementary decimal value for use in Apple Integer BASIC.

Page $00

Table B-1 lists the zero page addresses in hexadecimal and decimal form, followed by symbols denoting the firmware or system software that uses them.

- *M* denotes the monitor.
- *A* denotes Applesoft BASIC.
- *I* denotes Integer BASIC.
- *D* denotes DOS 3.3.
- *P* denotes ProDOS. Locations whose contents ProDOS saves and restores afterward have a P in parentheses, indicating that ProDOS has no net effect on them.

Table B-1
Page $00 use

Hex	Dec	Used by	Hex	Dec	Used by
$00	0	A	$30	48	M
$01	1	A	$31	49	M
$02	2	A	$32	50	M
$03	3	A	$33	51	M
$04	4	A	$34	52	M
$05	5	A	$35	53	M D
$06	6		$36	54	M D
$07	7		$37	55	M D
$08	8		$38	56	M D
$09	9		$39	57	M D
$0A	10	A	$3A	58	M P
$0B	11	A	$3B	59	M P
$0C	12	A	$3C	60	M P
$0D	13	A	$3D	61	M P
$0E	14	A	$3E	62	M D P
$0F	15	A	$3F	63	M D P
$10	16	A	$40	64	M D (P)
$11	17	A	$41	65	M D (P)
$12	18	A	$42	66	M D (P)
$13	19	A	$43	67	M D (P)
$14	20	A	$44	68	M D (P)
$15	21	A	$45	69	M D (P)
$16	22	A	$46	70	M D (P)
$17	23	A	$47	71	M D (P)
$18	24	A	$48	72	M D (P)
$19	25		$49	73	M (P)
$1A	26		$4A	74	I D (P)
$1B	27		$4B	75	I D (P)
$1C	28		$4C	76	I D (P)
$1D	29		$4D	77	I D (P)
$1F	31		$4F	79	M
$25	37	M	$55	85	M A I
$26	38	M D	$56	86	A I
$27	39	M D	$57	87	A I
$28	40	M D	$58	88	A I
$29	41	M D	$59	89	A I
$2A	42	M D	$5A	90	A I
$2B	43	M D	$5B	91	A I
$2C	44	M D	$5C	92	A I
$2D	45	M D	$5D	93	A I
$2E	46	M D	$5E	94	A I
$2F	47	M D	$5F	95	A I

Table B-1 (continued)
Page $00 use

Hex	Dec	Used by	Hex	Dec	Used by
$60	96	A I	$90	144	A I
$61	97	A I	$91	145	A I
$62	98	A I	$92	146	A I
$63	99	A I	$93	147	A I
$64	100	A I	$94	148	A I
$65	101	A I	$95	149	A I
$66	102	A I	$96	150	A I
$67	103	A I D	$97	151	A I
$68	104	A I D	$98	152	A I
$69	105	A I D	$99	153	A I
$6A	106	A I D	$9A	154	A I
$6B	107	A I	$9B	155	A I
$6C	108	A I	$9C	156	A I
$6D	109	A I	$9D	157	A I
$6E	110	A I	$9E	158	A I
$6F	111	A I D	$9F	159	A I
$70	112	A I D	$A0	160	A I
$71	113	A I	$A1	161	A I
$72	114	A I	$A2	162	A I
$73	115	A I	$A3	163	A I
$74	116	A I	$A4	164	A I
$75	117	A I	$A5	165	A I
$76	118	A I	$A6	166	A I
$77	119	A I	$A7	167	A I
$78	120	A I	$A8	168	A I
$79	121	A I	$A9	169	A I
$7A	122	A I	$AA	170	A I
$7B	123	A I	$AB	171	A I
$7C	124	A I	$AC	172	A I
$7D	125	A I	$AD	173	A I
$7E	126	A I	$AE	174	A I
$7F	127	A I	$AF	175	A I D
$80	128	A I	$B0	176	A I D
$81	129	A I	$B1	177	A I
$82	130	A I	$B2	178	A I
$83	131	A I	$B3	179	A I
$84	132	A I	$B4	180	A I
$85	133	A I	$B5	181	A I
$86	134	A I	$B6	182	A I
$87	135	A I	$B7	183	A I
$88	136	A I	$B8	184	A I
$89	137	A I	$B9	185	A I

Table B-1 (continued)
Page $00 use

Hex	Dec	Used by	Hex	Dec	Used by
$8A	138	A I	$BA	186	A I
$8B	139	A I	$BB	187	A I
$8C	140	A I	$BC	188	A I
$8D	141	A I	$BD	189	A I
$8E	142	A I	$BE	190	A I
$8F	143	A I	$BF	191	A I
$C0	192	A I	$E0	224	A
$C1	193	A I	$E1	225	A
$C2	194	A I	$E2	226	A
$C3	195	A I	$E3	227	
$C4	196	A I	$E4	228	A
$C5	197	A I	$E5	229	A
$C6	198	A I	$E6	230	A
$C7	199	A I	$E7	231	A
$C8	200	A I	$E8	232	A
$C9	201	A I	$E9	233	A
$CA	202	A I D	$EA	234	A
$CB	203	A I D	$EB	235	
$CC	204	A I D	$EC	236	
$CD	205	A I D	$ED	237	
$CE	206	I	$EE	238	
$CF	207	I	$EF	239	
$D0	208	A I	$F0	240	A
$D1	209	A I	$F1	241	A
$D2	210	A I	$F2	242	A
$D3	211	A I	$F3	243	A
$D4	212	A I	$F4	244	A
$D5	213	A I	$F5	245	A
$D6	214	I	$F6	246	A
$D7	215	I	$F7	247	A
$D8	216	A I D	$F8	248	A
$D9	217	A I	$F9	249	
$DA	218	A I	$FA	250	
$DB	219	A I	$FB	251	
$DC	220	A I	$FC	252	
$DD	221	A I	$FD	253	
$DE	222	A I	$FE	254	
$DF	223	A I	$FF	255	

Page $03

Most of page $03 is available for small machine-language programs. The built-in Monitor uses the top 16 addresses of page $03, as shown in Figure B-2; the XFer routine uses locations $03ED and $03EE. If you are using DOS or ProDOS, it also uses the 32 locations $03D0 through $03EF.

Table B-2
Page $03 use

Hex	Dec	Use
$03F0	1008	Address of BRK request handler (normally $59,
$03F1	1009	$FA)
$03F2	1010	Reset vector
$03F3	1011	
$03F4	1012	Power-up byte (see text)
$03F5	1013	Jump instruction to Applesoft &-command
$03F6	1014	handler (initially $4C, $58, $FF)
$03F7	1015	
$03F8	1016	Jump instruction to user Control-Y command
$03F9	1017	handler
$03FA	1018	
$03FB	1019	Jump instruction to NMI interrupt handler
$03FC	1020	(not used by Apple IIc)
$03FD	1021	
$03FE	1022	Address of user IRQ interrupt handler
$03FF	1023	

Screen holes

One result of the way the Apple IIc hardware maps display memory on the screen is that groups of 8 memory addresses are left over in 16 areas of the text and low-resolution display pages—8 areas in main RAM and 8 in auxiliary RAM. The firmware uses for these 128 bytes are shown in Tables B-3 and B-4.

Memory expansion

The version of the Apple IIc that supports the memory expansion card uses some of the screen holes differently than earlier versions. Where they differ, the memory expansion ROM assignments are given in parentheses in Tables B-3 and B-4 following the original and UniDisk 3.5 assignments.

Table B-3
Main memory screen hole allocations

Hex	Dec	Description
$0478	1144	Mouse port: low byte of clamping minimum
$0479	1145	Reserved for serial port 1
$047A	1146	Reserved for serial port 2
$047B	1147	Reserved
$047C	1148	Low byte of X coordinate (Reserved)
$047D	1149	Reserved for mouse port
$047E	1150	Reserved
$047F	1151	Reserved (Low byte of X coordinate)
$04F8	1272	Mouse port: low byte of clamping maximum
$04F9	1273	Reserved for serial port 1
$04FA	1274	Reserved for serial port 2
$04FB	1275	Reserved
$04FC	1276	Low byte of Y coordinate (Reserved)
$04FD	1277	Reserved for mouse port
$04FE	1278	Reserved
$04FF	1279	Reserved (Low byte of Y coordinate)
$0578	1400	Mouse port: high byte of clamping minimum
$0579	1401	Port 1 printer width (1–255; 0 = unlimited)
$057A	1402	Port 2 line length (1–255; 0 = unlimited)
$057B	1403	Cursor horizontal position (80-column display)
$057C	1404	High byte of X coordinate (Reserved)
$057D	1405	Reserved for mouse port
$057E	1406	Reserved
$057F	1407	Reserved (High byte of X coordinate)
$05F8	1528	Mouse port: high byte of clamping maximum
$05F9	1529	Port 1 temporary storage location
$05FA	1530	Port 2 temporary storage location
$05FB	1531	Reserved
$05FC	1532	High byte of Y coordinate (Reserved)
$05FD	1533	Reserved for mouse port
$05FE	1534	Reserved
$05FF	1535	Reserved (High byte of Y coordinate)

Table B-3 (continued)
Main memory screen hole allocations

Hex	Dec	Description
$0678	1656	Reserved
$0679	1657	Indicates when port 1 firmware is parsing a command
$067A	1658	Indicates when port 2 firmware is parsing a command
$067B	1659	Reserved
$067C	1660	Mouse port: reserved (Reserved)
$067D	1661	Reserved for mouse port
$067E	1662	Reserved
$067F	1663	Reserved (Mouse port: reserved)
$06F8	1784	Reserved
$06F9	1785	Current port 1 command character
$06FA	1786	Current port 2 command character
$06FB	1787	Reserved
$06FC	1788	Mouse port: reserved (Reserved)
$06FD	1789	Reserved for mouse port
$06FE	1790	Reserved
$06FF	1791	Reserved (Mouse port: reserved)
$0778	1912	DEVNO: $n0 = current active port number x 16
$0779	1913	Port 1 flags for echo and auto line feed
$077A	1914	Port 2 flags for each and auto line feed
$077B	1915	Reserved
$077C	1916	Mouse port status byte (Reserved)
$077D	1917	Reserved for mouse port
$077E	1918	Reserved
$077F	1919	Reserved (Mouse port status byte)
$07F8	2040	MSLOT: owner of $C800–$CFFF ($C3, video)
$07F9	2041	Port 1 current printer column
$07FA	2042	Port 2 current line position
$07FB	2043	Reserved
$07FC	2044	Mouse port mode byte (Reserved)
$07FD	2045	Reserved for mouse port
$07FE	2046	Reserved
$07FF	2047	Reserved (Mouse port mode byte)

Table B-4
Auxiliary memory screen hole allocations

Hex	Dec	Description
$0478	1144	Initial port 1 ACIA control register values ($9E)
$0479	1145	Initial port 1 ACIA command register values ($0B)
$047A	1146	Initial port 1 characteristics flags ($40)
$047B	1147	Initial port 1 printer width ($50)
$047C	1148	Initial port 2 ACIA control register values ($16)
$047D	1149	Initial port 2 ACIA command register values ($0B)
$047E	1150	Initial port 2 characteristics flags ($01)
$047F	1151	Initial port 2 line length ($00)
$04F8 through $04FF	1272 through 1279	Reserved
$0578 through $057F	1400 through 1407	Reserved
$05F8 through $05FF	1528 through 1535	Reserved
$0678 through $067F	1656 through 1663	Reserved
$06F8 through $06FF	1784 through 1791	Reserved
$0778 through $077F	1912 through 1919	Reserved
$07F8 through $07FF	2040 through 2047	Reserved

The hardware page

Tables B-5 through B-9 list all the hardware locations available for use in the Apple IIc. These tables have a column at the left that is not present in other tables. This column, labeled *RW,* indicates the action to take at a particular location.

- *R* means read.
- *RR* means read twice in succession.
- *R7* means read the byte and then check bit 7; in the use column, "See if..." refers to the condition represented by bit 7 = 1, unless otherwise specified. Bit 7 has a value of $80, so if the contents of the location are greater than or equal to $80, the bit is on.

  Another way to test bit 7 (the sign bit) is with a BIT instruction, followed by BPL (bit 7 was 0) or BMI (bit 7 was 1).
- *R/W* means to either read or write. For writing, the value is unimportant.
- *W* means to write only. The value is unimportant.
- *N* means not to read or write, because the location is reserved.

An address of the form $C00x refers to the 16 locations from $C000 through $C00F. Labels, when they are shown, are simply memory aids. Some of them correspond to the labels at those addresses in the firmware, others do not. Your program will have to assign a label for it anyway.

Table B-5
Addresses $C000–$C03F

RW	Hex	Dec	Neg dec	Label	Use
R	$C00x			KStrb	Read keyboard data (bits 0-6) and strobe (bit 7)
W	$C000	49152	–16384	80Store	Off: Page2 switches Page 1 and 2
W	$C001	49153	–16383	80Store	On: Page2 switches Page 1 and 1X
W	$C002	49154	–16382	RAMRd	Off: Read main 48K RAM
W	$C004	49156	–16380	RAMWrt	Off: Write in main 48K RAM
W	$C005	49157	–16379	RAMWrt	On: Write in auxiliary 48K RAM
W	$C006	49158	–16378		Reserved
W	$C007	49159	–16377		Reserved
W	$C008	49160	–16376	AltZP	Off: Use main P0, P1, bank-switched RAM
W	$C009	49161	–16375	AltZP	On: Use auxiliary P0, P1, bank-switched RAM
W	$C00A	49162	–16374		Reserved
W	$C00B	49163	–16373		Reserved
W	$C00C	49164	–16372	80Col	Off: 40-column display

Table B-5 (continued)
Addresses $C000–$C03F

RW	Hex	Dec	Neg dec	Label	Use
W	$C00D	49165	–16371	80Col	On: 80-column display
W	$C00E	49166	–16270	AltChar	Off: Display primary character set
W	$C00F	49167	–16369	AltChar	On: Display alternate character set
W	$C01x				Clear keyboard strobe ($C00x bit 7)
R7	$C010	49168	–16368	AKD	See if any key now down; clear strobe
R7	$C011	49169	–16367	RdBnk2	See if using $D000 bank 2 (or 1)
R7	$C012	49170	–16366	RdLCRAM	See if reading RAM (or ROM).
R7	$C013	49171	–16365	RdRAMRd	See if reading auxiliary 48K RAM (or main)
R7	$C014	49172	–16364	RdRAMWrt	See if writing auxiliary 48K RAM (or main)
R	$C015	49173	–16363	RstXInt	Reset mouse X0 interrupt
R7	$C016	49174	–16362	RdAltZP	See if auxiliary P0, P1 and bank-switched RAM
R	$C017	49175	–16361	RstYInt	Reset mouse Y interrupt
R7	$C018	49176	–16360	Rd80Store	See if 80Store on (or off)
R7	$C019	49177	–16359	RstVBl	See if VBlInt off (1); reset it
R7	$C01A	49178	–16358	RdTEXT	See if text (or graphics)
R7	$C01B	49179	–16357	RdMIXED	See if mixed mode switch on
R7	$C01C	49180	–16356	RdPage2	See if Page 2/1X selected (or 1)
R7	$C01D	49181	–16355	RdHiRes	See if high-resolution switch on
R7	$C01E	49182	–16354	RdAltChar	See if alternate character set (or primary)
R7	$C01F	49183	–16353	Rd80Col	See if 80-column hardware on
N	$C020 through	49184	–16352		Reserved (read and write)
N	$C02F	49199	–16337		
W	$C030	49200	–16336	Reserved	
R	$C030	49200	–16336		Toggle speaker
N	$C031 through	49201	–16335		Reserved (read and write)
N	$C03F	49215	–16321		

Table B-6
Addresses $C040–$C05F

RW	Hex	Dec	Neg dec	Label	Use
R7	$C040	49216	–16320	RdXYMsk	See if X0/Y0 mask set
R7	$C041	49217	–16319	RdVBlMsk	See if VBL mask set
R7	$C042	49218	–16318	RdX0Edge	See if interrupt on falling X0 edge
R7	$C043	49219	–16317	RdY0Edge	See if interrupt on falling Y0 edge
N	$C044	49220	–16316		Reserved
N	$C045	49221	–16315)		Reserved
N	$C046	49222	–16314		Reserved
N	$C047	49223	–16313		Reserved
R	$C048	49224	–16312	RstXY	Reset X0/Y0 interrupt flags
N	$C049	49225	–16311		Reserved
N	$C04A	49226	–16310		Reserved
N	$C04B	49227	–16309		Reserved
N	$C04C	49228	–16308		Reserved
N	$C04D	49229	–16307		Reserved
N	$C04E	49230	–16306		Reserved
N	$C04F	49231	–16305		Reserved
R/W	$C050	49232	–16304	TEXT	Off: Graphics display
R/W	$C051	49233	–16303	TEXT	On: Text display
R/W	$C052	49234	–16302	MIXED	Off: Text or graphics only
R/W	$C053	49235	–16301	MIXED	On: Combination text and graphics
R/W	$C054	49236	–16300	Page2	Off: Use Page 1
R/W	$C055	49237	–16299	Page2	On: Display Page 2 (80Store off); store to Page 1X (80Store on)
R/W	$C056	49238	–16298	HiRes	Off: Low resolution
R/W	$C057	49239	–16297	HiRes	On: High resolution; double if 80Col and DHiRes on
N	$C058	49240	–16296		Reserved if IOUDis on ($C07E bit 7=1)
R/W				DisX	Disable (mask) mouse X0/Y0 interrupts
N	$C059	49241	–16295		Reserved if IOUDis on
R/W				EnbXY	Enable (allow) mouse X0/Y0 interrupts
N	$C05A	49242	–16294		Reserved if IOUDis on
R/W				DisVBl	Disable (mask) VBL interrupts
N	$C05B	49243	–16293		Reserved if IOUDis on
R/W				EnVBl	Enable (allow) VBL interrupts
N	$C05C	49244	–16292		Reserved if IOUDis on
R/W				X0Edge	Interrupt on rising edge of X0
N	$C05D	49245	–1629		Reserved if IOUDis on
R/W				X0Edge	Interrupt on falling edge of X0
R/W	$C05E	49246	–16290	DHiRes	If IOUDis on: Set double high-resolution
R/W				Y0Edge	If IOUDis off: Interrupt on rising Y0
R/W	$C05F	49247	–16289	DHiRes	If IOUDis on: Clear double high-resolution
R/W				Y0Edge	If IOUDis off: Interrupt on falling Y0

Table B-7
Addresses $C060-$C07F

RW	Hex	Dec	Neg dec	Label	Use
W	$C06x				Reserved (write)
R7	$C060	4924	–16288	Rd80Sw	See if 80/40 switch down (= 40)
R7	$C061	49249	–16287	RdBtn0	See if mouse button/Open-Apple pressed
R7	$C062	49250	–16286	RdBtn1	See if switch 1/Solid Apple pressed
R7	$C063	49251	–16285	Rd63	See if mouse button not pressed
R7	$C064	49252	–16284	Pdl0	See if hand control button 0 pressed
R7	$C065	49253	–16283	Pdl1	See if hand control button 1 pressed
R7	$C066	49254	–16282	MouX1	See if mouse X1 (direction) is high
R7	$C067	49255	–16281	MouY1	See if mouse Y1 (direction) is high
N	$C068	49256	–16280		
	through				Reserved (write and read)
N	$C06F	49263	–16273		
R/W	$C07x				Trigger paddle timer; reset VBlInt; however, some $C07x are reserved
R/W	$C070	49264	–16272	PTrig	Designated trigger or reset location
N	$C071	49265	–16271		
	through				Reserved
N	$C07D	49277	–16259		
R7	$C07E	49278	–16258	RdIOUDis	See if IOUDis on; trigger paddle timer; reset VBlInt
W				IOUDis	On: Enable access to DHiRes switch; disable $C058–$C05F IOU access
R7	$C07F	49279	–16257	RdDHiRes	See if DHiRes on
W				IOUDis	Off: Disable access to DHiRes switch; enable $C058–$C05F IOU access

Table B-8
Addresses $C080-$C0AF

RW	Hex	Dec	Neg dec	Label	Use
R	$C080	49280	–16256		Read RAM; no write; use $D000 bank 2
RR	$C081	49281	–16255		Read ROM; write RAM; use $D000 bank 2
R	$C082	49282	–16254		Read ROM; no write; use $D000 bank 2
RR	$C083	49283	–16253		Read and write RAM; use $D000 bank 2
N	$C084	49284	–16252		Reserved
N	$C085	49285	–16251		Reserved
N	$C086	49286	–16250		Reserved
N	$C087	49287	–16249		Reserved
R	$C088	49288	–16248		Read RAM; no write; use $D000 bank 1
RR	$C089	49289	–16247		Read ROM; write RAM; use $D000 bank 1
R	$C08A	49290	–16246		Read ROM; no write; use $D000 bank 1
RR	$C08B	49291	–16245		Read and write RAM; use $D000 bank 1
N	$C08C	49292	–16244		Reserved
N	$C08D	49293	–16243		Reserved
N	$C08E	49294	–16242		Reserved
N	$C08F	49295	–16241		Reserved
N	$C090 through	49296	–16240		Reserved
N	$C097	49303	–16233		
R/W	$C098	49304	–16232		Port 1 ACIA transmit/receive register
R/W	$C099	49305	–16231		Port 1 ACIA status register
R/W	$C09A	49306	–16230		Port 1 ACIA command register
R/W	$C09B	49307	–16229		Port 1 ACIA control register
N	$C09C through	49308	–16228		Reserved
N	$C09F	49311	–16225		
N	$C0A0 through	49312	–16224		Reserved
N	$C0A7	49319	–16217		
R/W	$C0A8	49320	–16216		Port 2 ACIA transmit/receive register
R/W	$C0A9	49321	–16215		Port 2 ACIA status register
R/W	$C0AA	49322	–16214		Port 2 ACIA command register
R/W	$C0AB	49323	–16213		Port 2 ACIA control register
N	$C0AC through	49324	–16212		Reserved
N	$C0AF	49327	–16209		

Table B-9
Addresses $C0B0-$C0FF

RW	Hex	Dec	Neg Dec	Label	Use
N	$C0B0	49328	–16208		
	through				Reserved
N	$C0BF	49343	–16193		
N	$C0C0	49344	–16192		
	through				Reserved
N	$C0CF	49359	–16177		
N	$C0D0	49360	–16176		
	through				Reserved
N	$C0DF	49375	–16161		
N	$C0E0	49376	–16160		
	through				Reserved
N	$C0EF	49391	–16145		
N	$C0F0	49392	–16144		
	through				Reserved
N	$C0FF	49407	–16129		

Appendix C

Important Firmware Locations

This appendix lists all significant firmware addresses: entry points, locations containing the addresses of entry points, and locations where machine and device identification bytes reside.

Warning The Monitor firmware entry points are the only *published* entry points in the sense that they are the only ones that will remain in the same locations in future Apple II series computers.

The firmware protocol identification bytes and offsets will work with other Apple II-series computers only if used as directed.

The tables

This appendix supplements the chapter text by specifying three forms of each address: hexadecimal, decimal, and complementary (negative) decimal.

In these tables, some of the addresses are followed by a label. These labels are listed only to help you find the named location in the firmware listings, or to remember the function found at the address. The Apple IIc contains no global label table: your program must assign its own labels to the addresses as required.

There are several types of information at these firmware addresses: actual entry points (labeled *entry*), the low-order byte of an entry point (labeled *offset*), a device or machine identification byte (labeled *ident*), and indicators (labeled *indic*) specifying whether there are optional routines, vector addresses (labeled *vector*), or an RTS instruction location.

Each input/output port has an associated protocol table, as shown in Tables C-1 through C-4. Many of the bytes (labeled *offset*) in the protocol tables are the low-order bytes of addresses of I/O routines for the ports; the high-order byte of these addresses must be $Cn (where n is the port number). This structure is explained in Chapter 3. Although your program must perform some extra processing to use these tables, the benefit is simplified compatible port and slot I/O for all Apple II–series machines.

Port addresses

Addresses for serial ports 1 and 2, output port 3, and mouse input port 4 are shown in the following four tables.

Table C-1
Serial port 1 addresses

Hex	Dec	Neg dec	Label	Type	Description
$C100	49408	–16128		entry	Main port 1 entry point
$C105	49413	–16123		ident	ID byte ($38)
$C107	49415	–16121		ident	ID byte ($18)
$C10B	49419	–16117		ident	Firmware card signature ($01)
$C10C	49420	–16116		ident	Super Serial Card ID ($31)
$C10D	49421	–16115		offset	Low-order PInit address
$C10E	49422	–16114		offset	Low-order PRead address
$C10F	49423	–16113		offset	Low-order PWrite address
$C110	49424	–16112		offset	Low-order PStatus address
$C111	49425	–16111		indic	Non-zero: no optional routines

Table C-2
Serial port 2 addresses

Hex	Dec	Neg dec	Label	Type	Description
$C200	49664	–15872		entry	Main port 2 entry point
$C205	49669	–15867		iden	ID byte ($38)
$C207	49671	–15865		ident	ID byte ($18)
$C20B	49675	–15861		ident	Firmware card ID ($01)
$C20D	49676	–15860		ident	Super Serial Card ID ($31)
$C20D	49677	–15859		offset	Low-order PInit address
$C20E	49678	–15858		offset	Low-order PRead address
$C20F	49679	–15857		offset	Low-order PWrite address
$C210	49680	–15856		offset	Low-order PStatus address
$C211	49681	–15855		indic	Non-zero: no optional routines

Table C-3
Video firmware addresses

Hex	Dec	Neg Dec	Label	Type	Description
$C300	49920	–15616		entry	Main video entry point (output only)
$C305	49925	–15611	C3KeyIn	ident	ID byte ($38)
$C307	49927	–15609	C3COut1	ident	ID byte ($18)
$C30B	49931	–15605		ident	Firmware card signature ($01)
$C30C	49932	–15604		ident	80-column card ID ($88)
$C30D	49933	–15603		offset	Low-order PInit address
$C30E	49934	–15602		offset	Low-order PRead address
$C30F	49935	–15601		offset	Low-order PWrite address
$C310	49936	–15600		offset	Low-order PStatus address
$C311	49937	–15599	MoveAux	entry	Routine for main/auxiliary control swapping (also called *AuxMove*)

Table C-4
Mouse port addresses

Hex	Dec	Neg dec	Label	Type	Description
$C400	50176	–15360		entry	Main mouse entry point
$C405	50181	–15355		ident	ID byte ($38)
$C407	50183	–15353		ident	ID byte ($18)
$C40B	50187	–15349		ident	Firmware card signature ($01)
$C40C	50188	–15348		type	X-Y pointing device ID ($20)
$C40D	50189	–15347		offset	Low-order PInit address
$C40E	50190	–15346		offset	Low-order PRead address
$C40F	50191	–15345		offset	Low-order PWrite address
$C410	50192	–15344		offset	Low-order PStatus address
$C411	50193	–15343		indic	Optional routines follow ($00)
$C412	50194	–15342	SetMouse	offset	Low-order SetMouse address
$C413	50195	–15341	ServeMouse	offset	Low-order ServeMouse address
$C414	50196	–15340	ReadMouse	offset	Low-order ReadMouse address
$C415	50197	–15339	ClearMouse	offset	Low-order ClearMouse address
$C416	50198	–15338	PosMouse	offset	Low-order PosMouse address
$C417	50199	–15337	ClampMouse	offset	Low-order ClampMouse address
$C418	50200	–15336	HomeMouse	offset	Low-order HomeMouse address
$C419	50201	–15335	InitMouse	offset	Low-order InitMouse address

Memory expansion

The memory expansion version of the Apple IIc supports the mouse in port 7. This means that the firmware entry points are $C7XX addresses, instead of $C4XX address; change the 4's to 7's in Table C-4.

Other video and I/O firmware addresses

Miscellaneous firmware addresses are listed in Table C-5.

Table C-5
Apple IIc enhanced video and miscellaneous firmware

Hex	Dec	Neg dec	Label	Type	Description
$C600	50688	–14848		entry	Disk drive firmware entry point
$C700	50944	–14592		entry	External disk startup routine
$C803	51203	–14333	NewIRQ	entry	IRQ handling routine

Memory expansion $C700 supports the mouse in the memory expansion version.

Applesoft BASIC interpreter addresses

The addresses of Applesoft BASIC entry points are listed in the *Applesoft BASIC Programmer's Reference Manual.* The Applesoft interpreter occupies ROM addresses from $D000 through $F7FF.

Monitor addresses

Table C-6 lists the Monitor entry points, machine identifier bytes, interrupt vectors, and the address of a known RTS instruction.

Table C-6
Apple IIc monitor entry points and vectors

Hex	Dec	Neg dec	Label	Type	Description
$F800	63488	–2048	PLOT	entry	Plots a low-resolution block
$F819	63513	–2023	HLine	entry	Draws low-resolution horizontal line
$F828	63528	–2008	VLine	entry	Draws low-resolution vertical line
$F832	63538	–1998	ClrScr	entry	Clears low-resolution screen
$F836	63542	–1994	ClrTop	entry	Clears top 40 low-resolution lines
$F864	63588	–1948	SetCol	entry	Sets low-resolution color (Table 5-4)
$F871	63601	–1935	SCRN	entry	Reads color of low-resolution block
$F941	63809	–1727	PrntAX	entry	Displays A and X in hex
$F94A	63818	–1718	PrBl2	entry	Sends X blanks to output

Table C-6 (continued)
Apple IIc monitor entry points and vectors

Hex	Dec	Neg dec	Label	Type	Description
$FA47	63845	–1691	NewBRK	entry	Apple IIc break handler
$FA62	64098	–1438	Reset	entry	Hardware reset routine
$FB1E	64386	–1150	PRead	entry	Reads hand controller position
$FB6F	64467	–1169	SetPwrC	entry	Routine to create power-up byte
$FBB3	64535	–1101		ident	Machine identification byte
$FBC0	64548	–1088		ident	Machine identification byte
$FBDD	64477	–1059	Bell1	entry	Sends 1-kHz beep to speaker
$FC42	64578	–958	ClrEOP	entry	Clears from cursor to bottom
$FC58	64600	–936	HOME	entry	Clears from cursor to upper left
$FC9C	64668	–868	ClrEOL	entry	Clears from cursor to end of line
$FC9E	64670	–866	ClEOLZ	entry	Clears from BASL to end of line
$FCA8	64680	–856	WAIT	entry	Delays for time specified by A
$FD0C	64780	–756	RdKey	entry	Displays cursor, jumps to KSW
$FD1B	64795	–741	KeyIn	entry	Waits for keypress, reads key
$FD35	64821	–715	RdChar	entry	Gets input, interprets ESC codes
$FD67	64871	–665	GetLnZ	entry	Sends CR to output, goes to GetLn
$FD6A	64874	–662	GetLn	entry	Displays prompt, gets input line
$FD6F	64879	–657	GetLn1	entry	No prompt; gets input line
$FD8B	64907	–629	CROut1	entry	Clears to end of line, calls CROut
$FD8E	64910	–626	CROut	entry	Sends CR to output
$FDDA	64986	–550	PrByte	entry	Sends A to output
$FDE3	64995	–541	PrHex	entry	Displays low nibble of A in hex
$FDED	65005	–531	COut	entry	Jumps to CSW
$FDF0	65008	–528	COut1	entry	Displays A, advances cursor
$FE2C	65068	–468	MOVE	entry	Copies memory elsewhere
$FE36	65078	–458	VERIFY	entry	Compares two blocks of memory
$FF2D	65325	–211	PrErr	entry	Sends ERR to output; beeps
$FF3A	65338	–198	Bell	entry	Sends CONTROL-G to output
$FF3F	65343	–193	IORest	entry	Loads $45–$49 into registers
$FF4A	65354	–182	IOSave	entry	Stores A, X, Y, P, S at $45–$49
$FF58	65368	–168	IORTS	RTS	Location of known RTS instruction
$FF69	65385	–151	Monitor	entry	Standard Monitor entry point
$FFFA	65530	–6		vector	Low-order NMI vector (unused)
$FFFB	65531	–5		vector	High-order NMI vector (unused)
$FFFC	65532	–4		vector	Low-order reset vector ($62)
$FFFD	65533	–3		vector	High-order reset vector ($FA)
$FFFE	65534	–2	IRQVect	vector	Low-order IRQ vector ($03)
$FFFF	65535	–1		vector	High-order IRQ vector ($CB)

Appendix D

Operating Systems and Languages

This appendix is an overview of the characteristics of operating systems and languages when run on the Apple IIc. It is not intended to be a complete description. For more information, refer to the manuals that are provided with each product.

Operating systems

This section discusses the operating systems that the Apple IIc works with CP/M, and any other operating system that requires an interface card, does not work on the Apple IIc.

ProDOS

ProDOS is the preferred disk operating system for the Apple IIc. It supports startup from the external disk drive (on original Apple IIc's with the command PR#7), interrupts, and all other hardware and firmware features of the Apple IIc.

DOS

The Apple IIc works with DOS 3.3. Its built-in disk drive hardware and firmware can also access DOS 3.2 disks by using the *BASICS* disk. DOS support is provided for the sake of Apple II series compatibility; neither version of DOS takes full advantage of all the features of the Apple IIc.

Pascal Operating System

Versions 1.2 and later of the Pascal Operating System use the 80/40 switch and the interrupt features of the Apple IIc, while remaining compatible with the other Apple II series computers.

While the Apple IIc works with Pascal 1.1, this version of the Pascal Operating System does not use the 80/40 switch or handle interrupts.

The Apple IIc does not work with Pascal 1.0, because the I/O firmware entry points of that version of the operating system are rigidly defined (rather than being accessed via a table), and the Apple IIc's built-in firmware does not correspond to these entry points.

Languages

This section discusses using Apple programming languages with the Apple IIc. It is also a guide to using this reference manual with these languages.

Applesoft BASIC

The programming examples in this manual are almost entirely in assembly language, and so most addresses and values are given in hexadecimal notation.

Use a PEEK in BASIC (instead of LDA in assembly language) to read a location, and a POKE (instead of STA) to write to a location. The values used by Applesoft must be in decimal, so you will have to convert hexadecimal values given in this manual to decimal. (Several tables in this manual include decimal equivalents to make the job easier for you.)

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

Integer BASIC

You will have to run a version of DOS in your Apple IIc to use Integer BASIC. ProDOS does not support Integer BASIC.

Pascal

The Pascal language runs on the Apple IIc under versions 1.1 or later of the Pascal Operating System. However, for best performance, use Pascal versions 1.2 or later.

Fortran

Fortran runs under version 1.1 of the Pascal Operating System, which does not detect or use certain Apple IIc features, such as the 80/40 switch or auxiliary memory. Therefore, Fortran does not take advantage of these features either.

Logo II

Apple Logo II works under ProDOS on Apple II series machines with at least 128K of memory. Logo II is a version of the Logo language originally developed from the LISP (LISt Processing) language at MIT as a language to be used for learning. Logo II takes advantage of the Apple II's graphics and retains much of the power and flavor of LISP without LISP's somewhat cryptic syntax.

Appendix E

Interrupts

This appendix describes the sources of interrupts on the Apple IIc, how the firmware handles the interrupts, and how to use interrupt-driven features directly in those rare cases when the firmware cannot meet your needs.

Warning If you use interrupt hardware directly, instead of using the built-in interrupt-handling firmware, you can't be sure that your programs will be compatible with possible future Apple II series computers or revisions.

Introduction

This section describes interrupts and their effects on the Apple IIc hardware.

What is an interrupt?

An **interrupt** is a signal that a computer uses to know when to stop what it's doing so it can quickly handle a time-dependent task. For example, the Apple IIc mouse sends an interrupt to the computer every time it moves. This lets the system keep track of the mouse's position and maintain smooth movement of the pointer on the screen.

When an interrupt occurs, control passes to an interrupt handler, which must record the exact state of the computer at the moment of the interrupt, determine the source of the interrupt, and take appropriate action. It is important that the computer preserve a "snapshot" of its state when interrupted, so that when it continues later with what it was doing, those conditions can be restored.

Interrupts on Apple II computers

Interrupts have not always been fully supported on the Apple II. All versions of Apple's DOS, as well as the Monitor program, rely on the integrity of location $45, which the built-in interrupt handler has always destroyed by saving the accumulator in it. Most versions of Pascal simply do not work with interrupts enabled.

The Apple IIc built-in interrupt handler now saves the accumulator on the stack instead of in location $45. DOS 3.3, ProDOS, Pascal 1.2 (or later versions), and the Monitor all work with interrupts on the Apple IIc.

You should use either ProDOS or Pascal 1.2 (or later versions) if you want interrupt-using software to work on the Apple IIe and the Apple II Plus. Both operating systems have full interrupt support built in.

Interrupts are effective only if they are enabled most of the time since interrupts that occur while interrupts are disabled cannot be detected. Because of the critical timing of disk read and write operations, Pascal, DOS 3.3, and ProDOS turn off interrupts while accessing the disk. Thus it is important to remember that while a disk drive is being accessed, all the interrupt sources discussed below are turned off.

On the Apple IIe only, interrupts are periodically turned off while 80-column screen operations are being performed. This is most noticeable while the screen is scrolling. Also, most peripheral cards used in the Apple IIe disable interrupts while reading and writing.

Interrupt handling on the 65C02

From the point of view of the 65C02, there are three possible causes of interrupts.

1. The IRQ line on the microprocessor can be pulled low if 65C02 interrupts are not masked (that is, the CLI instruction has been used). This is the standard technique that devices use when they need immediate attention.
2. The processor executes a break (BRK, opcode $00) instruction.
3. A nonmaskable interrupt (NMI) occurs. Because the NMI line in the Apple IIc's 65C02 is not used, this never happens on the Apple IIc.

The first two possibilities cause the 65C02 to save the current program counter and status byte on the stack and then jump to the routine whose address is stored in $FFFE and $FFFF. The sequence performed by the 65C02 is:

1. If an IRQ occurs, finish executing the current instruction. (If a BRK occurs, the current instruction is already finished.)
2. Push the high byte of the program counter onto the stack.
3. Push the low byte of the program counter onto the stack.
4. Push the program status byte onto the stack.
5. Jump to the address stored in $FFFE, $FFFF—that is, JMP ($FFFE).

The different sources of interrupt signals are discussed below.

The interrupt vector at $FFFE

In the Apple IIc there are three separate regions of memory that contain address $FFFE: the built-in ROM, the bank-switched memory in main RAM, and the bank-switched memory in auxiliary RAM. The vector at $FFFE in the ROM points to Apple IIc's built-in interrupt handling routine. You should generally use the built-in interrupt handler, rather than writing your own, because of the complexity of interrupts on the Apple IIc.

When you initialize the mouse or serial communication firmware, copies of the ROM's interrupt vector are placed in the interrupt vector addresses in both main and auxiliary bank-switched memory. If you plan to use interrupts and the bank-switched memory without the mouse or communication firmware, you must copy the ROM's interrupt vector yourself.

The built-in interrupt handler

The built-in interrupt handler is responsible for determining whether a BRK or an IRQ interrupt occurred. If it was an IRQ interrupt, it decides whether the interrupt should be handled internally, handled by the user, or simply ignored.

The built-in interrupt-handling routine records the current memory configuration, then sets up its own standard memory configuration so that a user's interrupt handler knows the precise memory configuration when it is called.

Next the handler checks to see if the interrupt was caused by a break instruction, and if it was, handles it as described later in this appendix.

If the interrupt was not caused by a BRK, the handler checks for interrupts that it knows how to handle (for example, a properly initialized mouse) and handles them.

Depending on the state of the system, it either ignores other interrupts or passes them to a user's interrupt handling routine whose address is stored at $03FE and $03FF of main memory.

After handling an interrupt itself, or after the user's handler returns (with an RTI), the built-in interrupt handler restores the memory configuration, and then does an RTI to restore processing to where it was when the interrupt occurred. Table E-1 illustrates this whole process. Each of the steps is explained in detail in the sections that follow.

Table E-1
Interrupt-handling sequence

Interrupted program	Processor	Built-in handler	User's handler
Program →	Push address Push status		
	JMP ($FFFE) →	Save old and set new memory configuration	
		If BRK, then go to break handler ($FA47) →	
		Our interrupt?	
		NO: Push address Push status JMP ($03FE) →	Handle interrupt
			. . .
		YES: Handle it	. . .
		Restore memory configuration ←	RTI
	Pull status ←	RTI	
Program ←	Pull address		

Saving the memory configuration

The built-in interrupt handler saves the state of the system, and sets it to a known state according to these rules:

- If 80Store and Page2 are on, then it switches in text Page 1 (Page2 off) so that main screen holes are accessible.
- It switches in main memory for reading (RAMRd off).
- It switches in main memory for writing (RAMWrt off).
- It switches in ROM addresses $D000–$FFFF for reading (RdLCRAM off).
- It switches in main stack and zero page (AltZP off).
- It preserves the auxiliary stack pointer, and restores the main stack pointer.

- It preserves the current ROM state and switches in the ROM bank 1.

❖ *Note:* Because main memory is switched in, all memory addresses used later in this appendix are in main memory unless otherwise specified.

Managing main and auxiliary stacks

Because the Apple IIc has two stack pages, the firmware has established a convention that allows the system to be run with two separate stack pointers. Two bytes in the auxiliary stack page are to be used as storage for inactive stack pointers: $0100 for the main stack pointer when the auxiliary stack is active, and $0101 for the auxiliary stack pointer when the main stack is active.

When a program that uses interrupts switches in the auxiliary stack for the first time, it should place the value of the main stack pointer at auxiliary stack address $0100, and initialize the auxiliary stack pointer to $FF (the top of the stack). When it subsequently switches from one stack to the other, it should save the current stack pointer before loading the pointer for the other stack.

When an interrupt occurs while the auxiliary stack is switched in, the current stack pointer is stored at $0101, and the main stack pointer is retrieved from $0100. Then the main stack is switched in for use. After the interrupt has been handled, the stack pointer is restored to its original value.

User's interrupt handler at $03FE

You can set up screen hole locations to indicate that the user's interrupt handler should be called when certain interrupts occur. To do this, place your interrupt handler's address at $03FE and $03FF in main memory, low byte first.

The user's interrupt handler should do the following:

- Verify that the interrupt came from the expected source. The following sections describe how this should be done for each interrupt source.
- Handle the interrupt as desired.
- Clear the interrupt, if necessary. The following sections describe how to clear the interrupts.
- Return with an RTI.

If your interrupt handler needs to know the memory configuration at the time of the interrupt, it can check the encoded byte stored four bytes down on the stack. This byte is explained later in this appendix.

In general there is no guaranteed *maximum* response time for interrupts. This is because the system may be doing a disk operation, which could last for several seconds.

Once the built-in interrupt handler has been called, it takes about 250 to 300 microseconds for it to call your interrupt-handling routine. After your routine returns, it takes 40 to 140 microseconds to restore memory and return to the interrupted program.

If memory is in the standard state when the interrupt occurs, the total overhead for interrupt processing is about 150 microseconds less than if memory is in the worst possible state (80Store and Page2 on, auxiliary memory switched in for reading and writing, auxiliary bank-switched memory page $02 switched in for reading and writing).

Handling break instructions

After the interrupt handler has set the memory configuration, it checks to see if the interrupt was caused by a BRK (opcode $00) instruction. (If it was, bit 4 of the processor status byte is a 1.) If so, it jumps to a break-handling routine, which saves the state of the computer at the time of the break as follows:

Information	Location
Program counter (low byte)	$3A
Program counter (high byte)	$3B
Encoded memory state	$44
Accumulator	$45
X register	$46
Y register	$47
Status register	$48

Finally, the break routine jumps to the routine whose address is stored at $03F0 and $03F1.

The encoded memory state in location $44 can be interpreted as follows:

Bit 7 = 0
Bit 6 = 1 if 80Store and Page2 both on
Bit 5 = 1 if auxiliary RAM switched in for reading
Bit 4 = 1 if auxiliary RAM switched in for writing
Bit 3 = 1 if bank-switched RAM being read
Bit 2 = 1 if bank-switched $D000 page $01 switched in
Bit 1 = 1 if bank-switched $D000 page $02 switched in
Bit 0 = 0

Sources of interrupts

The Apple IIc can receive interrupts from many different sources. Each source is enabled and used slightly differently from the others. There are two basic sources of interrupts: use of the mouse, and actions affecting the two 6551 ACIA circuits (the chips that control serial communication). How to use these sources of interrupts in conjunction with the built-in interrupt handler is discussed later in this appendix.

Mouse use can cause interrupts when

- the mouse is moved in the horizontal (X) direction
- the mouse is moved in the vertical (Y) direction
- the mouse button is pressed

Interrupts can also be generated every 1/60 second by the rising edge of the vertical blanking signal. This is called the *vertical blanking* (VBL) interrupt and is synchronized with a signal used for the video display.

Actions affecting the ACIA circuits can cause interrupts when

- a key is pressed (the firmware can use this interrupt to buffer keystrokes, or it can pass the interrupt on to the user)
- either ACIA has received a byte of data from its port (the firmware can use this interrupt to buffer data or it can pass the interrupt on to the user)
- pin 5 of either serial port changes state (device ready/not ready to accept data) (when the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data)

- either ACIA is ready to accept another character to be transmitted (when the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data)
- the keyboard strobe is cleared (the firmware absorbs this interrupt)

An interrupt can also be generated by a device attached to the external disk drive port. The firmware can pass this interrupt on to the user.

Firmware handling of interrupts

The following sections discuss how the various sources of interrupts should be used together with the built-in interrupt handler.

Firmware for mouse and VBL

As described in Chapter 9, the mouse can be initialized (by the SetMouse call) to nine different modes that enable one or more sources of interrupts. In transparent mode, the interrupts are entirely handled by the built-in interrupt handler; the other modes require a user-installed interrupt handler.

When the mouse is initialized, the interrupt vector is copied to addresses $FFFE and $FFFF in main and auxiliary bank-switched RAM. This permits mouse interrupts with any memory configuration.

When the mouse is active, possible sources of interrupts are those listed earlier in this appendix as resulting from mouse use.

When an interrupt occurs, the built-in interrupt handler determines whether that particular interrupt source was enabled (by the SetMouse call). If so, the user's interrupt handler, whose address is stored at $03FE, is called.

The user's interrupt handler should first call ServeMouse to determine the source of the interrupt. This call updates the mouse status byte at $077C and returns with the carry bit clear if mouse movement, button, or vertical blanking was the source of the interrupt.

The values of this mouse status byte at $077C ($077F in the memory expansion IIc) are as follows:

Bit	1 means that
3	Interrupt was from vertical blanking
2	Interrupt was from button
1	Interrupt was from mouse movement

If the interrupt was due to mouse movement or button, the user's interrupt handler should then do a call to ReadMouse. This causes the mouse coordinates and status to be updated as follows:

$047C	Low byte of X coordinate
$04FC	Low byte of Y coordinate
$057C	High byte of X coordinate
$05FC	High byte of Y coordinate
$077C	Button and movement status

Bit	Means
7	0 = button up; 1 = button down
6	0 = button up on last ReadMouse 1 = button down on last ReadMouse
5	0 = no movement since last ReadMouse 1 = movement since last ReadMouse
3–1	Always set to 0 (interrupt cleared)

After the interrupt has been handled, the routine should terminate with an RTI.

Remember that interrupts may be missed during disk accesses.

If you turn on mouse interrupts without initializing the mouse, the built-in interrupt handler will absorb the interrupts. If you want to handle mouse interrupts yourself, you must write your own interrupt handler and place vectors to it at addresses $FFFE and $FFFF in bank-switched RAM. Interrupts will be ignored whenever the $D000-$FFFF ROM is switched in.

Firmware for keyboard interrupts

The Apple IIc hardware is able to generate an interrupt when a key is pressed. The firmware is able to buffer up to 128 keystrokes, completely transparently, when properly enabled to do so. It saves them in the second half of page $08 of auxiliary memory. After the buffer is full, subsequent keystrokes are ignored. Because interrupts are only generated when keypresses occur, characters generated by the auto-repeat feature are not buffered. They can, however, be read when the buffer is empty.

Once keyboard buffering has been turned on, the next key should be read by calling RdKey ($FD0C).

Warning Do not call the buffer reading routine directly. Its entry address will not be the same in future versions of the computer.

The special characters Control-S (stop list) and Control-C (stop Applesoft execution) do not work while keyboard buffering is turned on. A new keystroke, Solid Apple-Control-X, clears the buffer.

Using keyboard buffering firmware

Keyboard buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself this way:

Memory expansion The Apple IIc that supports memory expansion places the keyboard screen holes in different locations from those used in earlier versions. For the memory expansion IIc, change all $nnnF addresses to $nnnC (that is, change $05FF to $05FC).

1. Disable processor interrupts (SEI).
2. Set location $05FA to $80. This tells the firmware to buffer keystrokes without calling the user's interrupt handler.
3. Set locations $05FF and $06FF to $80. These are pointers to where in the buffer the next keystroke will be stored and where the next will be read from, respectively.
4. Turn on the ACIA for port 2 by setting the low nibble of $C0AA to the value $0F. For example:

```
LDA $C0AA    Read port 2 ACIA command register
ORA #$0F     Set low nibble to $0F
STA $C0AA    Set port 2 ACIA command register
```

   If you are using the serial ports at the same time, just set the low bit of $C0AA to 1. This prevents receiver interrupts from being turned off.

   A PR#2 or IN#2 or the equivalent will shut off keyboard interrupts.

5. Enable processor interrupts (CLI).

Using keyboard interrupts through firmware

Keyboard interrupts are received through the ACIA for port 2. They can be enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location $05FA to $C0. This tells the firmware to identify a keystroke interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 2 by setting the low nibble of $C0AA to the value $0F. For example:

```
LDA $C0AA    Read port 2 ACIA command register
ORA #$0F     Set low nibble to $0F
STA $C0AA    Set port 2 ACIA command register
```

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify the keyboard as the interrupt source by reading location $04FA. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 2, bit 7 is set. If the interrupt was caused by a keystroke, bit 6 is set and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting $04FA to $00.

Using external interrupts through firmware

Pin 9 of the external disk drive connector (EXTINT) can be used to generate interrupts through the ACIA for port 1. It can be used as a source of interrupts (on a high-to-low transition) if enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location $05F9 to $C0. This tells the firmware to identify an external interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 1 by setting the low nibble of $C09A to the value $0F. For example:

```
LDA $C09A    Read port 1 ACIA command register
ORA #$0F     Set low nibble to $0F
STA $C09A    Set port 1 ACIA command register
```

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify this interrupt by reading location $04F9. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 1, bit 7 is set. If the interrupt was caused by the external interrupt line, bit 6 is clear and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting $04F9 to $00.

Firmware for serial interrupts

The Apple IIc hardware is able to generate interrupts both when the ACIA receives data and when it is ready to send data. The built-in interrupt handler responds to incoming data only. The firmware is able to buffer up to 128 incoming bytes of serial data from either serial port. After the buffer is full, data are ignored. Only one port can be buffered at a time. The following sections assume that the serial port to be buffered is already initialized, as explained in Chapter 8.

Using serial buffering transparently

Serial buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself, as follows:

Memory expansion

For the memory expansion IIc, change all $nnnF addresses to $nnnC and change the $0D value to $09.

1. Disable processor interrupts (SEI).
2. Set location $04FF to $C1 to buffer port 1, or to $C2 to buffer port 2.
3. Set locations $057F and $067F to $00. These are pointers to the next byte in the buffer to be used and the next character to be read from the buffer, respectively.
4. Turn on the ACIA for the port by setting the low nibble of $C09A for port 1 or $C0AA for port 2 to $0D. For example:

```
LDA $C09A    Read port 1 ACIA command register
AND $F0      Clear low nibble
ORA #$0D     Set low nibble to $0D
STA $C09A    Set port 1 ACIA command register
```

   The 0 in bit 1 of the command register enables receiver interrupts; thus an interrupt is generated when a byte of data is received.

5. Enable processor interrupts (CLI).

When serial port buffering is thus enabled, normal reads from the serial port firmware fetch data from the buffer rather than directly from the ACIA.

Using serial interrupts through firmware

It is also possible to use the firmware to call the user interrupt handler whenever a byte of data is read by the ACIA. In this mode buffering is not performed by the firmware.

Memory expansion

For the memory expansion IIc, change all $nnnF addresses to $nnnC and change the $0D value to $09.

1. Disable processor interrupts (SEI).
2. Set location $04FF to a value other than $C1 or $C2.
3. Turn on the ACIA for the port by setting the low nibble of $C09A for port 1 or $C0AA for port 2 to $0D. For example:

```
LDA $C09A    Read port 1 ACIA command register
AND $F0      Clear low nibble
ORA #$0D     Set low nibble to $0D
STA $C09A    Set port 1 ACIA command register
```

   The 0 in bit 1 of the command register enables receiver interrupts; thus an interrupt is generated when a byte of data is received.

4. Enable processor interrupts (CLI).

When a serial port is thus enabled, the user's interrupt handler is called each time the port receives a byte of data. The status byte saved by the firmware ($04F9 for port 1; $04FA for port 2) has the high bit set if the interrupt occurred on that port. Bit 3 is set if the interrupt was due to a received byte of data.

The interrupt handler should clear the interrupt by clearing bits 7 and 3 of that port's status byte ($04F9 for port 1; $04FA for port 2).

Transmitting serial data

The serial firmware does not implement buffering for serial output. Instead it waits for two conditions to be true before transmitting a character:

- The ACIA's transmit register must be ready to accept a character. This is true if bit 4 of the ACIA's status register is 1.
- The device must signal that it is ready to accept data. This is true if bit 5 of the ACIA's status register is 0. Bit 5 is 0 if pin 5 of the port's connector is also 0.

When the serial firmware is active, a change of state on pin 5 of that port generates an interrupt. That interrupt is absorbed, but the data remain in bit 5 of the status register. Interrupts from the ACIA's transmit register are normally disabled.

A loophole in the firmware

So that programs can make use of interrupts on the ACIAs without affecting mouse interrupt handling, there is a tiny loophole purposely left in the built-in interrupt handler. If transmit interrupts are enabled on the ACIA—that is, if bits 3, 2, and 0 of the ACIA's command register have the values 0, 1, and 1, respectively—then control is passed to the user's interrupt handler if the interrupt is not intended for the mouse (movement, button, or VBL).

This means that you can write more sophisticated serial interrupt-handling routines than the limited firmware space could provide (such as printer spooling). The firmware will still set memory to its standard state, handle mouse interrupts, and restore memory after your routine is finished.

When you receive the interrupt, neither ACIA's status register has been read. You are fully responsible for checking for interrupts on both ACIAs, determining which of the four interrupt sources on each ACIA caused the interrupt, and how to handle them. Refer to the 6551 specification for more details. The built-in firmware itself is an excellent example of how interrupts on the ACIA can be handled.

Bypassing the interrupt firmware

The following sections give further details on using interrupts on the Apple IIc computer without using the built-in interrupt handler.

A method of handling mouse interrupts directly is described in Chapter 9.

Using mouse interrupts without the firmware

To use mouse interrupts without the firmware, as mentioned above, you must set your own interrupt vectors. If the $D000–$FFFF ROM is ever switched in, the built-in interrupt handler will absorb the mouse interrupts.

Tables E-2 and E-3 show how to activate and read mouse interrupts without using the firmware. Remember to disable interrupts (SEI) before enabling mouse interrupts, then turn them on when done (CLI).

Table E-2
Activating mouse interrupts

To activate interrupts on	Enable IOU access	Select source	Enable source	Disable IOU access
Mouse X (rising edge)	STA $C079	STA $C05C	STA $C059	STA $C078
Mouse X (falling edge)	STA $C079	STA $C05D	STA $C059	STA $C078
Mouse Y (rising edge)	STA $C079	STA $C05E	STA $C059	STA $C078
Mouse Y (falling edge)	STA $C079	STA $C05F	STA $C059	STA $C078
VBL	STA $C079		STA $C05B	STA $C078

Table E-3
Reading mouse interrupts

To read interrupts from	Read direction (A.S.A.P)	Determine source	Handle it	Return
Mouse X	LDA $C066	LDA $C015 (bit 7=1 if true)	...	RTI
Mouse Y	LDA $C067	LDA $C017 (bit 7=1 if true)	...	RTI
VBL		LDA $C019 (bit 7=1 if true)	...	RTI

The mouse direction data read from $C066 and $C067 are guaranteed valid for at least 40 microseconds, and average duration is at least 200 microseconds, so you should read the direction as soon as possible.

Using ACIA interrupts without the firmware

To use ACIA interrupts without the firmware, you must set your own interrupt vectors. If the $D000–$FFFF ROM is ever switched in, the built-in interrupt handler will handle the interrupt as determined by certain mode bytes.

When writing your serial interrupt handler, refer to Figures 11-31 through 11-33 and to the Synertek 6551 ACIA specification. As shown in Chapter 11, the ACIAs have the following connections:

Port 1 DSR line connected to the EXTINT line on the external disk port.
DCD line connected to pin 5 of port 1 connector.

Port 2 DSR line goes high when a key is pressed.
DCD line connected to pin 5 of port 2 connector.

The ACIA registers have the following addresses:

Port 1		**Port 2**	
Data register	= $C098	Data register	= $C0A8
Status register	= $C099	Status register	= $C0A9
Command register	= $C09A	Command register	= $C0AA
Control register	= $C09B	Control register	= $C0AB

Appendix F

Apple II Series Differences

This appendix compares the Apple IIc to the Apple IIe, Apple II Plus, and Apple II. It does not contain an exhaustive list of differences, but it does mention those differences most likely to affect the accuracy of programs, displays, and instructions created for end users of two or more Apple II series models.

Overview

The differences between the Apple II series computers can be expressed as a series of equations: this computer equals that one plus or minus certain features.

The following equations compare each model of Apple II series with its predecessor in terms of functional equivalence, not literal equality. For example,

Apple II Plus = Apple II – Integer BASIC firmware

does not mean that Integer BASIC firmware can be removed from the Apple II—just that the one machine functions as if it were the other without such firmware.

Apple II Plus = II
+ Autostart ROM
+ Applesoft firmware
+ 48K RAM standard
– old Monitor ROM
– Integer BASIC firmware

Apple IIe = II Plus
+ Apple Language Card (with 16K of RAM)
+ 80-column (enhanced) video firmware
+ built-in diagnostics
+ full ASCII keyboard
+ internal power light
+ FCC approval
+ improved back panel
+ 9-pin back panel game connector
+ auxiliary slot (with possibility of 80-column text card and extra 64K RAM)
– slot 0
+ interrupt support in firmware (enhanced Apple IIe)
+ Mini-Assembler in firmware (enhanced Apple IIe)

Apple IIc = IIe
+ extended 80-column text card
+ 80/40 switch
+ keyboard switch
+ disk-use light
+ disk controller port
+ disk drive
+ mouse port
+ serial printer port
+ serial communication port
+ built-in port firmware
+ video expansion connector
– removable cover
– slots 1 to 7
– auxiliary slot
– internal power light
– cassette I/O connectors
– internal game I/O connector (hence no game output)
– auxiliary video pin
– Monitor cassette support
+ Mini-Assembler in firmware (Apple IIc with UniDisk 3.5 support)
+ Smartport in firmware (UniDisk 3.5 and memory expansion Apple IIc)
+ memory expansion card support (memory expansion Apple IIc)

Type of processor

The processor in the Apple II and II Plus is the 6502. The original Apple IIe uses a 6502A. The Apple IIc and enhanced Apple IIe both use the 65C02: this is a redesigned CMOS CPU that has 27 new instructions, new addressing modes, and for some instructions a differing execution scheme and machine cycle counts (see Appendix A).

Programs written for the Apple IIc will run on the earlier machines only if they do not contain instructions unique to the 65C02, or depend on shared instructions whose cycle times differ. Programs should also use only published entry points in the Monitor firmware to allow maximum compatibility between different Apple II series computers.

Machine identification

Identification of Apple II series computers is as shown in Table F-1.

Table F-1
Apple II series identification bytes

Machine	$FBB3	$FB1E	$FBC0	$FBBF
Apple II	$38			
Apple II Plus	$EA	$AD		
Apple IIe	$06		$EA	
Apple IIe (enhanced)	$06		$E0	
Apple IIc	$06		$00	$FF
Apple IIc (UniDisk 3.5 support)	$06		$00	$00
Apple IIc (memory expansion)	$06		$00	$03
Apple III in Apple II emulation mode	$EA	$8A		

Any future Apple II series computer or ROM release will have different values in these locations. Machine identification routines are available from Apple Vendor Technical Support.

The MachID byte for ProDOS ($BF98 on the global page) will have bit 3 set to 0 if the computer is an Apple II, II Plus, IIe, or III, and to 1 if the computer is not one of these machines. In an Apple IIc, bits 7 and 6 are also set to binary 10.

Bits 7 and 6 set to binary 10 indicate that a computer is Apple IIe and IIc compatible, regardless of the value of bit 3.

Memory structure

This section compares the memory organization of the Apple IIc with that of the Apple II, II Plus, and IIe. These machines differ in RAM space, ROM space, slot or port address space, and hardware page use.

Amount and address ranges of RAM

The Apple II could have as little as 4K of RAM at the time of purchase, and could be upgraded to as much as 48K of RAM.

The Apple II Plus has 48K of RAM ($0000 through $BFFF) as a standard feature. With the addition of an Apple Language Card, a 48K Apple II or II Plus could be expanded to have 64K of RAM.

The Apple IIe has a full 64K of RAM. The top 12K addresses overlap with the ROM addresses $D000 through $FFFF. There is an additional bank-switched area of 4K from $D000 through $DFFF. This arrangement is equivalent to an Apple II Plus with an Apple Language Card installed. A program selects between the RAM and ROM address spaces and between the $Dxxx banks by changing soft switches located in memory.

With an Extended 80-Column Text Card installed in its auxiliary slot, an Apple IIe has an additional 64K of RAM available, although no more than half of the 128K of RAM space is available at any given time. Soft switches located in memory control these address space selections.

The RAM in the Apple IIc is equivalent to the RAM in an Apple IIe with an Extended 80-Column Text Card. The optional memory expansion card can add as much as 1Mb of RAM to the IIc in 256K steps.

Amount and address ranges of ROM

The Apple II has 8K of ROM ($E000 through $FFFF), and the Apple II Plus has 12K of ROM ($D000 through $FFFF). Users can plug their own ROMs into the sockets provided. The on-board (as opposed to slot) ROM address range is from $D000 through $FFFF.

The Apple IIe has 16K of ROM, of which it uses 15.75K (addresses $C100 through $FFFF; page $C0 addresses are for I/O hardware). ROM addresses $C300 through $C3FF (normally assigned to the ROM in a card in slot 3) and $C800 through $CFFF contain 80-column video firmware; ROM addresses $C100 through $C2FF and $C400 through $C7FF (normally assigned to the ROM on cards in slots 1, 2, 4, 5, 6, and 7) contain built-in self-test routines.

A soft switch in RAM controls whether the video firmware or slot 3 card ROM is active. Invoking the self-tests with Solid Apple-Control-Reset causes the self-test firmware to take over the slot ROM address spaces.

The Apple IIc ROM also uses the 15.75K from $C100 through $FFFF, and its enhanced video firmware has the same entry point addresses as on the Apple IIe. However, there are only rudimentary built-in self-tests, and these do not preempt any port firmware space.

UniDisk 3.5

The Apple IIc with built-in UniDisk 3.5 support has twice the ROM (32K) of the original Apple IIc. The extra ROM contains support for the Smartport, a Mini-Assembler, STEP and TRACE functions in the Monitor firmware, expanded self-test routines, and improved interrupt support.

In the Apple IIc, addresses $C100 through $CFFF contain I/O and interrupt firmware, addresses $D000 through $F7FF contain the Applesoft BASIC interpreter, and addresses $F800 through $FFFF contain the Monitor.

Peripheral-card memory spaces

Each Apple IIc port has up to 16 peripheral-card I/O space locations in main memory on the hardware page (beginning at location $C0s0 + $80 for slot or port s), allocated in the standard Apple II series way (that is, beginning at location $C0s0 + $80 for each slot s).

The peripheral-card ROM space (page $Cs for slot s in the Apple II, II Plus, and IIe) contains the starting and entry-point addresses for port s, but port routines are not limited to their allocated $Cs pages.

The 2K-byte expansion ROM space from $C800 to $CFFF in the Apple IIc is used by the enhanced video firmware and miscellaneous I/O and memory-transfer routines.

The 128 bytes of peripheral-card RAM space (or scratch-pad RAM) (64 screen holes in main memory and their equivalent addresses in auxiliary memory) are reserved for use by the built-in firmware. It is extremely important for the correct operation of Apple IIc firmware that these locations not be altered by software except for the specific purposes described in Chapters 7, 8, and 9, and in Appendix E.

Hardware addresses

The hardware page (the addresses from $C000 through $C0FF) controls memory selection and input/output hardware characteristics. All input and output (except video output) takes place at one or more hardware page addresses. For the sake of simplicity, this section presents only a general comparison between the Apple IIc on the one hand, and the Apple II, II Plus, and IIe on the other, with respect to hardware page use. However, for many characteristics, the Apple IIe and IIc work one way, while the Apple II and II Plus work another.

$C000–$C00F

On all Apple II series computers, reading any one of these addresses reads the keyboard data and strobe. On the Apple IIe and IIc, writing to each of these addresses turns memory and display switches on and off. Writing to addresses $C006, $C007, $C00A, and $C00B performs ROM selection on the Apple IIe. Writing to these four addresses is reserved on the Apple IIc.

For reading the keyboard, use $C000; reserve $C001 through $C00F.

$C010–$C01F

On all Apple II series computers, writing to any one of these addresses clears the keyboard strobe. On the Apple IIe and IIc, reading each of these addresses checks the status of a memory or display switch, or the any-key-down flag.

For clearing the keyboard strobe, use $C010; reserve $C011 through $C01F.

Reading $C015 checks the SLOTCXROM switch on the Apple IIe, but it resets the X-movement interrupt (XInt) on the Apple IIc. Similarly, reading $C017 checks the SLOTC3ROM switch on the Apple IIe, but it resets the Y-movement interrupt (YInt) on the Apple IIc.

Reading $C019 checks the current state of vertical blanking (VBL) on the Apple IIe, but it resets the latched vertical blanking interrupt (VBlInt) on the Apple IIc.

$C020–$C02F

On the Apple II, II Plus, and IIe, reading any address $C02x toggles the cassette output signal. On the original Apple IIc, both reading from and writing to these locations are reserved. The Apple IIc with 32K of ROM uses $C028 to switch in or out the extra 16K of ROM.

$C030–$C03F

On all Apple II series computers, reading an address of the form $C03x toggles the speaker. For full Apple II series compatibility, toggle the speaker using $C030, and reserve $C031 through $C03F.

On the Apple IIc, writing to $C031 through $C03F is explicitly reserved.

$C040–$C04F

On the Apple II, II Plus, and IIe, reading any address of the form $C04x triggers the utility strobe. The Apple IIc has no utility strobe.

On the Apple IIc, addresses $C044 through $C047 are explicitly reserved, and reading or writing any address from $C048 through $C04F resets both the X and Y mouse interrupts (XInt and YInt).

$C050–$C05F

Addresses $C050 through $C057 work the same on the Apple IIc as on the Apple IIe: they turn the TEXT, MIXED, Page2, and HiRes switches on and off.

On the Apple IIe, addresses $C058 through $C05F turn the annunciator outputs on and off. On an Apple IIe with a revision B main logic board or later, an Apple Extended 80-Column Text Card, and a jumper installed on the card, reading locations $C05E and $C05F set and clear double high-resolution display mode.

On the Apple IIc, if the IOUDis switch is on, both reading from and writing to addresses $C058 through $C05D are reserved, and addresses $C05E and $C05F set and clear the double high-resolution display (as on the Apple IIe equipped as described in the preceding paragraph). If the IOUDis switch is off, then addresses $C058 through $C05F control various characteristics of mouse and vertical blanking interrupts (Table 9-2).

$C060–$C06F

On the Apple IIc, writing to any address of the form $C06x is reserved, and reading addresses $C068 through $C06F is reserved.

Reading addresses $C061 and $C062 is the same as on the Apple IIe (switch inputs and Apple keys). Reading addresses $C064 and $C065 is the same as on all other Apple II series computers (analog inputs 0 and 1).

On the Apple IIc, address $C063 bit 7 is 1 if the mouse switch is not pressed, and 0 if it is pressed, so that software looking for the shift-key mod (used on Apple II, II Plus, and IIe with some text cards) will find it and display lowercase correctly. If by chance the mouse button is pressed when the software checks location $C063, it will appear that the Shift-key mod is not present.

On the Apple IIc, address $C060 is used for reading the state of the 80/40 switch; on the Apple II, II Plus, and IIe, this address is for reading cassette input.

The Apple IIc has two, rather than four, analog (paddle) inputs. Addresses $C066 and $C067 are used for reading the mouse X and Y direction bits.

$C070–$C07F

On the Apple II, II Plus, and IIe, reading from or writing to any address of the form $C07x triggers the (analog input) paddle timers.

On the Apple IIc, only address $C070 is to be used for that one function. Addresses $C071 through $C07D are explicitly reserved. The results of reading from or writing to addresses $C07E and $C07F are described in Table 5-8.

$C080–$C08F

On the Apple IIe and IIc, accessing addresses in this range selects different combinations of bank-switched memory banks. However, addresses $C084 through $C087 duplicate the functions of the four addresses preceding them, and addresses $C08C through $C08F do also. These eight addresses are explicitly reserved on the Apple IIc.

$C090–$C0FF

On the Apple II, II Plus, and IIe, each group of 16 addresses of the form $C080 + $s0 is allocated to an interface card (if present) in slot s.

On the Apple IIc, addresses corresponding to slots 1, 2, 3, 4, and 6 are allocated to a serial interface card, communication interface card, 80-column text card, mouse interface card, and disk controller card, respectively. All other addresses in this range are reserved.

Monitors

The older models of the Apple II and Apple II Plus included a different version of the System Monitor from the one built into more recent models (and the Apple IIe and IIc). The older version, called the Monitor ROM, had the same standard I/O subroutines as the newer Autostart ROM, but a few of their features were different; for example, there were no arrow keys for vertical cursor motion.

When you start the Apple IIc with a DOS or *BASICS* disk and it loads Integer BASIC into the bank-switched area in RAM, it loads the old Monitor along 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 activate the 80-column firmware.

I/O in general

Apple IIc I/O is different from I/O on the Apple II, II Plus, and IIe in three important respects: the possibility of direct memory access (DMA) transfers, the presence or absence of slots, and the presence or absence of built-in interrupt handling.

DMA transfers

The Apple II, II Plus, and IIe allow DMA transfers, because both the address and the data bus are available at the slots. No true DMA transfer is possible with the Apple IIc because neither bus is available at any of the back panel connectors.

Slots versus ports

The Apple II and II Plus have eight identical slots; the Apple IIe has seven identical slots plus a 60-pin auxiliary slot for video, add-on memory, and test cards. The Apple IIc has no slots; instead, it has back panel connectors and built-in hardware and firmware that are functional equivalents of slots with cards in them. The back panel connectors are called *ports* on the Apple IIc.

Interrupts

The Apple IIc is the first computer in the Apple II series to have built-in interrupt-handling capabilities. The enhanced Apple IIe has very similar interrupt-handling capability included.

The keyboard

Both keyboard layout and character sets vary in the Apple II series computers. The major keyboard difference in the Apple II series is that the Apple IIe and IIc have full ASCII keyboards, while the Apple II and II Plus do not.

Keys, switches, and lights

The Apple II and II Plus have identical 52-key keyboards. The Apple IIe and Apple IIc keyboards have the same 63-key full ASCII keyboard layout, with new and repositioned keys and characters as compared to the Apple II and II Plus. While the Apple II and II Plus have a Rept key, the IIe and IIc have an auto-repeat feature built into each character key.

Some Apple II and Apple II Plus machines have a slide switch inside the case, under the keyboard edge of the cover, for selecting whether or not Reset works without Control. On the Apple IIe and Apple IIc, there is no choice: Control-Reset works, and Reset alone does not.

The Apple IIc and IIe have an Open Apple and a Solid Apple key; the Apple II and II Plus do not have these two keys.

The captions on several keys—Escape, Tab, Control, Shift, Caps Lock, Delete, Return, and Reset—can vary: on the Apple II and II Plus some are abbreviated or missing; on the Apple IIc all keycaps are lowercase italic; on international models, some captions are replaced by symbols (Appendix G).

The Apple IIc has two switches that the other models do not have. One switch is for changing between 40-column and 80-column display, the other is for selecting keyboard layout (Sholes versus Dvorak on USA models), or both keyboard layout and character set (on international models).

The position of the power-on light differs on the Apple II and II Plus, Apple IIe, and Apple IIc. The Apple IIc has a disk-use light as well.

Character sets

The Apple II and II Plus keyboard character sets are the same. They are described in the *Apple II Reference Manual.*

The Apple IIe and Apple IIc keyboard character sets are the same: full ASCII. The standard (Sholes) layout and key assignments are described in the *Apple IIe Reference Manual.* The Dvorak layout and key assignments are described in Chapter 4 and Appendix G of this manual.

To change between the two available keyboard layouts requires modification to the main logic board on the Apple IIe, but only toggling of the keyboard switch on the Apple IIc.

Apple Computer, Inc., manufactures fully localized models (with regard to power supply and character sets) of both the Apple IIe and the Apple IIc. However, there are minor variations in keyboard layout, even among early and late production models of the same machine. For further details, refer to Appendix G of this manual or to the Apple IIe *Supplement to the Owner's Manual.*

The speaker

The Apple IIc has two speaker features that the three previous models do not have. They are a two-channel, but monaural, audio output jack for headphones—which disconnects the internal speaker when something is plugged into it—and a volume control.

The video display

This section discusses the general differences between Apple IIc video display capabilities and those of the other computers in the series. Note, however, that as new ROMs become available for the Apple IIe, many differences between these two machines will vanish.

Character sets

The Apple II and II Plus display only uppercase characters, but they display them in three ways: normal, inverse, and flashing. The Apple IIc and IIe can display uppercase characters in all three ways, and they can display lowercase characters in the normal way. This combination is called the *primary character set.*

The Apple IIc and IIe have another character set, called the *alternate character set,* that displays a full set of normal and inverse uppercase and lowercase characters, but can't display flashing characters. The primary and alternate character sets are described in Chapter 5. You can switch character sets at any time by means of the AltChar soft switch, also described in Chapter 5.

Flashing display must not be used with the enhanced video firmware active. Use it in 40-column mode with the enhanced video firmware turned off; otherwise, strange displays may result, such as MouseText characters appearing in place of uppercase letters.

To be sure of compatibility with some software, you have to switch the Apple IIc keyboard to uppercase by pressing Caps Lock.

MouseText

MouseText characters (Chapter 5) are available on every Apple IIc, and on the enhanced Apple IIe.

Vertical blanking

A signal called *vertical blanking* indicates when a display device should stop projecting dots until the display mechanism returns from the bottom of the screen to the top to make another pass. During this interval, a program can make changes to display memory pages, and thus provide a smooth, flicker-free transition to a new display.

On the Apple IIe, vertical blanking (VBL) is a signal whose level must be polled. (VBL is not available to software on the Apple II or II Plus.) On the Apple IIc, vertical blanking is an interrupt (VBlInt) that occurs on the trailing edge of the active-low VBL signal. Programs intended to run on all Apple II series computers must take this difference into account.

Display modes

All models have 40-column text mode, low-resolution graphics mode, high-resolution graphics mode, and mixed graphics and text modes. The Apple IIe (revision B motherboard) with an Apple Extended 80-Column Text Card, and the Apple IIc have double high-resolution graphics mode also.

Disk I/O

The Apple II, II Plus, and IIe can support up to six disk drives (although four is the recommended maximum) attached in controller cards plugged into slots 6, 5, and 4. The Apple IIc supports up to two disk drives: its built-in drive (treated as slot 6, drive 1), and one external disk drive (treated as slot 6, drive 2; also treated as slot 7, drive 1 under ProDOS) for external-drive startup purposes.

UniDisk 3.5 The Apple IIc with UniDisk 3.5 support does not use slot 7, drive 1 for external drives. They are handled through the Smartport described in Chapter 6. The firmware for slot 7 ($C7xx) is needed for other parts of the firmware.

Serial I/O

The Apple IIc serial ports (ports 1 and 2) are similar to Super Serial Cards installed in slots 1 and 2 of an Apple IIe. The serial port commands are a slightly modified subset of Super Serial Card commands. This subset includes all the commands supported by the earlier Apple Serial Interface Card and Communication Card.

Serial ports versus serial cards

There are several important differences between Apple IIc serial ports and other Apple II series computers with serial cards installed in them.

Apple IIc serial ports have no switches. Instead, initial values are moved from firmware locations into auxiliary memory when the power is turned on. Changes made to these values in auxiliary memory remain in effect until the power is turned off. Pressing Open Apple-Control-Reset does not change them.

When the port itself is turned on (with an IN or PR command), the initial values in auxiliary memory are placed in the main memory screen holes assigned to the port. These characteristics can be changed by the port commands. The changed characteristics remain in effect until the port is turned off and then on again (with PR and IN commands).

The command syntax for the Apple IIc ports also differs from the syntax for serial cards. A separate command character, Control-A or Control-I, must precede each individual port command, whereas several commands to a serial card can be strung together between the command character and a carriage return character.

The letters used for some of the commands have been changed from those used with the Super Serial Card (such as S instead of B for sending a BREAK signal). Each serial port command letter is unique, to simplify command interpretation.

Changing the command character from Control-A to Control-I, or vice versa, makes the Super Serial Card change from communication mode to printer mode and back; this is not the case with Apple IIc serial ports. With the Apple IIc, use the *System Utilities* disk to change modes.

Super Serial Card commands support some functions that Apple IIc serial port commands don't support: translating incoming characters, such as changing lowercase to uppercase (for the benefit of the Apple II or II Plus); delaying after sending carriage return, line feed, or form feed, and so on.

UniDisk 3.5 Several new serial port commands are available on the Apple IIc with UniDisk 3.5 support. These commands have been added to make it easier to write programs that are also compatible with the Super Serial Card. See Chapters 7 and 8 for these new commands.

Following a Control-I nnnN command, the Apple IIc automatically generates a carriage return after nnnN characters; with the Super Serial Card, you need to turn this on with Control-I C.

Serial I/O buffers

The communication port firmware uses auxiliary memory page $08 as an input and output buffer. By doing so, the firmware can keep up with higher baud rates. It can also hide data from the Monitor, Applesoft, and other system software.

Programs written for the Apple IIe or IIc can, of course, store information in auxiliary memory page $08. However, such information is destroyed when the communication port is activated.

Mouse and hand controllers

The DB-9 back panel connector on the Apple IIc is used for both the mouse and hand controllers. On the Apple IIe, the DB-9 connector supports hand controllers only; the mouse must use the connector on the interface card.

Mouse input

The Apple IIc provides built-in firmware support for a mouse connected to the DB-9 mouse and hand controller connector. Apple IIc mouse support includes mouse movement and button interrupts (and vertical blanking interrupts for synchronization with the display); Apple IIe mouse support relies on polling VBL instead of vertical blanking interrupts.

As a result of how interrupts are handled on the two machines, the mouse firmware routine calls function somewhat differently for the Apple IIc and Apple IIe. However, using the calls in the manner described in Chapter 9 ensures mouse support compatibility between the two machines. The ratio of mouse movement to cursor movement is different on the Apple IIc from on the Apple IIe.

Hand controller input and output

The Apple II, II Plus, and IIe have a 16-pin game I/O connector inside the case that supports three switch inputs, four analog (paddle) inputs, and four annunciator outputs. The Apple IIe and Apple IIc have a DB-9 back panel connector that supports the three switch inputs and two paddle inputs (plus two more on the internal GAME I/O connector of the Apple II, II Plus, and IIe).

The Apple IIc does not support the four annunciator outputs.

The characteristic response curve for hand controllers differs for the Apple IIc from that of the Apple II, II Plus, and IIe. Compare Figure F-1 with Figure 11-42. This was done so the hardware would support identifiable mouse and hand controller signals using the same circuits.

The paddle-timing circuit on the Apple II Plus is slightly different from the one on the Apple IIe and IIc. On the Apple IIe and IIc the 100-ohm fixed resistor is between the NE556 discharge lead and the capacitor; the variable resistor in the paddle is connected directly to the capacitor. On the Apple II Plus, the capacitor is connected directly to the discharge lead, and the fixed resistor is in series with the paddle resistor.

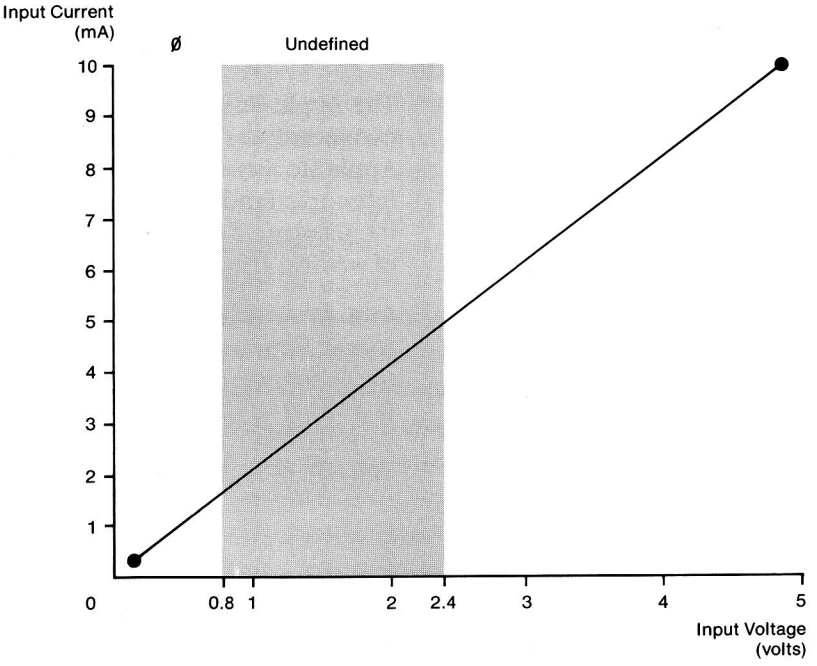


Figure F-1
Apple II, II Plus, and IIe hand controller signals

Cassette I/O

The Apple II, II Plus, and IIe all have cassette input and output jacks, memory locations, and Monitor support. The Apple IIc does not.

UniDisk 3.5 If you plan to run a program on your Apple IIc that handles cassette I/O, make sure that it does not access $C028. The Apple IIc with UniDisk 3.5 support uses address $C028 to toggle between its two 16K banks of memory.

Hardware

Besides the different microprocessors used in various models in the Apple II series, there are important differences in power specifications and custom chips.

Power

The power supplies for the Apple II, II Plus, and IIe are essentially the same. The floor transformer and voltage converter for the Apple IIc have smaller capacity for current and heat dissipation. Therefore, it is important to observe the load limits specified in each of the reference manuals.

Custom chips

The Apple IIe custom chips (memory management unit and input/output unit) replaced dozens of Apple II Plus chips, and added the functionality of dozens more. The Apple IIc has custom MMU and IOU chips, too, but they represent different bonding options, and so their pin assignments are not compatible.

In addition, the Apple IIc has a custom general logic unit (GLU), timing generator (TMG), and disk controller unit (also known as an Integrated Woz Machine, or IWM). The Apple IIc has two hybrid units (AUD and VID) for audio and video amplification.

Appendix G

USA and International Models

This appendix repeats some of the keyboard information given in Chapter 4 for the two USA keyboard layouts, for easy comparison with the other layouts available. Following these is a composite table of the ASCII codes and the characters associated with them on all the models discussed.

Keyboard layouts and codes

Each of the following subsections has a keyboard illustration and a table of the codes that result from the possible keystrokes. Note, however, that Table G-1 is the basic table of keystrokes and their codes. For simplicity, subsequent tables (up to Table G-7) list only the keystrokes and codes that differ from those in Table G-1.

For example, pressing the A key produces *a* (hexadecimal 61); pressing Shift-A produces uppercase *A* (hexadecimal 41); pressing Control-A or Control-Shift-A produces *SOH* (the ASCII Start Of Header control character, hexadecimal 01). You can tell that this key has the same effect on all keyboards because nothing appears in Tables G-2 through G-7 for that key.

A quick way to find out which characters in the ASCII set change on international keyboards is to check Table G-8. In fact, only a few of them change. The pairing of characters on keys varies more.

❖ *Note:* On all but the French and Italian keyboards, Caps Lock affects only keys that can produce both lowercase letters (with or without an accent) and their uppercase equivalents. With these keys, Caps Lock down is equivalent to holding down Shift, resulting in uppercase instead of lowercase. If a key produces only a lowercase version of an accented letter, then Caps Lock does not affect it.

On the French and Italian keyboards, Caps Lock shifts all the keys. Furthermore, on the French keyboard, when Caps Lock is down the Shift key undoes the shifting.

The shapes and arrangement of keys in Figures G-1 and G-2 follow the ANSI (American National Standards Institute) standard, which is used mainly in North and South America. The shapes and arrangement of keys in Figure G-3 follow the ISO (International Standards Organization) standard used in Europe and elsewhere.

The only differences between the ANSI and ISO versions of the USA keyboard are

- the shapes of three keys: the left Shift key, Caps Lock, and Return
- the resulting repositioning of two keys (| and ~) in Figures G-1 and G-3
- for some countries, the arrow symbols on Tab, Caps Lock, Return, and the two Shift keys (as shown in Figure G-3)

USA standard (Sholes) keyboard

Figure G-1 shows the standard (Sholes) keyboard as it is laid out for USA models of the Apple IIc with the keyboard switch up. Table G-1 lists the ASCII codes resulting from all simple and combination keystrokes on this keyboard.

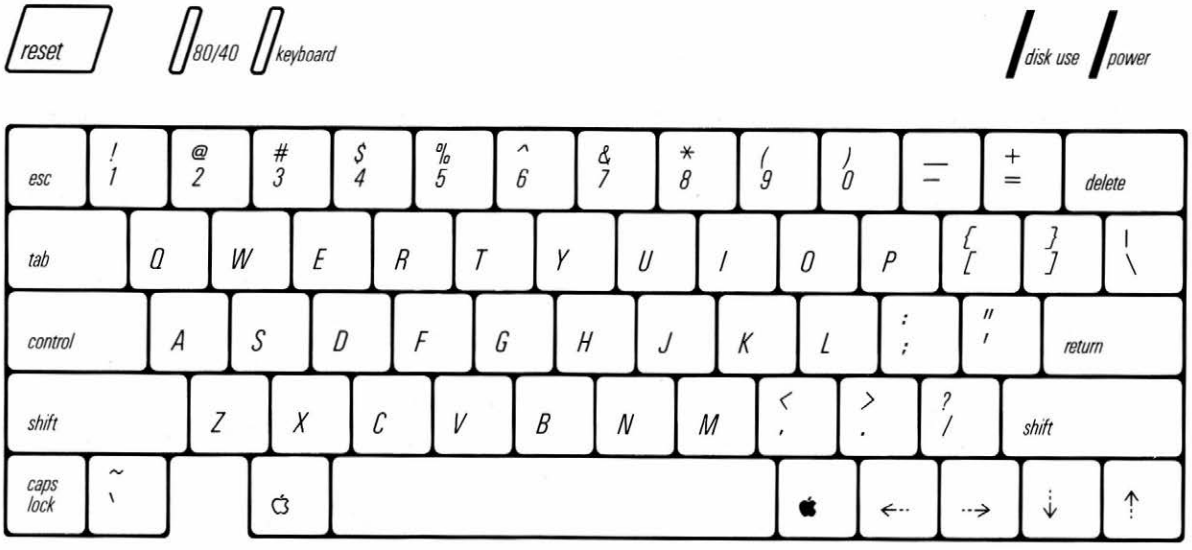


Figure G-1
USA standard (or Sholes) keyboard, keyboard switch up

Table G-1
Keys and ASCII codes

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
Delete	7F	DEL	7F	DEL	7F	DEL	7F	DEL
Left Arrow	08	BS	08	BS	08	BS	08	BS
Tab	09	HT	09	HT	09	HT	09	HT
Down Arrow	0A	LF	0A	LF	0A	LF	0A	LF
Up Arrow	0B	VT	0B	VT	0B	VT	0B	VT
Return	0D	CR	0D	CR	0D	CR	0D	CR
Right Arrow	15	NAK	15	NAK	15	NAK	15	NAK
Escape	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	NUI
3 #	33	3	33	3	23	#	23	#

Table G-1 (continued)
Keys and ASCII codes

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
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	~
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

Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

USA simplified (Dvorak) keyboard

Figure G-2 shows the Dvorak layout of the USA keyboard. Characters are paired up on keys in exactly the same way as on the USA standard keyboard; only individual key positions are changed. In fact, you can change the keycap arrangement to match Figure G-2, lock the keyboard switch in its down position, and have a working Dvorak keyboard. All keystrokes produce the same ASCII codes as those shown in Table G-1.

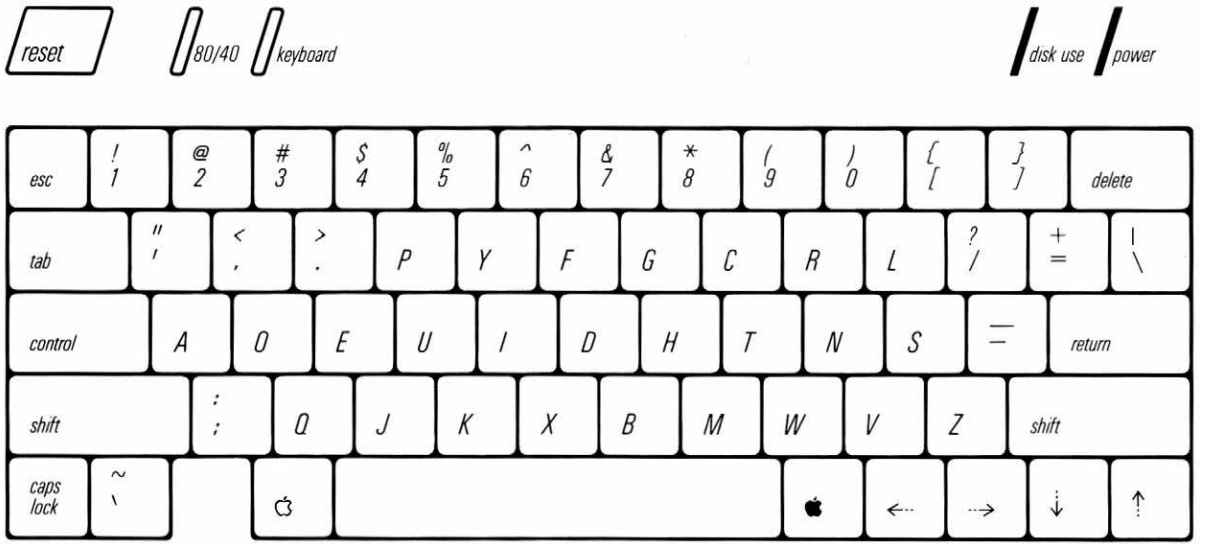


Figure G-2
USA simplified (or Dvorak) keyboard, keyboard switch down

ISO layout of USA keyboard

Figure G-3 shows the layout of all ISO European keyboards (except the Italian keyboard) when the keyboard switch is up. All keystrokes produce the same ASCII codes as those shown in Table G-1.

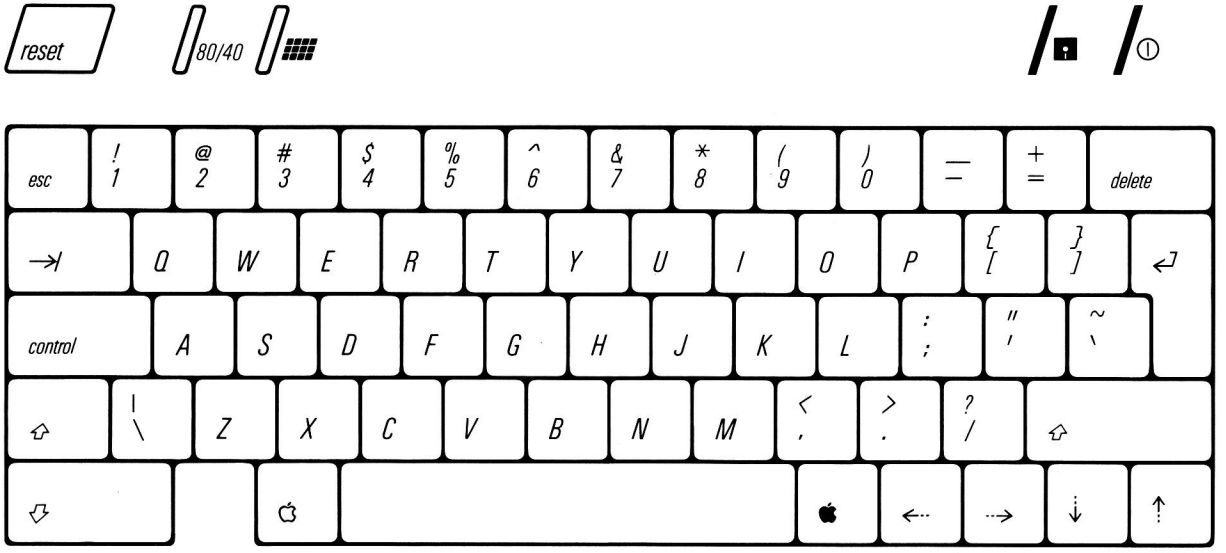


Figure G-3
ISO version of USA standard keyboard, keyboard switch up

English keyboard

With the keyboard switch up, the English model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the English model keyboard layout is as shown in Figure G-4. The change in ASCII code production (from that in Table G-1) is shown in Table G-2.

The only changed character is the substitution of the British pound-sterling symbol (£) for the cross-hatch symbol (#) on the shifted 3-key.

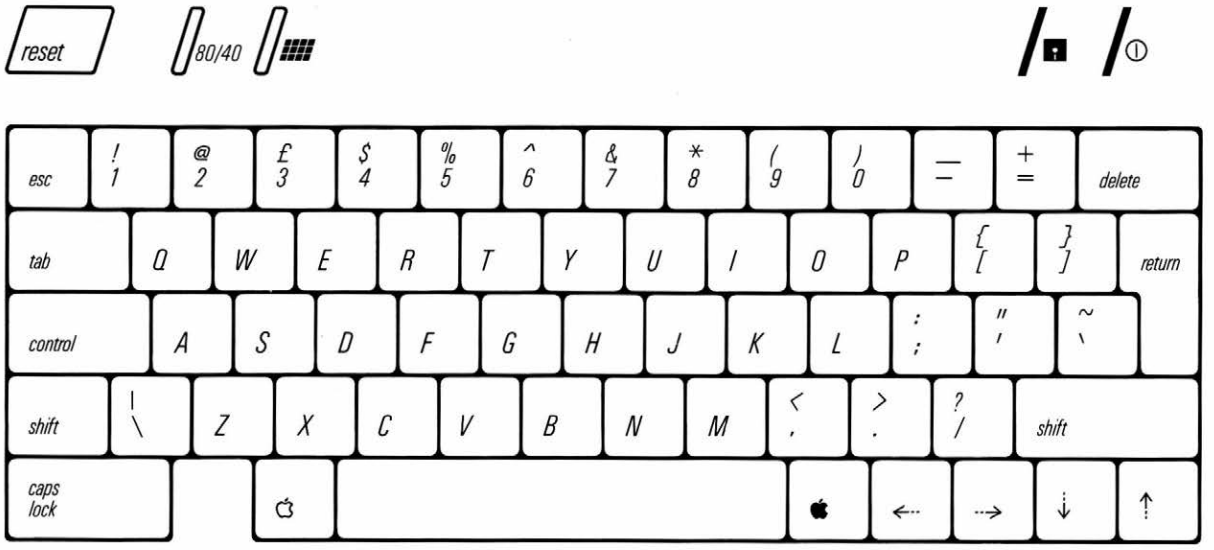


Figure G-4
English keyboard, keyboard switch down

Table G-2
English keyboard code differences from Table G-1

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
3 £	33	3	33	3	23	£	23	£

Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

French keyboard

With the keyboard switch up, the French model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the French model keyboard layout is as shown in Figure G-5. The changes in ASCII code production (from that in Table G-1) are shown in Table G-3.

Note that on the French keyboard, Caps Lock shifts to the upper characters on all keys. With Caps Lock on, Shift "unshifts" to the lower character on any key pressed with it.

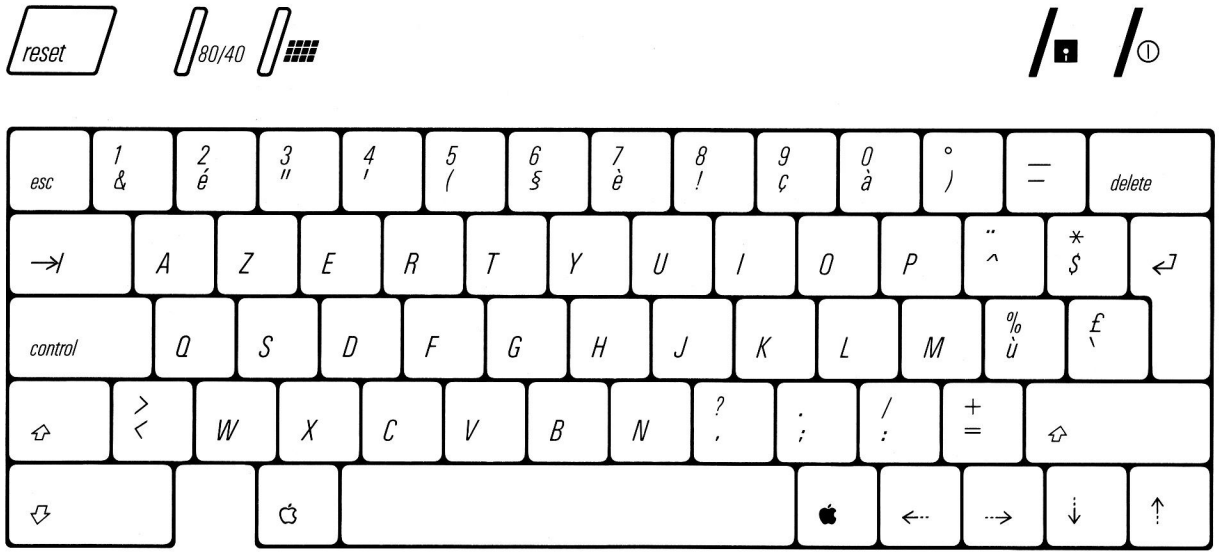


Figure G-5
French keyboard, keyboard switch down

Table G-3
French keyboard code differences from Table G-1

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
& 1	26	&	26	&	31	1	31	1
é 2	7B	é	7B	é	32	2	32	2
" 3	22	"	22	"	33	3	33	3
' 4	27	'	27	'	34	4	34	4
(5	28	(	28	(	35	5	35	5
§ 6	5D	§	1D	GS	36	6	1D	GS
è 7	7D	è	7D	è	37	7	37	7
! 8	21	!	21	!	38	8	38	8
ç 9	5C	ç	1C	FS	39	9	1C	FS
à 0	40	à	00	NUL	30	0	00	NUL
) °	29	)	1B	ESC	5B	°	1B	ESC
^ ¨	5E	^	1E	RS	7E	¨	1E	RS
$ *	24	$	24	$	2A	*	2A	*
ù %	7C	ù	7C	ù	25	%	25	%
` £	60	`	60	`	23	£	23	£
< >	3C	<	3C		3E	>	3E	>
, ?	2C	,	2C	,	3F	?	3F	?
; .	3B	;	3B	;	2E	.	2E	.
: /	3A	:	3A	:	2F	/	2F	/

Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Canadian keyboard

With the keyboard switch up, the Canadian model of the Apple IIc keyboard layout is as shown in Figure G-1, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the Canadian model keyboard layout is as shown in Figure G-6. The changes in ASCII code production (from that in Table G-1) are shown in Table G-4.

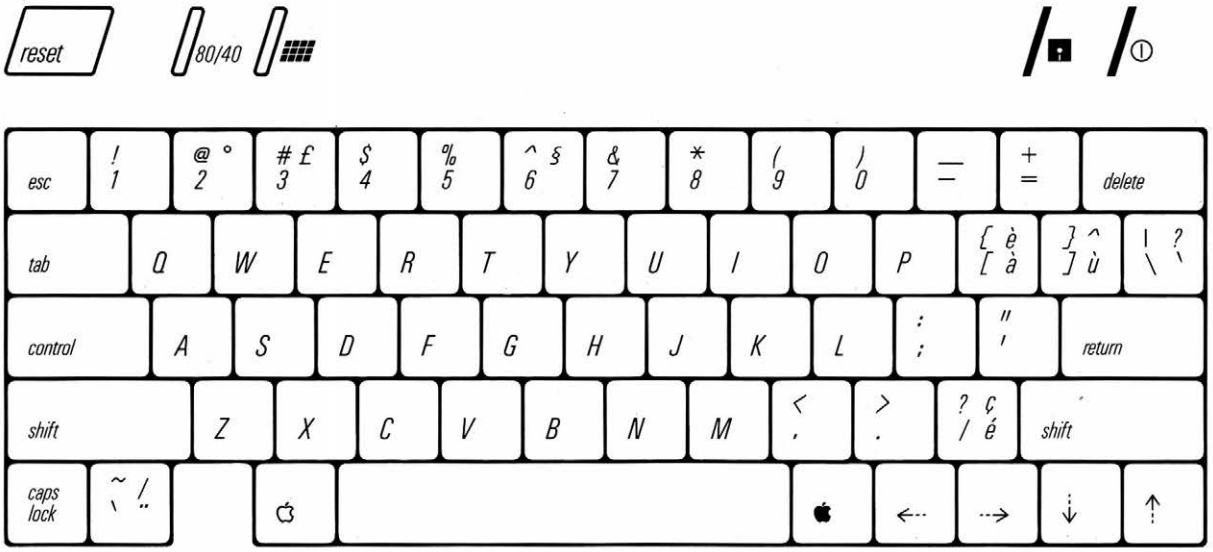


Figure G-6
Canadian keyboard, keyboard switch down

Table G-4
Canadian keyboard code differences from Table G-1

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
2 °	32	2	00	NUL	5B	°	00	NUL
3 £	33	3	33	3	23	£	23	£
6 §	36	6	RS	1E	5D	§	RS	1E
à è	40	à	7F	DEL	7D	è	7F	DEL
Ù ^	7C	Ù	7C	Ù	5E	^	5E	^
` ?	60	`	ESC	1B	3F	?	1D	GS
é ç	7B	é	1C	FS	5C	ç	1C	FS
" /	7E	"	7E	"	2F	/	2F	/

Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

German keyboard

With the keyboard switch up, the German model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the German model keyboard layout is as shown in Figure G-7. The change in ASCII code production (from that in Table G-1) is shown in Table G-5.

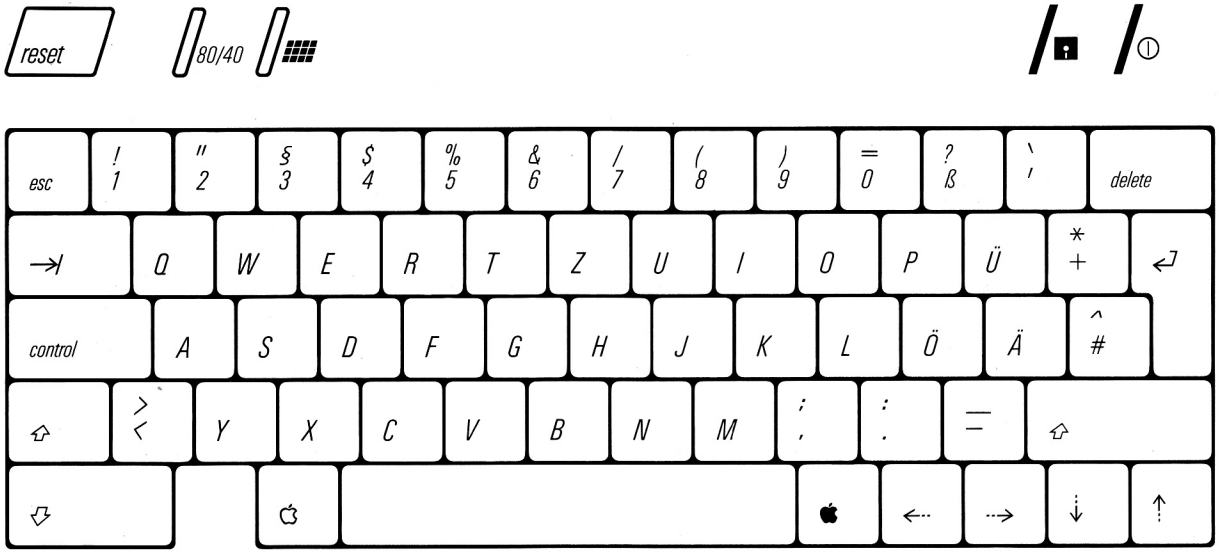


Figure G-7
German keyboard, keyboard switch down

Table G-5
German keyboard code differences from Table G-1

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
2 "	32	2	32	2	22	"	22	"
3 §	33	3	00	NUL	40	§	00	NUL
6 &	36	6	36	6	26	&	26	&
7 /	37	7	37	7	2F	/	2F	/
8 (	38	8	38	8	28	(	28	(
9)	39	9	39	9	29	)	29	)
0 =	30	0	30	0	3D	=	3D	=
ß ?	7E	ß	7E	ß	3F	?	3F	?
Ü	7D	Ü	1D	GS	5D	Ü	1D	GS
+ *	2B	+	2B	+	2A	*	2A	*
Ö	7C	Ö	1C	FS	5C	Ö	1C	FS
Ä	7B	Ä	1B	ESC	5B	Ä	1B	ESC
# ^	23	#	1E	RS	5E	^	1E	RS
< >	3C	<	3C	<	3E	>	3E	>
, ;	2C	,	2C	,	3B	;	3B	;
. :	2E	.	2E	.	3A	:	3A	:

Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Italian keyboard

With the keyboard switch down, the Italian model keyboard layout is as shown in Figure G-8. The change in ASCII code production (from that in Table G-1) is shown in Table G-6.

With the keyboard switch up, the Italian model keyboard produces exactly the same ASCII codes for each key, but what is displayed differs for the ten characters indicated with the circled numbers 0, 2–5, and 7–11 in Table G-8.

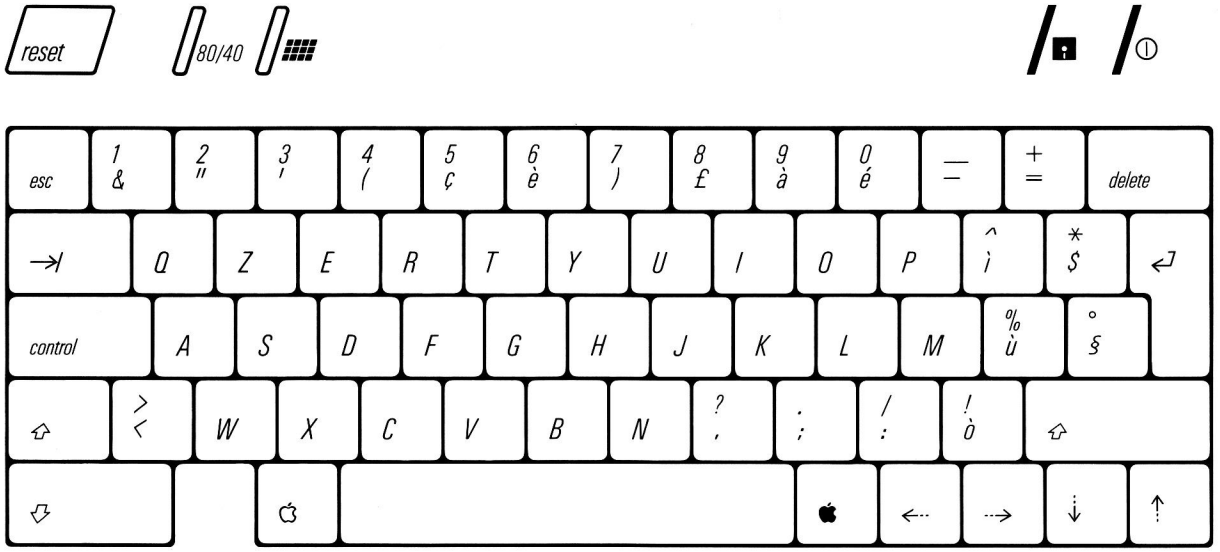


Figure G-8
Italian keyboard, keyboard switch down

Table G-6
Italian keyboard code differences from Table G-1

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
& 1	26	&	26	&	31	1	31	1
" 2	22	"	22	"	32	2	32	2
' 3	27	'	27	'	33	3	33	3
(4	28	(	28	(	34	4	34	4
ç 5	5C	ç	1C	FS	35	5	1C	FS
è 6	7D	è	7D	è	36	6	36	6
) 7	29	)	29	)	37	7	37	7
£ 8	23	£	23	£	38	8	38	8
à 9	7B	à	7B	à	39	9	39	9
é 0	5D	é	1D	GS	30	0	1D	GS
Ì ^	7E	Ì	1E	RS	5E	^	1E	RS
$ *	24	$	24	$	2A	*	2A	*
ù %	60	ù	60	ù	25	%	25	%
§°	40	§	00	NUL	5B	°	1B	ESC
< >	3C	<	3C	<	3E	>	3E	>
, ?	2C	,	2C	,	3F	?	3F	?
; .	3B	;	3B	;	2E	.	2E	.
: /	3A	:	3A	:	2F	/	2F	/
ò !	7C	ò	7C	ò	21	!	21	!

Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

Western Spanish keyboard

With the keyboard switch up, the Western (that is, American) Spanish model of the Apple IIc keyboard layout is as shown in Figure G-1, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the Western Spanish model keyboard layout is as shown in Figure G-9. The change in ASCII code production (from that in Table G-1) is shown in Table G-7.

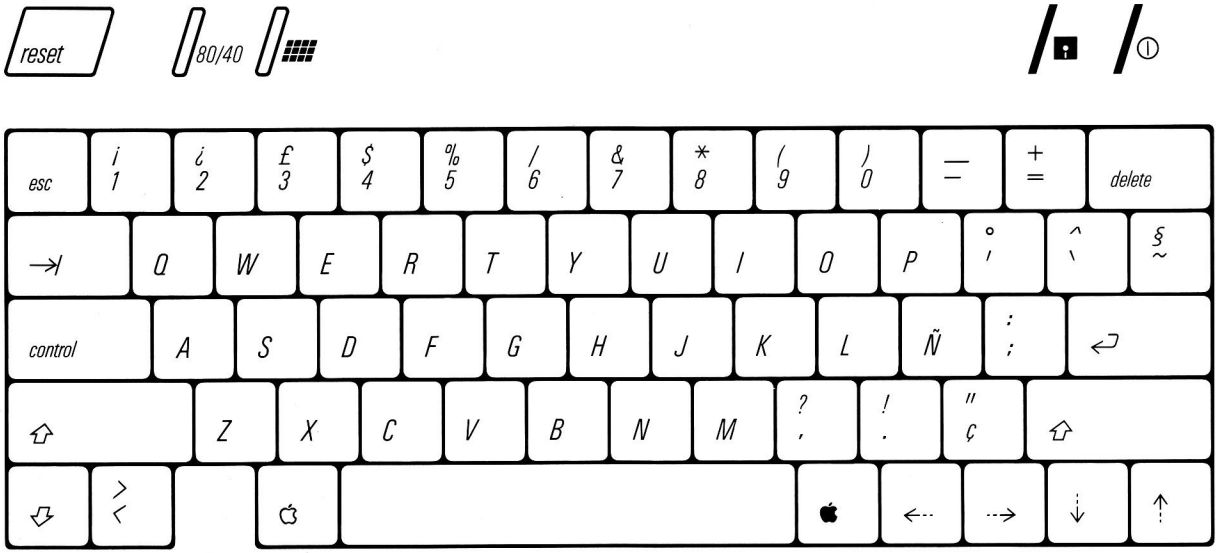


Figure G-9
Western Spanish keyboard, keyboard switch down

Table G-7
Western Spanish keyboard code differences from Table G-1

	Key alone		+ Control		+ Shift		+ Both	
Key	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**	**Code**	**Char**
1 ¡	31	1	31	1	5B	¡	5B	¡
2 ¿	32	2	32	2	5D	¿	5D	¿
3 £	33	3	33	3	23	£	23	
6 /	36	6	36	6	2F	/	2F	/
´ °	27	´	27	´	7B	°	7B	°
`^	60	`	00	NUL	5E	^	00	NUL
~ §	7E	~	7F	DEL	40	§	7F	DEL
Ñ	7C	Ñ	1C	FS	5C	Ñ	1C	FS
, ?	2C	,	2C	,	3F	?	3F	?
. !	2E	.	2E	.	21	!	21	!
ç"	7D	ç	1D	GS	22	"	1D	GS
< >	3C	<	1E	RS	3E	>	1E	RS

Note: Codes are in hexadecimal here; refer to Table G-8 for decimal equivalents.

ASCII character sets

Table G-8 lists the ASCII (American National Standard Code for Information Interchange) codes that the Apple IIc uses, as well as the decimal and hexadecimal equivalents. Where there are differences between character sets, an asterisked number in the main table refers to a column in the following part of the table.

Table G-8
ASCII code equivalents

ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex
NUL	00	00	SP	32	20	2*	64	40	7*	96	60
SOH	01	01	!	33	21	A	65	41	a	97	61
STX	02	02	"	34	22	B	66	42	b	98	62
ETX	03	03	0*	35	23	C	67	43	c	99	63
EOT	04	04	1*	36	24	D	68	44	d	100	64
ENQ	05	05	%	37	25	E	69	45	e	101	65
ACK	06	06	&	38	26	F	70	46	f	102	66
BEL	07	07	'	39	27	G	71	47	g	103	67

Table G-8 (continued)
ASCII code equivalents

ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex
BS	08	08	(	40	28	H	72	48	h	104	68
HT	09	09	)	41	29	I	73	49	i	105	69
LF	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	11	0B	+	43	2B	K	75	4	k	107	6B
FF	12	0C	,	44	2C	L	76	4C	l	108	6C
CR	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	8	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	27	1B	;	59	3B	3*	91	5B	8*	123	7B
FS	28	1C	<	60	3C	4*	92	5C	9*	124	7C
GS	29	1D	=	61	3D	5*	93	5D	10*	125	7D
RS	30	1E	>	62	3E	6*	94	5E	11*	126	7E
US	31	1F	?	63	3F	_	95	5F	DEL	127	7F

The following characters correspond to those followed by an asterisk in the preceding part of the table.

*	0	1	2	3	4	5	6	7	8	9	10	11
Hexadecimal	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
English (USA)	#	$	@	[	\	]	^	`	{	|	}	~
English (UK)	£	$	@	[	\	]	^	`	{	|	}	~
German	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
French	£	$	à	°	ç	§	^	`	é	Ù	è	¨
Italian	£	$	§	°	ç	é	^	Ù	à	ò	è	Ì
Spanish	£	$	§	¡	Ñ	¿	^	`	°	ñ	ç	~

Certification

In the countries where it is applicable, the following product safety certification supplements the USA FCC Class B notice printed on the inside front cover of this manual. The safety instructions apply to all countries.

Product safety

This product is designed to meet the requirements of safety standard IEC 380, Safety of Electrically Energized Office Machines.

Important safety instructions

This equipment is intended to be electrically grounded. This product is equipped with a plug having a third (grounding) pin. This plug will fit only into a grounding-type alternating current outlet. This is a safety feature.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

Do not defeat the purpose of the grounding-type plug.

Power supply specifications

The basic specifications of the power supply furnished with the Apple IIc for use in Europe and other countries having 50-Hz alternating current are shown in Table G-8.

Table G-8
50-Hz power supply specifications

Line voltage	199 to 255 VAC, 50 Hz
Maximum input power consumption	25 W
Supply voltage	+15 VDC (nominal)
Supply current	1.2 A (nominal)

Appendix H

Conversion Tables

This briefly discusses bits and bytes and what they can represent, and peripheral identification numbers. It also contains conversion tables for hexadecimal to decimal and negative decimal, and a number of 8-bit codes.

These tables are intended for convenient reference. This appendix is not intended as a tutorial for the materials discussed. The brief section introductions are for orientation only.

Bits and bytes

This section discusses the relationships between bit values and their position within a byte. Here are some rules of thumb regarding the 65C02:

- A **bit** is a binary digit; it can be either a 0 or a 1.
- A bit can be used to represent any two-way choice. Some choices that a bit can represent in the Apple IIc are listed in Table H-1.
- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of four bits.
- Four bits make a **nibble** (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 and (because our decimal system has only 10 of the 16 digits we need) A through F.
- Eight bits (two nibbles) make a byte (Figure H-1).

- One **byte** can represent any of 16 x 16 (or 256) values. The value can be specified by exactly two hexadecimal digits.
- Bits within a byte are numbered from bit 0 on the right to bit 7 on the left.
- The bit number is the same as the power of 2 that it represents, in a manner completely analogous to the digits in a decimal number.
- One memory position in the Apple IIc contains one 8-bit byte of data.
- How byte values are interpreted depends on whether the byte is an instruction in a language, part or all of an address, an ASCII code, or some other form of data. Tables H-6 through H-9 list some of the ways bytes are commonly interpreted.
- Two bytes make a **word.** The 16 bits of a word can represent any one of 256 x 256 (or 65,536) different values.
- The 65C02 uses a 16-bit word to represent memory locations. It can therefore distinguish among 65,536 (64K) locations at any given time.
- A memory location is one byte of a 256-byte page. The low-order byte of an address specifies this byte. The high-order byte specifies the memory page the byte is on.

Table H-1
What a bit can represent

Context	Representing	0 =	1 =
Binary number	Place value	0	1 x that power of 2
Logic	Condition	False	True
Any switch	Position	Off	On
Any switch	Position	Clear*	Set
Serial transfer	Beginning	Start	Carrier (no information yet)
Serial transfer	Data	0 value	1 value
Serial transfer	Parity	SPACE	MARK
Serial transfer	End		Stop bit(s)
Serial transfer	Communication state	BREAK	Carrier

Table H-1 (continued)
What a bit can represent

Context	Representing	0 =	1 =
P reg. bit N	Neg. result?	No	Yes
P reg. bit V	Overflow?	No	Yes
P reg. bit B	BRK command?	No	Yes
P reg. bit D	Decimal mode?	No	Yes
P reg. bit I	IRQ interrupts	Enabled	Disabled (masked out)
P reg. bit Z	Zero result?	No	Yes
P reg. bit C	Carry required?	No	Yes

* Sometimes ambiguously termed *reset*.

Figure H-1
Bits, nibbles, and bytes

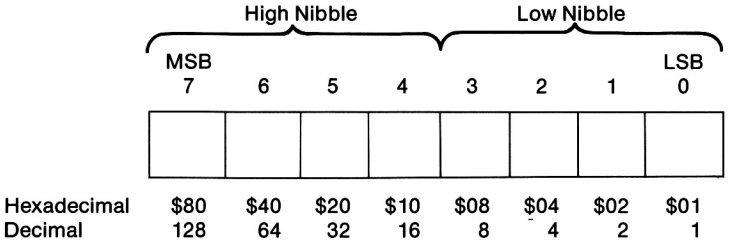


Table H-2
Values represented by a nibble

Binary	Hex	Dec	Binary	Hex	Dec
0000	$0	0	1000	$8	8
0001	$1	1	1001	$9	9
0010	$2	2	1010	$A	10
0011	$3	3	1011	$B	11
0100	$4	4	1100	$C	12
0101	$5	5	1101	$D	13
0110	$6	6	1110	$E	14
0111	$7	7	1111	$F	15

Hexadecimal and decimal

Use Table H-3 for conversion of hexadecimal and decimal numbers.

Table H-3
Hexadecimal/decimal conversion

Digit	$x000	$0x00	$00x0	$000x
F	61440	3840	240	15
E	57344	3584	224	14
D	53248	3328	208	13
C	49152	3072	192	12
B	45056	2816	176	11
A	40960	2560	160	10
9	36864	2304	144	9
8	32768	2048	128	8
7	28672	1792	112	7
6	24576	1536	96	6
5	20480	1280	80	5
4	16384	1024	64	4
3	12288	768	48	3
2	8192	512	32	2
1	4096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up.

For example:

```
$3C = ?
$30 = 48
$0C = 12
--------
$3C = 60
```

```
$FD47 = ?
$F000 = 61440
$ D00 =  3328
$  40 =    64
$   7 =     7
-----------
$FD47 = 64839
```

To convert a decimal number to hexadecimal, subtract from the decimal number the largest decimal entry in the table that is less than it. Write down the hexadecimal digit (noting its place value) also. Now subtract the largest decimal number in the table that is less than the decimal remainder, and write down the next hexadecimal digit. Continue until you have 0 left. Add up the hexadecimal numbers.

For example:

```
16215  = $ ?
16215  - 12288  = 3927    12288  =  $7000
 3927  -  3840  = 87       3840  =  $ F00
   87  -    80  = 7          80  =  $  50
              7                7  =  $   7
                           ------------
                           16215  =  $7F57
```

Hexadecimal and negative decimal

If a number is larger than decimal 32,767, Applesoft BASIC allows and Integer BASIC requires you to use the negative-decimal equivalent of the number. Table H-4 is set up to make it easy for you to convert a hexadecimal number directly to a negative-decimal number.

Table H-4
Hexadecimal to negative decimal conversion

Digit	$x000	$$0x00	$$00x0	$$000x
F	0	0	0	–1
E	–4096	–256	–16	–2
D	–8192	–512	–32	–3
C	–12288	–768	–48	–4
B	–16384	–1024	–64	–5
A	–20480	–1280	–80	–6
9	–24576	–1536	–96	–7
8	–28672	–1792	–112	–8
7		–2048	–128	–9
6		–2304	–144	–10
5		–2560	–160	–11
4		–2816	–176	–12
3		–3072	–192	–13
2		–3328	–208	–14
1		–3584	–224	–15
0		–3840	–240	–16

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (0's included). Then add their values (ignoring their signs for a moment). The resulting number, with a minus sign in front of it, is the desired negative-decimal number.

For example:

```
$C010  =   - ?
$C000:     -12288
$ 000:     - 3840
$  10:     -  224
$   0:     -   16
-------------------
$C010      -16368
```

To convert a negative-decimal number directly to a positive-decimal number, add it to 65,536. (This addition ends up looking like subtraction.)

For example:

```
-151 = + ?
65536 + (-151) = 65536 - 151 = 65385
```

To convert a negative-decimal number to a hexadecimal number, first convert it to a positive-decimal number, then use Table H-3.

Peripheral identification numbers

Many Apple products now use peripheral identification numbers (called *PIN numbers*) as shorthand to designate serial device characteristics. The Apple II series *Universal Utilities* disk presents a menu from which to select the characteristics of, say, a printer or modem. From the selections made, it generates a PIN for the user. Other products have a ready-made PIN that the user can simply type in.

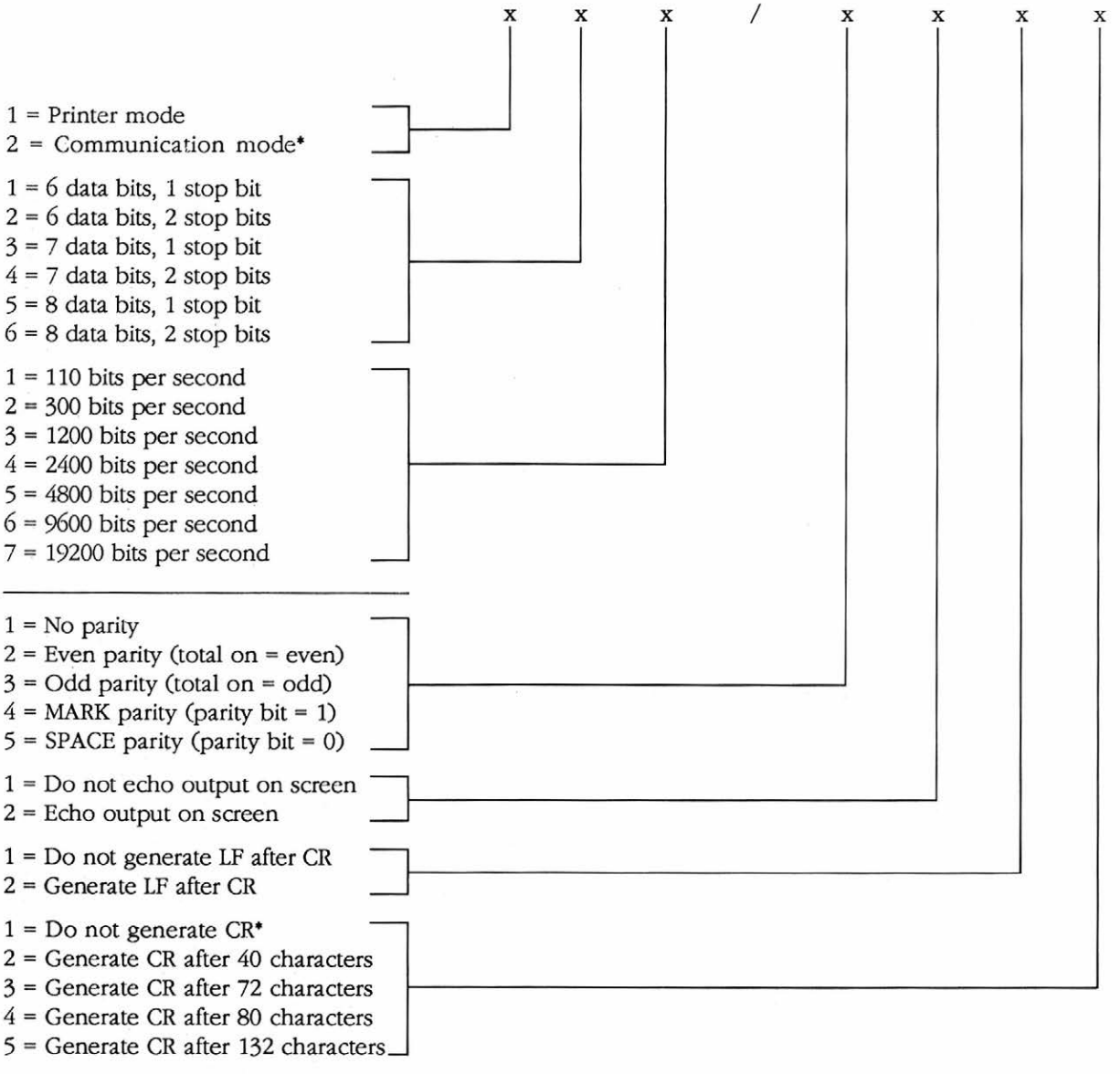
Table H-5 is a definition of the PIN number digits. When communication mode is selected, the seventh digit is ignored.

For example:

252/1111 means:

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

Table H-5
PIN numbers



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

Eight-bit code conversions

Tables H-6 through H-9 show the entire ASCII character set. Note that character values are shown with the high bit off. Unless otherwise noted, all ASCII character values above $7F (127 decimal) generate the same character as that value with the high bit off. Here is how to interpret these tables:

- The *Binary* column has the 8-bit code for each ASCII character.
- The first 128 ASCII entries represent 7-bit ASCII codes plus a high-order bit of 0 (SPACE parity or Pascal)—for example, 01001000 for the letter *H.*
- The last 128 ASCII entries (from 128 through 255) represent 7-bit ASCII codes plus a high-order bit of 1 (MARK parity or BASIC)—for example, 11001000 for the letter *H.*
- A transmitted or received ASCII character will take whichever form (in the communication register) is appropriate if odd or even parity is selected—for example, 11001000 for an odd-parity *H,* 01001000 for an even-parity *H.*
- The *ASCII Char* column gives the ASCII character name.
- The *Interpretation* column spells out the meaning of special symbols and abbreviations, where necessary.
- The *What to type* column indicates what keystrokes generate the ASCII character (where it is not obvious).
- The columns marked *Pri* and *Alt* indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

  Note that the values $40 through $5F (and $C0 through $DF) in the alternate character set are displayed as MouseText characters (Figure 5-1) if the firmware is set to do so, or if the firmware is bypassed.

❖ *Note:* The primary and alternate displayed character sets in Tables H-6 through H-9 are the result of firmware mapping. The character generator ROM actually contains only one character set. The firmware mapping procedure is described in Chapter 3.

Table H-6
Control characters, high bit off

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
0000000	0	$00	NUL	Blank (null)	Control-@	@	@
0000001	1	$01	SOH	Start of header	Control-A	A	A
0000010	2	$02	STX	Start of text	Control-B	B	B
0000011	3	$03	ETX	End of text	Control-C	C	C
0000100	4	$04	EOT	End of transm.	Control-D	D	D
0000101	5	$05	ENQ	Enquiry	Control-E	E	E
0000110	6	$06	ACK	Acknowledge	Control-F	F	F
0000111	7	$07	BEL	Bell	Control-G	G	G
0001000	8	$08	BS	Backspace	Control-H or Left Arrow-H	H	H
0001001	9	$09	HT	Horizontal tab	Control-I or Tab	I	I
0001010	10	$0A	LF	Line feed	Control-J or Down Arrow-J	J	J
0001011	11	$0B	VT	Vertical tab	Control-K or Up Arrow	K	K
0001100	12	$0C	FF	Form feed	Control-L	L	L
0001101	13	$0D	CR	Carriage return	Control-M or Return	M	M
0001110	14	$0E	SO	Shift out	Control-N	N	N
0001111	15	$0F	SI	Shift in	Control-O	O	O
0010000	16	$10	DLE	Data link escape	Control-P	P	P
0010001	17	$11	DC1	Device control 1	Control-Q	Q	Q
0010010	18	$12	DC2	Device control 2	Control-R	R	R
0010011	19	$13	DC3	Device control 3	Control-S	S	S
0010100	20	$14	DC4	Device control 4	Control-T	T	T
0010101	21	$15	NAK	Neg. acknowledge	Control-U or Right Arrow	U	U
0010110	22	$16	SYN	Synchronization	Control-V	V	V
0010111	23	$17	ETB	End of text blk.	Control-W	W	W
0011000	24	$18	CAN	Cancel	Control-X	X	X
0011001	25	$19	EM	End of medium	Control-Y	Y	Y
0011010	26	$1A	SUB	Substitute	Control-Z	Z	Z
0011011	27	$1B	ESC	Escape	Control-[or Escape	[	[
0011100	28	$1C	FS	File separator	Control-\	\	\
0011101	29	$1D	GS	Group separator	Control-]	]	]
0011110	30	$1E	RS	Record separator	Control-^	^	^
0011111	31	$1F	US	Unit separator	Control-_	_	_

Table H-7
Special characters, high bit off

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
0100000	32	$20	SP	Space	Space bar		
0100001	33	$21	!			!	!
0100010	34	$22	"			"	"
0100011	35	$23	#			#	#
0100100	36	$24	$			$	$
0100101	37	$25	%			%	%
0100110	38	$26	&			&	&
0100111	39	$27	'	Apostrophe		'	'
0101000	40	$28	(			(	(
0101001	41	$29	)			)	)
0101010	42	$2A	*			*	*
0101011	43	$2B	+			+	+
0101100	44	$2C	,	Comma		,	,
0101101	45	$2D	-	Hyphen		-	-
0101110	46	$2E	.	Period		.	.
0101111	47	$2F	/			/	/
0110000	48	$30	0			0	0
0110001	49	$31	1			1	1
0110010	50	$32	2			2	2
0110011	51	$33	3			3	3
0110100	52	$34	4			4	4
0110101	53	$35	5			5	5
0110110	54	$36	6			6	6
0110111	55	$37	7			7	7
0111000	56	$38	8			8	8
0111001	57	$39	9			9	9
0111010	58	$3A	:			:	:
0111011	59	$3B	;			;	;
0111100	60	$3C	<			<	<
0111101	61	$3D	=			=	=
0111110	62	$3E	>			>	>
0111111	63	$3F	?			?	?

Table H-8
Uppercase characters, high bit off

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
1000000	64	$40	@			@	
1000001	65	$41	A			A	
1000010	66	$42	B			B	
1000011	67	$43	C			C	
1000100	68	$44	D			D	
1000101	69	$45	E			E	
1000110	70	$46	F			F	
1000111	71	$47	G			G	
1001000	72	$48	H			H	
1001001	73	$49	I			I	
1001010	74	$4A	J			J	
1001011	75	$4B	K			K	
1001100	76	$4C	L			L	
1001101	77	$4D	M			M	
1001110	78	$4E	N			N	
1001111	79	$4F	O			O	
1010000	80	$50	P			P	
1010001	81	$51	Q			Q	
1010010	82	$52	R			R	
1010011	83	$53	S			S	
1010100	84	$54	T			T	
1010101	85	$55	U			U	
1010110	86	$56	V			V	
1010111	87	$57	W			W	
1011000	88	$58	X			X	
1011001	89	$59	Y			Y	
1011010	90	$5A	Z			Z	
1011011	91	$5B	[		Opening bracket	[	
1011100	92	$5C	\		Reverse slant	\	
1011101	93	$5D	]		Closing bracket	]	
1011110	94	$5E	^		Caret	^	
1011111	95	$5F	_		Underline	_	

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

Table H-9
Lowercase characters, high bit off

Binary	Dec	Hex	ASCII char	Interpretation	What to type	Pri	Alt
1100000	96	$60	`	Grave accent			`
1100001	97	$61	a			*!*	**a**
1100010	98	$62	b			*"*	**b**
1100011	99	$63	c			*#*	**c**
1100100	100	$64	d			*$*	**d**
1100101	101	$65	e			*%*	**e**
1100110	102	$66	f			*&*	**f**
1100111	103	$67	g			*'*	**g**
1101000	104	$68	h			*(*	**h**
1101001	105	$69	i			*)*	**i**
1101010	106	$6A	j			*	**j**
1101011	107	$6B	k			+	**k**
1101100	108	$6C	l			,	**l**
1101101	109	$6D	m			-	**m**
1101110	110	$6E	n			.	**n**
1101111	111	$6F	o			*/*	**o**
1110000	112	$70	p			*0*	**p**
1110001	113	$71	q			*1*	**q**
1110010	114	$72	r			*2*	**r**
1110011	115	$73	s			*3*	**s**
1110100	116	$74	t			*4*	**t**
1110101	117	$75	u			*5*	**u**
1110110	118	$76	v			*6*	**v**
1110111	119	$77	w			*7*	**w**
1111000	120	$78	x			*8*	**x**
1111001	121	$79	y			*9*	**y**
1111010	122	$7A	z			*:*	**z**
1111011	123	$7B	{	Opening brace		*;*	**{**
1111100	124	$7C	|	Vertical line		<	**|**
1111101	125	$7D	}	Closing brace		=	**}**
1111110	126	$7E	~	Overline (tilde)		>	**~**
1111111	127	$7F	DEL	Delete/rubout		*?*	**DEL**

Appendix I

Firmware Listings

Appendix I is a listing of the source code for the Monitor firmware (including the Smartport) contained in the memory expansion version of the Apple IIc.

If you are developing products for an earlier version of the IIc, you can obtain a complete set of firmware listings from the Apple Programmer's and Developer's Association (APDA). You can also order additional technical information on other Apple products from the APDA. To obtain the listing set or other technical information, write to

Apple Programmer's and Developer's Association
290 SW 43d Street
Renton, WA 98055
206-251-6548

Table I-1
Main side ROM map

C100	serial port
C200	communications port
C300	80-column routines
C400	memory expansion card: boot code/entry point for ProDOS/entry point for DOS
C500–C58D	UniDisk 3.5 routines
C58E	miscellaneous
C600	Disk II routines
C700–C77F	Mouse entry points
C780	ROM switch routines
C7FC	last screen hole
C800	Interrupt routines
C900	Paddle patch; Mini-Assembler
CA00	Mini-Assembler; step and trace
CB00	scroll routines
CC00	pasinvert; picky; showcur; update; escape
CD00	video routines
CE00	video routines
CF00	video routines; miscellaneous
D000–F7FF	Applesoft BASIC
F800–FFFF	RESET vectors

Table I-2
Auxiliary side ROM map

C100	Mouse interrupt handler; ACIA interrupt handler
C200	continue; ACIA routines
C300	ACIA routines; moveaux; xfer; memory test
C400	continue; switch test
C500–C57F	diagnostics
C780	ROM switch routines
C800–C87F	miscellaneous
C880–CFFF	UniDisk 3.5 routines
D000	command processor for serial and communications ports
D100	continue
D200–D249	Mouse BASIC routines
D24A–D3FF	empty
D400	basilin; hex-to-dec
D4A9–D52B	zero page test
D500–D52B	miscellaneous
D52C–D5FF	empty
D600–D700	Mouse routines
D800	execute; xdiag; xstatus; pstat0; pcn41; pstatus; xread; xwrite; prdblk; pwrblk; pread
D900	pread continue; pwrite
DA00	dospatch; dosconv; stattbl; parmtbl; undtbl; testsize; format
DB00	fmpas; fmdos; makecat; cattbl; procut
DC00	doscut; pascut
DCF0–DCFF	diagnostics (orged at $DD00)
DD00	stattest; addresstest; rollovertest; addbustest; cleartest; fulltest
DE00	continue; computed; pass; fail; miscellaneous
DF00	print; messages
E000–FFF9	empty

```
SOURCE   FILE #01 =>FIRM
 INCLUDE FILE #02 =>NAMES
 INCLUDE FILE #03 =>EQUATES
 INCLUDE FILE #04 =>SERIAL
 INCLUDE FILE #05 =>SER
 INCLUDE FILE #06 =>COMM
 INCLUDE FILE #07 =>C3SPACE
 INCLUDE FILE #08 =>EQUATES2
 INCLUDE FILE #09 =>SLINKY
 INCLUDE FILE #10 =>MISC
 INCLUDE FILE #11 =>BOOT
 INCLUDE FILE #12 =>MOUSE
 INCLUDE FILE #13 =>MCODE.X
 INCLUDE FILE #14 =>SWITCHER
 INCLUDE FILE #15 =>IRQBUF
 INCLUDE FILE #16 =>MINI
 INCLUDE FILE #17 =>SCROLLING
 INCLUDE FILE #18 =>ESCAPE
 INCLUDE FILE #19 =>PASCAL
 INCLUDE FILE #20 =>MOREMISC
 INCLUDE FILE #21 =>AUTOST1
 INCLUDE FILE #22 =>AUTOST2
 INCLUDE FILE #23 =>BANK2
 INCLUDE FILE #24 =>MINT
 INCLUDE FILE #25 =>AUXSTUFF
 INCLUDE FILE #26 =>BANGER2
 INCLUDE FILE #27 =>RW.SLINKY
 INCLUDE FILE #28 =>MCODE.X.AUX
 INCLUDE FILE #29 =>SWITCHER2
 INCLUDE FILE #30 =>COMMAND
 INCLUDE FILE #31 =>MBASIC
 INCLUDE FILE #32 =>BANGER
 INCLUDE FILE #33 =>MOUSEIN7.X
 INCLUDE FILE #34 =>S.EXECUTE
 INCLUDE FILE #35 =>S.MAKECAT
 INCLUDE FILE #36 =>S.DIAG0.SRC
 INCLUDE FILE #37 =>VECTORS2
```



```
0000:          0000    2          x6502

0000:                  3          lst   on,vsym,asym
0000:                  4 ************************************************
0000:                  5 *
0000:                  6 * Firmware for the Apple //c
0000:                  7 * Copyright Apple Computer Inc., 1983,1985,1986
0000:                  8 * All Rights Reserved
0000:                  9 *
0000:                 10 * Initial release -- December 1983
0000:                 11 * Written by Rich Williams, Ernie Beernink and James R Huston
0000:                 12 *
0000:                 13 ************************************************
0000:                 14 *
0000:                 15 * Revision 2 -- May 1985, Richard Williams
0000:                 16 * rom expanded to 32K in 2 16K banks
0000:                 17 * new features added:
0000:                 18 *   Protocol converter slot 5
0000:                 19 *   AppleTalk slot 7
0000:                 20 *   //e diagnostics
0000:                 21 *   Enhanced serial port commands
0000:                 22 *   Mini assembler
0000:                 23 *   Step and trace
0000:                 24 * most $F8 rom changes marked with a +
0000:                 25 *
0000:                 26 ************************************************
0000:                 27 *
0000:                 28 * Revision 3 -- April 1986, Raymond Chiang
0000:                 29 * mouse moved from slot 4 to slot 7
0000:                 30 * appletalk removed from slot 7
0000:                 31 * memory card (slinky) added to slot 4
0000:                 32 * boot sequence is now slot 4, then slot 6 and then slot 5
0000:                 33 * $fbbf the rev byte will be 3.  1 and 2 will not be used
0000:                 34 *
0000:                 35 ************************************************
0000:          F800   36 F8ORG     EQU   $F800

 INCLUDE FILE #02 =>NAMES
C100:                 39          lst   on,vsym,asym
C100:                 40          include equates     ;Equates for Video & Monitor ROM
```

```
C100:                  2 ********************************
C100:                  3 *
C100:                  4 * Apple //c
C100:                  5 * Video Firmware and
C100:                  6 * Monitor ROM Source
C100:                  7 *
C100:                  8 * COPYRIGHT 1977-1983 BY
C100:                  9 * APPLE COMPUTER, INC.
C100:                 10 *
C100:                 11 * ALL RIGHTS RESERVED
C100:                 12 *
C100:                 13 * S. WOZNIAK          1977
C100:                 14 * A. BAUM             1977
C100:                 15 * JOHN A          NOV 1978
C100:                 16 * R. AURICCHIO     SEP 1981
C100:                 17 * E. BEERNINK         1983
C100:                 18 *
C100:                 19 ********************************
C100:                 20 *
C100:                 21 * ZERO PAGE EQUATES
C100:                 22 *
C100: 0000            23 LOC0      EQU   $00          ;vector for autostart from disk
C100: 0001            24 LOC1      EQU   $01
C100: 0020            25 WNDLFT    EQU   $20          ;left edge of text window
C100: 0021            26 WNDWDTH   EQU   $21          ;width of text window
C100: 0022            27 WNDTOP    EQU   $22          ;top of text window
C100: 0023            28 WNDBTM    EQU   $23          ;bottom+1 of text window
C100: 0024            29 CH        EQU   $24          ;cursor horizontal position
C100: 0025            30 CV        EQU   $25          ;cursor vertical position
C100: 0026            31 GBASL     EQU   $26          ;lo-res graphics base addr.
C100: 0027            32 GBASH     EQU   $27
C100: 0028            33 BASL      EQU   $28          ;text base address
C100: 0029            34 BASH      EQU   $29
C100: 002A            35 BAS2L     EQU   $2A          ;temp base for scrolling
C100: 002B            36 BAS2H     EQU   $2B
C100: 002C            37 H2        EQU   $2C          ;temp for lo-res graphics
C100: 002C            38 LMNEM     EQU   $2C          ;temp for mnemonic decoding
C100: 002C            39 RTNL      equ   $2C          ;Step return address
C100: 002D            40 V2        EQU   $2D          ;temp for lo-res graphics
C100: 002D            41 RMNEM     EQU   $2D          ;temp for mnemonic decoding
C100: 002D            42 rtnh      equ   $2D          ;Step return address
C100: 002E            43 MASK      EQU   $2E          ;color mask for lo-res gr.
C100: 002E            44 FORMAT    EQU   $2E          ;temp for opcode decode
C100: 002F            45 LENGTH    EQU   $2F          ;temp for opcode decode
C100: 0030            46 COLOR     EQU   $30          ;color for lo-res graphics
C100: 0031            47 MODE      EQU   $31          ;Monitor mode
C100: 0032            48 INVFLG    EQU   $32          ;normal/inverse(/flash)
C100: 0033            49 PROMPT    EQU   $33          ;prompt character
C100: 0034            50 YSAV      EQU   $34          ;position in Monitor command
C100: 0035            51 YSAV1     EQU   $35          ;temp for Y register
C100: 0036            52 CSWL      EQU   $36          ;character output hook
C100: 0037            53 CSWH      EQU   $37
C100: 0038            54 KSWL      EQU   $38          ;character input hook
C100: 0039            55 KSWH      EQU   $39
C100: 003A            56 PCL       EQU   $3A          ;temp for program counter
C100: 003B            57 PCH       EQU   $3B
C100: 003C            58 XQT       EQU   $3C          ;Step and trace execute area
C100: 003C            59 A1L       EQU   $3C          ;Monitor temp
```



```
C100: 003D            60 A1H       EQU   $3D          ;Monitor temp
C100: 003E            61 A2L       EQU   $3E          ;Monitor temp
C100: 003F            62 A2H       EQU   $3F          ;Monitor temp
C100: 0040            63 A3L       EQU   $40          ;Monitor temp
C100: 0041            64 A3H       EQU   $41          ;Monitor temp
C100: 0042            65 A4L       EQU   $42          ;Monitor temp
C100: 0043            66 A4H       EQU   $43          ;Monitor temp
C100: 0044            67 A5L       EQU   $44          ;Monitor temp
C100: 0045            68 A5H       EQU   $45          ;Monitor temp
C100:                 69 *
C100:                 70 * Note: In Apple II, //e, both interrupts and BRK destroyed
C100:                 71 * location $45.  Now only BRK destroys $45 (ACC) and it
C100:                 72 * also destroys $44 (MACSTAT).
C100:                 73 *
C100: 0044            74 MACSTAT   EQU   $44          ;Machine state after BRK
C100: 0045            75 ACC       EQU   $45          ;Acc after BRK
C100:                 76 *
C100: 0046            77 XREG      EQU   $46          ;X reg after break
C100: 0047            78 YREG      EQU   $47          ;Y reg after break
C100: 0048            79 STATUS    EQU   $48          ;P reg after break
C100: 0049            80 SPNT      EQU   $49          ;SP after break
C100: 004E            81 RNDL      EQU   $4E          ;random counter low
C100: 004F            82 RNDH      EQU   $4F          ;random counter high
C100:                 83 *
C100:                 84 * Value equates
C100:                 85 *
C100: 0006            86 GOODF8    EQU   $06          ;value of //e, lolly ID byte
C100: 0095            87 PICK      EQU   $95          ;CONTROL-U character
C100: 009B            88 ESC       EQU   $9B          ;what ESC generates
C100:                 89 *
C100:                 90 * Characters read by GETLN are placed in
C100:                 91 * IN, terminated by a carriage return.
C100:                 92 *
C100: 0200            93 IN        EQU   $0200        ;input buffer for GETLN
C100:                 94 *
C100:                 95 * Page 3 vectors
C100:                 96 *
C100: 03F0            97 BRKV      EQU   $03F0        ;vectors here after break
C100: 03F2            98 SOFTEV    EQU   $03F2        ;vector for warm start
C100: 03F4            99 PWREDUP   EQU   $03F4        ;THIS MUST = EOR #$A5 OF SOFTEV+1
C100: 03F5           100 AMPERV    EQU   $03F5        ;APPLESOFT & EXIT VECTOR
C100: 03F8           101 USRADR    EQU   $03F8        ;APPLESOFT USR function vector
C100: 03FB           102 NMI       EQU   $03FB        ;NMI vector
C100: 03FE           103 IRQLOC    EQU   $03FE        ;Maskable interrupt vector
C100: 0400           104 LINE1     EQU   $0400        ;first line of text screen
C100: 07F8           105 MSLOT     EQU   $07F8        ;owner of $C8 space
C100:                106 *
C100:                107 * HARDWARE EQUATES
C100:                108 *
C100: C000           109 IOADR     EQU   $C000        ;for IN#, PR# vector
C100: C000           110 KBD       EQU   $C000        ;>127 if keystroke
C100: C000           111 CLR80COL  EQU   $C000        ;disable 80 column store
C100: C001           112 SET80COL  EQU   $C001        ;enable 80 column store
C100: C002           113 RDMAINRAM EQU   $C002        ;read from main 48K RAM
C100: C003           114 RDCARDRAM EQU   $C003        ;read from alt. 48K RAM
C100: C004           115 WRMAINRAM EQU   $C004        ;write to main 48K RAM
C100: C005           116 WRCARDRAM EQU   $C005        ;write to alt. 48K RAM
C100: C008           117 SETSTDZP  EQU   $C008        ;use main zero page/stack
```

```
C100:          C009  118 SETALTZP  EQU   $C009        ;use alt. zero page/stack
C100:          C00C  119 CLR80VID  EQU   $C00C        ;disable 80 column hardware
C100:          C00D  120 SET80VID  EQU   $C00D        ;enable 80 column hardware
C100:          C00E  121 CLRALTCHAR EQU  $C00E        ;normal LC, flashing UC
C100:          C00F  122 SETALTCHAR EQU  $C00F        ;normal inverse, LC; no flash
C100:          C010  123 KBDSTRB   EQU   $C010        ;turn off key pressed flag
C100:          C011  124 RDLCBNK2  EQU   $C011        ;>127 if LC bank 2 is in
C100:          C012  125 RDLCRAM   EQU   $C012        ;>127 if LC RAM read enabled
C100:          C013  126 RDRAMRD   EQU   $C013        ;>127 if reading main 48K
C100:          C014  127 RDRAMWRT  EQU   $C014        ;>127 if writing main 48K
C100:          C016  128 RDALTZP   EQU   $C016        ;>127 if Alt ZP and LC switched in
C100:          C018  129 RD80COL   EQU   $C018        ;>127 if 80 column store
C100:          C019  130 RDVBLBAR  EQU   $C019        ;>127 if not VBL
C100:          C01A  131 RDTEXT    EQU   $C01A        ;>127 if text (not graphics)
C100:          C01B  132 RDMIX     EQU   $C01B        ;>127 if mixed mode on
C100:          C01C  133 RDPAGE2   EQU   $C01C        ;>127 if TXTPAGE2 switched in
C100:          C01D  134 RDHIRES   EQU   $C01D        ;>127 if HIRES is on
C100:          C01E  135 ALTCHARSET EQU  $C01E        ;>127 if alternate char set in use
C100:          C01F  136 RD80VID   EQU   $C01F        ;>127 if 80 column hardware in
C100:          C028  137 ROMBANK   EQU   $C028        ;Switches rombanks
C100:          C030  138 SPKR      EQU   $C030        ;clicks the speaker
C100:          C050  139 TXTCLR    EQU   $C050        ;switch in graphics (not text)
C100:          C051  140 TXTSET    EQU   $C051        ;switch in text (not graphics)
C100:          C052  141 MIXCLR    EQU   $C052        ;clear mixed-mode
C100:          C053  142 MIXSET    EQU   $C053        ;set mixed-mode (4 lines text)
C100:          C054  143 TXTPAGE1  EQU   $C054        ;switch in text page 1
C100:          C055  144 TXTPAGE2  EQU   $C055        ;switch in text page 2
C100:          C056  145 LORES     EQU   $C056        ;low-resolution graphics
C100:          C057  146 HIRES     EQU   $C057        ;high-resolution graphics
C100:          C058  147 CLRAN0    EQU   $C058
C100:          C059  148 SETAN0    EQU   $C059
C100:          C05A  149 CLRAN1    EQU   $C05A
C100:          C05B  150 SETAN1    EQU   $C05B
C100:          C05C  151 CLRAN2    EQU   $C05C
C100:          C05D  152 SETAN2    EQU   $C05D
C100:          C05E  153 CLRAN3    EQU   $C05E
C100:          C05F  154 SETAN3    EQU   $C05F
C100:          C060  155 RD40SW    EQU   $C060        ;>127 if 40/80 switch in 40 pos
C100:          C061  156 BUTN0     EQU   $C061        ;open apple key
C100:          C062  157 BUTN1     EQU   $C062        ;closed apple key
C100:          C064  158 PADDL0    EQU   $C064        ;read paddle 0
C100:          C070  159 PTRIG     EQU   $C070        ;trigger the paddles
C100:          C081  160 ROMIN     EQU   $C081        ;switch in $D000-$FFFF ROM
C100:          C083  161 LCBANK2   EQU   $C083        ;switch in LC bank 2
C100:          C08B  162 LCBANK1   EQU   $C08B        ;switch in LC bank 1
C100:          CFFF  163 CLRROM    EQU   $CFFF        ;switch out $C8 ROMs
C100:          E000  164 BASIC     EQU   $E000        ;BASIC entry point
C100:          E003  165 BASIC2    EQU   $E003        ;BASIC warm entry point
C100:                166 *
C100:          04FB  167 VMODE     EQU   $4F8+3       ;OPERATING MODE
C100:                168 *
C100:                169 * BASIC VMODE BITS
C100:                170 *
C100:                171 * 1....... - BASIC active
C100:                172 * 0....... - Pascal active
C100:                173 * .0......
C100:                174 * .1......
C100:                175 * ..0..... - Print control characters
```



```
C100:                176 * ..1..... - Don't print ctrl chars
C100:                177 * ...0.... -
C100:                178 * ...1.... -
C100:                179 * ....0... - Print control characters
C100:                180 * ....1... - Don't print ctrl chars.
C100:                181 * .....0.. -
C100:                182 * .....1.. -
C100:                183 * ......0. -
C100:                184 * ......1. -
C100:                185 * .......0 - Print mouse characters
C100:                186 * .......1 - Don't print mouse characters
C100:                187 *
C100:          0040  188 M.40      EQU   $40
C100:          0020  189 M.CTL2    EQU   $20          ;Don't print controls
C100:          0008  190 M.CTL     EQU   $08          ;Don't print controls
C100:          0001  191 M.MOUSE   EQU   $01          ;Don't print mouse chars
C100:                192 *
C100:                193 * Pascal Mode Bits
C100:                194 *
C100:                195 * 1....... - BASIC active
C100:                196 * 0....... - Pascal active
C100:                197 * .0......
C100:                198 * .1......
C100:                199 * ..0..... -
C100:                200 * ..1..... -
C100:                201 * ...0.... - Cursor always on
C100:                202 * ...1.... - Cursor always off
C100:                203 * ....0... - GOTOXY n/a
C100:                204 * ....1... - GOTOXY in progress
C100:                205 * .....0.. - Normal Video
C100:                206 * .....1.. - Inverse Video
C100:                207 * ......0. -
C100:                208 * ......1. -
C100:                209 * .......0 - Print mouse chars
C100:                210 * .......1 - Don't print mouse chars
C100:                211 *
C100:          0080  212 M.PASCAL  EQU   $80          ;Pascal active
C100:          0010  213 M.CURSOR  EQU   $10          ;Don't print cursor
C100:          0008  214 M.GOXY    EQU   $08          ;GOTOXY IN PROGRESS
C100:          0004  215 M.VMODE   EQU   $04
C100:                216 *
C100:          0478  217 ROMSTATE  EQU   $478         ;temp store of ROM state
C100:          04F8  218 TEMP1     EQU   $4F8         ;used by CTLCHAR
C100:          0578  219 TEMPA     EQU   $578         ;used by scroll
C100:          05F8  220 TEMPY     EQU   $5F8         ;used by scroll
C100:                221 *
C100:          047B  222 OLDCH     EQU   $478+3       ;last value of CH
C100:          057B  223 OURCH     EQU   $578+3       ;80-COL CH
C100:          05FB  224 OURCV     EQU   $5F8+3       ;CURSOR VERTICAL
C100:          067B  225 VFACTV    EQU   $678+3       ;Bit7=video firmware inactive
C100:          06FB  226 XCOORD    EQU   $6F8+3       ;X-COORD (GOTOXY)
C100:          077B  227 NXTCUR    EQU   $778+3       ;next cursor to display
C100:          07FB  228 CURSOR    EQU   $7F8+3       ;the current cursor char
C100:                229 *
C100:                230 * Disk II boot rom equates
C100:                231 *
C100:          0356  232 DNIBL     EQU   $356
C100:          0300  233 NBUF1     EQU   $300
```

```
03 EQUATES        Apple //c Video Firmware          20-OCT-86  06:41 PAGE 7

C100:       002B  234 SLOTZ     EQU   $2B
C100:       003C  235 BOOTTMP   EQU   $3C
C100:       004F  236 BOOTDEV   EQU   $4F



C100:             238 ****************************************
C100:             239 * Entry points for other modules
C100:             240 ****************************************
C100:       C880  241 pcnv      equ   $C880
C100:       C5F5  242 bootfail  equ   $C5F5         ;Boot fails message
C100:       C5F8  243 pcnvrst   equ   $C5F8         ;Protocol converter reset
C100:       C580  244 atalk     equ   $C580         ;Apple talk
C100:              41           include serial      ;Equates for serial code
```

```
04 SERIAL         Serial & Communications equates   20-OCT-86  06:41 PAGE 8

C100:               3 ****************************************
C100:               4 *
C100:               5 * Apple Lolly communications driver
C100:               6 *
C100:               7 * By
C100:               8 * Rich Williams
C100:               9 * August 1983
C100:              10 * November 5 - j.r.huston
C100:              11 *
C100:              12 ****************************************
C100:              13 *
C100:              14 * Command codes
C100:              15 *
C100:              16 * Default command char is ctrl-A (^A)
C100:              17 *
C100:              18 *     ^AnnB: Set baud rate to nn
C100:              19 *     ^AnnD: Set data format bits to nn
C100:              20 *     ^AI:   Enable video echo
C100:              21 *     ^AK:   Disable CRLF
C100:              22 *     ^AL:   Enable CRLF
C100:              23 *     ^AnnN: Disable video echo & set printer width
C100:              24 *     ^AnnP: Set parity bits to nn
C100:              25 *     ^AQ    Quit terminal mode
C100:              26 *     ^AR    Reset the ACIA, IN#0 PR#0
C100:              27 *     ^AS    Send a 233 ms break character
C100:              28 *     ^AT    Enter terminal mode
C100:              29 *     ^AZ:   Zap control commands
C100:              30 *     ^Ax:   Set command char to ^x
C100:              31 *     ^AnnCR:Set printer width (CR = carriage return)
C100:              32 *
C100:              33 * New commands added in rev 1 E = enable D = Disable
C100:              34 *
C100:              35 *     ^AC E/D Column overflow
C100:              36 *     ^AL E/D Linefeed same as L & K
C100:              37 *     ^AM E/D Mask incoming linefeeds
C100:              38 *     ^AX E/D Xon Xoff handshaking
C100:              39 *     ^AF E/D Find keyboard
C100:              40 *
C100:              41 ****************************************
C100:       C100   42 serslot   equ   $C100
C100:       C200   43 comslot   equ   $C200
C100:              44           msb   ON
C100:       00BF   45 cmdcur    equ   '?'          ;Cursor while in command mode
C100:       00DF   46 termcur   equ   ' '          ;Cursor while in terminal mode
C100:              47           msb   OFF
C100:       008A   48 lfeed     equ   $8A          ;Linefeed
C100:       0091   49 xon       equ   $91          ;XON character
C100:       0093   50 xoff      equ   $93          ;XOFF character
C100:       03B8   51 sermode   equ   $3B8         ;D7=1 if in command D6=1 if terminal
C100:       0438   52 astat     equ   $438         ;Acia status from int 4F9 & 4FA
C100:       04B8   53 pwdth     equ   $4B8         ;Printer width 579 & 57A
C100:       0538   54 extint    equ   $538         ;extint & typhed enable 5F9 & 5FA
C100:       05F9   55 extint2   equ   $5F9
C100:       05FA   56 typhed    equ   $5FA
C100:       0679   57 oldcur    equ   $679         ;Saves cursor while in command
C100:       067A   58 oldcur2   equ   $67A         ;Saves cursor while in terminal mode
C100:       0638   59 eschar    equ   $638         ;Current escape character 6F9 & 6FA
C100:       06B8   60 flags     equ   $6B8         ;D7 = Video echo D6 = CRLF 779 & 77A
```

```
C100:      0738   61 col       equ   $738        ;Current printer column 7F9 & 7FA
C100:      077E   62 number    equ   $77E        ;Number accumulated in command
C100:      04FC   63 aciabuf   equ   $4FC        ;Owner of serial buffer
C100:      057C   64 twser     equ   $57C        ;Storage pointer for serial buffer
C100:      05FC   65 twkey     equ   $5FC        ;Storage pointer for type ahead buffer
C100:      067C   66 trser     equ   $67C        ;Retrieve pointer for serial buffer
C100:      06FF   67 trkey     equ   $6FF        ;Retrieve buffer for type ahead buffer
C100:      0800   68 thbuf     equ   $800        ;Buffer in alt ram space
C100:      06F8   69 temp      equ   $6F8        ;Temp storage
C100:      05FE   70 charbuf   equ   $5FE        ;5FE, 67E are one byte character buffers
C100:      BFF8   71 sdata     equ   $BFF8       ;+$N0+$90 is output port
C100:      BFF9   72 sstat     equ   $BFF9       ;ACIA status register
C100:      BFFA   73 scomd     equ   $BFFA       ;ACIA command register
C100:      BFFB   74 scntl     equ   $BFFB       ;ACIA control register
C100:             42           include ser       ;Printer port @ $C100
```



```
C100:                 3 *org serslot
C100:2C 89 C1         4          bit   serrts      ;Set V to indicate initial entry
C103:70 0C   C111     5          bvs   entr1       ;Always taken
C105:38               6          sec                ;Input entry point
C106:90               7          dfb   $90         ;BCC opcode
C107:18               8          clc
C108:B8               9          clv                ;V = 0 since not initial entry
C109:50 06   C111     10         bvc   entr1       ;Always taken

C10B:01               12         dfb   $01         ;pascal signiture byte
C10C:31               13         dfb   $31         ;device signiture
C10D:9E               14         dfb   >p1init
C10E:A8               15         dfb   >p1read
C10F:B4               16         dfb   >p1write
C110:BB               17         dfb   >p1status

C111:DA               19 entr1   phx                ;Save the reg
C112:A2 C1            20         ldx   #<serslot   ;X = Cn
C114:4C 1C C2         21         jmp   setup       ;Set mslot, etc
C117:90 03   C11C     22 serport bcc   serisout    ;Only output allowed
C119:4C E5 C7         23         jmp   swzznm      ;Reset the hooks
C11C:0A               24 serisout asl  A           ;A = flags
C11D:7A               25         ply                ;Get char
C11E:5A               26         phy
C11F:BD B8 04         27         lda   pwdth,x     ;Formatting enabled?
C122:F0 42   C166     28         beq   prnow
C124:A5 24            29         lda   ch          ;Get current horiz position
C126:B0 1C   C144     30         bcs   servid      ;Branch if video echo
C128:DD B8 04         31         cmp   pwdth,x     ;If CH >= PWIDTH, then CH = COL
C12B:90 03   C130     32         bcc   chok
C12D:BD 38 07         33         lda   col,x
C130:DD 38 07         34 chok    cmp   col,x       ;Must be > col for valid tab
C133:B0 0B   C140     35         bcs   fixch       ;Branch if ok
C135:C9 11            36         cmp   #$11        ;8 or 16?
C137:B0 11   C14A     37         bcs   prnt        ;If > forget it
C139:09 F0            38         ora   #$F0        ;Find next comma cheaply
C13B:3D 38 07         39         and   col,x       ;Don't blame me it's Dick's trick
C13E:65 24            40         adc   ch
C140:85 24            41 fixch   sta   ch          ;Save the new position
C142:80 06   C14A     42         bra   prnt
C144:C5 21            43 servid  cmp   wndwdth     ;If ch>= wndwdth go back to start of line
C146:90 02   C14A     44         blt   prnt
C148:64 24            45         stz   ch          ;Go back to left edge


C14A:                 47 * We have a char to print
C14A:7A               48 prnt    ply
C14B:5A               49         phy
C14C:BD 38 07         50         lda   col,x       ;Have we exceeded width?
C14F:DD B8 04         51         cmp   pwdth,x
C152:B0 08   C15C     52         bge   toofar
C154:C5 24            53         cmp   ch          ;Are we tabbing?
C156:B0 0E   C166     54         bge   prnow
C158:A9 40            55         lda   #$40        ;Space * 2
C15A:80 02   C15E     56         bra   tab
C15C:A9 1A            57 toofar  lda   #$1A        ;CR * 2
C15E:C0 80            58 tab     cpy   #$80        ;C = High bit
```

```
05 SER            Serial output port routine        20-OCT-86  06:41 PAGE 11

C160:6A             59           ror   A           ;Shift it into char
C161:20 9B C1       60           jsr   goser3      ;Out it goes
C164:80 E4   C14A   61           bra   prnt
C166:98             62 prnow     tya
C167:20 8A C1       63           jsr   serout      ;Print the actual char
C16A:BD B8 04       64           lda   pwdth,x     ;Formatting enabled
C16D:F0 17   C186   65           beq   done
C16F:3C B8 06       66           bit   flags,x     ;In video echo?
C172:30 12   C186   67           bmi   done
C174:BD 38 07       68           lda   col,x       ;Check if within 8 chars of right edge
C177:FD B8 04       69           sbc   pwdth,x     ;So BASIC can format output
C17A:C9 F8          70           cmp   #$F8
C17C:90 04   C182   71           bcc   setch       ;If not within 8, we're done
C17E:18             72           clc
C17F:65 21          73           adc   wndwdth
C181:AC             74           dfb   $AC         ;Dummy LDY to skip next two bytes
C182:A9 00          75 setch     lda   #0          ;Keep cursor at 0 if video off
C184:85 24          76           sta   ch
C186:68             77 done      pla               ;Restore regs
C187:7A             78           ply
C188:FA             79           plx
C189:60             80 serrts    rts



C18A:       C18A    82 serout    equ   *           ;Serial output
C18A:20 A9 C7       83           jsr   swcmd       ;Check if command
C18D:90 FA   C189   84           bcc   serrts      ;All done if it is
C18F:       C18F    85 serout2   equ   *
C18F:3C B8 06       86           bit   flags,x     ;N=1 iff video on
C192:10 07   C19B   87           bpl   goser3
C194:C9 91          88           cmp   #xon        ;Don't echo ^Q
C196:F0 03   C19B   89           beq   goser3
C198:20 F0 FD       90           jsr   cout1       ;Echo it
C19B:4C CD C7       91 goser3    jmp   swser3      ;Go to serout3



C19E:               93 * Pascal support stuff
C19E:5A             94 plinit    phy
C19F:48             95           pha
C1A0:20 B6 C2       96           jsr   default     ;set defaults, enable acia
C1A3:9E B8 06       97           stz   flags,x
C1A6:80 07   C1AF   98           bra   plread2     ;all done...

C1A8:5A            100 plread    phy
C1A9:20 D9 C7      101           jsr   swread      ;read data from serial port (or buffer)
C1AC:90 FA   C1A8  102           bcc   plread      ;Branch if data not ready
C1AE:90            103           dfb   $90         ;BCC to skip pla
C1AF:68            104 plread2   pla
C1B0:7A            105           ply
C1B1:A2 00         106           ldx   #0
C1B3:60            107           rts

C1B4:5A            109 plwrite   phy
C1B5:48            110           pha
C1B6:20 8A C1      111           jsr   serout      ;Go output character
C1B9:80 F4   C1AF  112           bra   plread2
```

```
05 SER            Serial output port routine        20-OCT-86  06:41 PAGE 12

C1BB:5A            113 plstatus  phy
C1BC:48            114           pha
C1BD:4A            115           lsr   A           ;C = 0 output, 1 input
C1BE:D0 15   C1D5  116           bne   plerr       ;Branch if bad call
C1C0:08            117           php
C1C1:20 D3 C7      118           jsr   swgetst     ;Get status in A
C1C4:28            119           plp
C1C5:90 05   C1CC  120           bcc   plstwr
C1C7:29 28         121           and   #$28        ;Test DCD = 0 & rcvr full
C1C9:0A            122           asl   A           ;$08 -> $10
C1CA:80 02   C1CE  123           bra   plstrd
C1CC:29 30         124 plstwr    and   #$30        ;Test DCD = 0 & xmit empty
C1CE:C9 10         125 plstrd    cmp   #$10        ;Is it what we want?
C1D0:F0 DD   C1AF  126           beq   plread2     ;C = 1 if equal
C1D2:18            127           clc               ;Not ready
C1D3:80 DA   C1AF  128           bra   plread2
C1D5:A2 40         129 plerr     ldx   #$40        ;Bad call
C1D7:68            130           pla
C1D8:7A            131           ply
C1D9:18            132           clc
C1DA:60            133           rts

C1DB:              135           MSB   ON
C1DB:       C1DB   136 utsmsg    equ   *
C1DB:D5 CE C1 C2   137           asc   'UNABLE     TO START FROM MEMORY CARD'
C1FB:00            138           dfb   0
C1FC:              139           MSB   OFF

C1FC:       0004   141           ds    comslot-*,$00
C200:               43           include comm      ;Communications port @ $C200
```

```
C200:2C 89 C1      3          bit   serrts       ;Set V to indicate initial entry
C203:70 14   C219  4          bvs   entr
C205:38            5 sin      sec                 ;Input entry point
C206:90            6          dfb   $90          ;BCC opcode to skip next byte
C207:18            7 sout     clc                 ;Output entry point
C208:B8            8          clv                 ;Mark not initial entry
C209:50 0E   C219  9          bvc   entr          ;Branch around pascal entry stuff

C20B:01           11          dfb   $01          ;pascal signiture byte
C20C:31           12          dfb   $31          ;device signiture
C20D:11           13          dfb   >p2init
C20E:13           14          dfb   >p2read
C20F:15           15          dfb   >p2write
C210:17           16          dfb   >p2status

C211:             18 * Pascal support stuff

C211:80 8B   C19E 20 p2init   bra   p1init
C213:80 93   C1A8 21 p2read   bra   p1read
C215:80 9D   C1B4 22 p2write  bra   p1write
C217:80 A2   C1BB 23 p2status bra   p1status

C219:DA           25 entr     phx
C21A:A2 C2        26          ldx   #<comslot    ;X = <CN00
C21C:       C21C  27 setup    equ   *
C21C:5A           28          phy
C21D:48           29          pha
C21E:8E F8 07     30          stx   mslot
C221:50 22   C245 31          bvc   sudone       ;First call?
C223:A5 36        32          lda   cswl         ;If both hooks CN00 setup defaults
C225:45 38        33          eor   kswl
C227:F0 06   C22F 34          beq   sudodef
C229:A5 37        35          lda   cswh         ;If both hooks CN then don't do def
C22B:C5 39        36          cmp   kswh         ;since it has already been done
C22D:F0 03   C232 37          beq   sunodef
C22F:20 B6 C2     38 sudodef  jsr   default      ;Set up defaults
C232:8A           39 sunodef  txa
C233:45 39        40          eor   kswh         ;Input call?
C235:05 38        41          ora   kswl
C237:D0 07   C240 42          bne   suout        ;Must be Cn00
C239:A9 05        43          lda   #>sin        ;Fix the input hook
C23B:85 38        44          sta   kswl
C23D:38           45          sec                 ;C = 1 for input call
C23E:80 05   C245 46          bra   sudone
C240:A9 07        47 suout    lda   #>sout       ;Fix output hook
C242:85 36        48          sta   cswl         ;Note C might not be 0
C244:18           49          clc                 ;C=0 for output
C245:BD B8 06     50 sudone   lda   flags,x      ;Check if serial or comm port
C248:89 01        51          bit   #1           ;Leave flags in a for serport
C24A:D0 03   C24F 52          bne   commport
C24C:4C 17 C1     53 comout   jmp   serport
C24F:90 FB   C24C 54 commport bcc   comout       ;Output?
C251:68           55          pla                 ;Get the char
C252:80 28   C27C 56          bra   term1        ;Input
C254:3C B8 03     57 noesc    bit   sermode,x    ;In terminal mode?
C257:50 1C   C275 58          bvc   exit1        ;If not, return key
C259:20 8F C1     59          jsr   serout2      ;Out it goes
C25C:80 1E   C27C 60          bra   term1
```



```
C25E:       C25E  61 testkbd  equ   *
C25E:68           62          pla                 ;Get current char
C25F:20 70 CC     63          jsr   update       ;Update cursor & check keyboard
C262:10 1B   C27F 64          bpl   serin        ;N=0 if no new key
C264:20 A9 C7     65          jsr   swcmd        ;Test for command
C267:B0 EB   C254 66          bcs   noesc        ;Branch if not
C269:29 5F        67          and   #$5f         ;upshift for following tests
C26B:C9 51        68          cmp   #'Q'         ;Quit?
C26D:F0 04   C273 69          beq   exitX
C26F:C9 52        70          cmp   #'R'         ;Reset?
C271:D0 09   C27C 71          bne   term1        ;Go check serial
C273:A9 98        72 exitX    lda   #$98         ;return a CTRL-X
C275:7A           73 exit1    ply
C276:FA           74          plx
C277:60           75          rts
C278:18           76 goremote clc                 ;Into remote mode
C279:20 A3 C7     77 goterm   jsr   swsttm       ;Into terminal mode
C27C:       C27C  78 term1    equ   *
C27C:20 4C CC     79          jsr   showcur      ;Get current char on screen
C27F:48           80 serin    pha
C280:20 D9 C7     81 sinokbd  jsr   swread       ;Is it ready?
C283:B0 09   C28E 82          bcs   sidata       ;Branch if we got data
C285:BD B8 06     83          lda   flags,x      ;Is keyboard enabled?
C288:29 10        84          and   #$10
C28A:F0 D2   C25E 85          beq   testkbd      ;Branch if enabled
C28C:80 F2   C280 86          bra   sinokbd      ;Go test acia again
C28E:A8           87 sidata   tay                 ;Save new input in y for now
C28F:68           88          pla
C290:5A           89          phy                 ;Save new char on stack
C291:20 B8 C3     90          jsr   storch       ;Fix the screen
C294:68           91          pla                 ;Get the new data
C295:BC 38 06     92          ldy   eschar,x     ;If 0, don't modify char
C298:F0 12   C2AC 93          beq   sinomod
C29A:09 80        94          ora   #$80         ;Apple loves the high bit
C29C:C9 91        95          cmp   #xon
C29E:F0 DC   C27C 96          beq   term1        ;Ignore ^Q
C2A0:C9 FF        97          cmp   #$FF         ;Ignore FFs
C2A2:F0 D8   C27C 98          beq   term1
C2A4:C9 92        99          cmp   #$92         ;^R for remote?
C2A6:F0 D0   C278 100         beq   goremote
C2A8:C9 94        101         cmp   #$94         ;^T for terminal mode?
C2AA:F0 CD   C279 102         beq   goterm
C2AC:3C B8 03     103 sinomod bit   sermode,x    ;In terminal mode?
C2AF:50 C4   C275 104         bvc   exit1        ;Return to user if not A = char
C2B1:20 ED FD     105         jsr   cout         ;Onto the screen with it
C2B4:80 C6   C27C 106         bra   term1
C2B6:       C2B6  107 default   equ   *           ;Set up the defaults
C2B6:20 A0 CF     108         jsr   moveirq      ;make sure irq vectors ok
C2B9:BC 29 C2     109         ldy   defidx-$C1,x ;Index into alt screen. Table in command
C2BC:20 7C C3     110 defloop jsr   getalt       ;Get default from alt screen
C2BF:48           111         pha
C2C0:88           112         dey
C2C1:30 04   C2C7 113         bmi   defff        ;Done if minus
C2C3:C0 03        114         cpy   #3
C2C5:D0 F5   C2BC 115         bne   defloop      ;Or if 2
C2C7:20 A0 CF     116 defff   jsr   moveirq      ;Jam irq vector into LC
C2CA:68           117         pla                 ;Command, control & flags on stack
C2CB:BC 2B C2     118         ldy   devno,x
```

```
06 COMM          Communications port routine        20-OCT-86  06:41 PAGE 15

C2CE:99 FB BF     119           sta   scntl,y       ;Set command reg
C2D1:68           120           pla
C2D2:99 FA BF     121           sta   scomd,y
C2D5:68           122           pla
C2D6:9D B8 06     123           sta   flags,x       ;And the flags
C2D9:29 01        124           and   #1            ;A = $01 (^A) if comm mode
C2DB:D0 02   C2DF 125           bne   defcom
C2DD:A9 09        126           lda   #9            ;^I for serial port
C2DF:9D 38 06     127 defcom    sta   eschar,x
C2E2:68           128           pla                  ;Get printer width
C2E3:9D B8 04     129           sta   pwdth,x
C2E6:9E B8 03     130           stz   sermode,x
C2E9:60           131           rts
C2EA:03 07        132 defidx    dfb   3,7
C2EC:       00C1  133 sltdmy    equ   <serslot      ;Make table for hardware access
C2EC:       C22B  134 devno     equ   *-sltdmy
C2EC:A0 B0        135           dfb   $A0,$B0

C2EE:       0012  137           ds    $C300-*,$00
C300:              44           include c3space     ;80 column card @ $C300
```

```
07 C3SPACE       Communications port routine        20-OCT-86  06:41 PAGE 16

C300:              2 ****************************************
C300:              3 *
C300:              4 * THIS IS THE $C3XX ROM SPACE:
C300:              5 *
C300:              6 ****************************************
C300:48            7 C3ENTRY   PHA                 ;save regs
C301:DA            8           PHX
C302:5A            9           PHY
C303:80 12   C317 10           BRA   BASICINIT     ;and init video firmware
C305:38           11 C3KEYIN   SEC                 ;Pascal 1.1 ID byte
C306:90           12           DFB   $90           ;BCC OPCODE (NEVER TAKEN)
C307:18           13 C3COUT1   CLC                 ;Pascal 1.1 ID byte
C308:80 1A   C324 14           BRA   BASICENT      ;=>go print/read char
C30A:EA           15           NOP
C30B:             16 *
C30B:             17 * PASCAL 1.1 FIRMWARE PROTOCOL TABLE:
C30B:             18 *
C30B:01           19           DFB   $01           ;GENERIC SIGNATURE BYTE
C30C:88           20           DFB   $88           ;DEVICE SIGNATURE BYTE
C30D:             21 *
C30D:2C           22           DFB   >JPINIT       ;PASCAL INIT
C30E:2F           23           DFB   >JPREAD       ;PASCAL READ
C30F:32           24           DFB   >JPWRITE      ;PASCAL WRITE
C310:35           25           DFB   >JPSTAT       ;PASCAL STATUS
C311:             26 ****************************************
C311:             27 *
C311:             28 * 128K SUPPORT ROUTINE ENTRIES:
C311:             29 *
C311:4C AF C7     30           JMP   SWAUX         ;MEMORY MOVE ACROSS BANKS
C314:4C B5 C7     31           JMP   SWXFER        ;TRANSFER ACROSS BANKS
C317:             32 ****************************************
C317:             33 *
C317:             34 ****************************************
C317:             35 * BASIC I/O ENTRY POINT:
C317:             36 ****************************************
C317:             37 *
C317:20 20 CE     38 BASICINIT JSR   HOOKUP        ;COPYROM if needed, sethooks
C31A:20 BE CD     39           JSR   SET80         ;setup 80 columns
C31D:20 58 FC     40           JSR   HOME          ;clear screen
C320:7A           41           PLY
C321:FA           42           PLX                 ;restore X
C322:68           43           PLA                 ;restore char
C323:18           44           CLC                 ;output a character
C324:             45 *
C324:B0 03   C329 46 BASICENT  BCS   BINPUT        ;=>carry me to input
C326:4C F6 FD     47 BPRINT    JMP   COUT          ;print a character
C329:4C 1B FD     48 BINPUT    JMP   KEYIN         ;get a keystroke
C32C:             49 *
C32C:4C 41 CF     50 JPINIT    JMP   PINIT         ;pascal init
C32F:4C 35 CF     51 JPREAD    JMP   PASREAD       ;pascal read
C332:4C C2 CE     52 JPWRITE   JMP   PWRITE        ;pascal write
C335:4C B1 CE     53 JPSTAT    JMP   PSTATUS       ;pascal status call
C338:             54 *
C338:             55 * COPYROM is called when the video firmware is
C338:             56 * initialized.  If the language card is switched
C338:             57 * in for reading, it copies the F8 ROM to the
C338:             58 * language card and restores the state of the
C338:             59 * language card.
```

```
C338:                   60 *
C338:A9 06              61 COPYROM  LDA   #GOODF8       ;get the ID byte
C33A:                   62 *
C33A:                   63 * Compare ID bytes to whatever is readable.  If it
C33A:                   64 * matches, all is ok.  If not, need to copy.
C33A:                   65 *
C33A:CD B3 FB           66          CMP   F8VERSION     ;does it match?
C33D:F0 3C   C37B       67          BEQ   ROMOK
C33F:20 60 C3           68          JSR   SETROM        ;read ROM, write RAM, save state
C342:A9 F8              69          LDA   #$F8          ;from F800-FFFF
C344:85 37              70          STA   CSWH
C346:64 36              71          STZ   CSWL
C348:B2 36              72 COPYROM2 LDA   (CSWL)        ;get a byte
C34A:92 36              73          STA   (CSWL)        ;and save a byte
C34C:E6 36              74          INC   CSWL
C34E:D0 F8   C348       75          BNE   COPYROM2
C350:E6 37              76          INC   CSWH
C352:D0 F4   C348       77          BNE   COPYROM2      ;fall into RESETLC
C354:                   78 *
C354:                   79 * RESETLC resets the language card to the state
C354:                   80 * determined by SETROM.  It always leaves the card
C354:                   81 * write enabled.
C354:                   82 *
C354:DA                 83 RESETLC  PHX                 ;save X
C355:AE 78 04           84          LDX   ROMSTATE      ;get the state
C358:3C 81 C0           85          BIT   ROMIN,X       ;set bank & ROM/RAM read
C35B:3C 81 C0           86          BIT   ROMIN,X       ;set write enable
C35E:FA                 87          PLX                 ;restore X
C35F:60                 88          RTS
C360:                   89 *
C360:                   90 * SETROM switches in the ROM for reading, the RAM
C360:                   91 * for writing, and it saves the state of the
C360:                   92 * language card.  It does not save the write
C360:                   93 * protect status of the card.
C360:                   94 *
C360:DA                 95 SETROM   PHX                 ;save x
C361:A2 00              96          LDX   #0            ;assume write enable,bank2,ROMRD
C363:2C 11 C0           97          BIT   RDLCBNK2      ;is bank 2 switched in?
C366:30 02   C36A       98          BMI   NOT1          ;=>yes
C368:A2 08              99          LDX   #$8           ;indicate bank 1
C36A:2C 12 C0          100 NOT1     BIT   RDLCRAM       ;is LC RAM readable?
C36D:10 02   C371      101          BPL   NOREAD        ;=>no
C36F:E8                102          INX                 ;indicate RAM read
C370:E8                103          INX
C371:2C 81 C0          104 NOREAD   BIT   $C081         ;ROM read
C374:2C 81 C0          105          BIT   $C081         ;RAM write
C377:8E 78 04          106          STX   ROMSTATE      ;save state
C37A:FA                107          PLX                 ;restore X
C37B:60                108 ROMOK    RTS
C37C:                  109 *
C37C:                  110 * GETALT reads a byte from aux memory screenholes.
C37C:                  111 * Y is the index to the byte (0-7) indexed off of
C37C:                  112 * address $478.
C37C:                  113 *
C37C:AD 13 C0          114 GETALT   LDA   RDRAMRD       ;save state of aux memory
C37F:0A                115          ASL   A
C380:AD 18 C0          116          LDA   RD80COL       ;and of the 80STORE switch
C383:08                117          PHP
```



```
C384:8D 00 C0          118          STA   CLR80COL      ;no 80STORE to get page 1
C387:8D 03 C0          119          STA   RDCARDRAM     ;pop in the other half of RAM
C38A:B9 78 04          120          LDA   $478,Y        ;read the desired byte
C38D:28                121          PLP                 ;and restore memory
C38E:B0 03   C393      122          BCS   GETALT1
C390:8D 02 C0          123          STA   RDMAINRAM
C393:10 03   C398      124 GETALT1  BPL   GETALT2
C395:8D 01 C0          125          STA   SET80COL
C398:60                126 GETALT2  RTS
C399:                  127 *
C399:09 80             128 UPSHIFT0 ORA   #$80          ;set high bit for execs
C39B:C9 FB             129 UPSHIFT  CMP   #$FB
C39D:B0 06   C3A5      130          BCS   X.UPSHIFT
C39F:C9 E1             131          CMP   #$E1
C3A1:90 02   C3A5      132          BCC   X.UPSHIFT
C3A3:29 DF             133          AND   #$DF
C3A5:60                134 X.UPSHIFT RTS
C3A6:                  135 *
C3A6:                  136 * GETCOUT performs COUT for GETLN.  It disables the
C3A6:                  137 * echoing of control characters by clearing the
C3A6:                  138 * M.CTL mode bit, prints the char, then restores
C3A6:                  139 * M.CTL.  NOESC is used by the RDKEY routine to
C3A6:                  140 * disable escape sequences.
C3A6:                  141 *
C3A6:48                142 GETCOUT  PHA                 ;save char to print
C3A7:A9 08             143          LDA   #M.CTL        ;disable control chars
C3A9:1C FB 04          144          TRB   VMODE         ;by clearing M.CTL
C3AC:68                145          PLA                 ;restore character
C3AD:20 ED FD          146          JSR   COUT          ;and print it
C3B0:4C 44 FD          147          JMP   NOESCAPE      ;enable control chars
C3B3:                  148 *
C3B3:                  149 * STORCH determines loads the current cursor position,
C3B3:                  150 *        inverts the character, and displays it
C3B3:                  151 * STORCHAR inverts the character and displays it at the
C3B3:                  152 *       position stored in Y
C3B3:                  153 * STORY  determines the current cursor position, and
C3B3:                  154 *       displays the character without inverting it
C3B3:                  155 * STORE  displays the char at the position in Y
C3B3:                  156 *
C3B3:                  157 * If mouse characters are enabled (VMODE bit 0 = 0)
C3B3:                  158 * then mouse characters ($40-$5F) are displayed when
C3B3:                  159 * the alternate character set is switched in.  Normally
C3B3:                  160 * values $40-$5F are shifted to $0-$1F before display.
C3B3:                  161 *
C3B3:                  162 * Calls to GETCUR trash Y
C3B3:                  163 *
C3B3:20 9D CC          164 STORY    JSR   GETCUR        ;get newest cursor into Y
C3B6:80 09   C3C1      165          BRA   STORE
C3B8:                  166 *
C3B8:20 9D CC          167 STORCH   JSR   GETCUR        ;first, get cursor position
C3BB:24 32             168          BIT   INVFLG        ;normal or inverse?
C3BD:30 02   C3C1      169          BMI   STORE         ;=>normal, store it
C3BF:29 7F             170          AND   #$7F          ;inverse it
C3C1:5A                171 STORE    PHY                 ;save real Y
C3C2:09 00             172          ORA   #0            ;does char have high bit set?
C3C4:30 15   C3DB      173          BMI   STORE1        ;=>yes, don't do mouse check
C3C6:48                174          PHA                 ;save char
C3C7:AD FB 04          175          LDA   VMODE         ;is mouse bit set?
```

```
07 C3SPACE        Communications port routine        20-OCT-86  06:41 PAGE 19

C3CA:6A             176           ROR   A
C3CB:68             177           PLA                 ;restore char
C3CC:90 0D   C3DB  178           BCC   STORE1        ;=>no, don't do mouse shift
C3CE:2C 1E C0      179           BIT   ALTCHARSET    ;no shift if ][ char set
C3D1:10 08   C3DB  180           BPL   STORE1        ;=> it is!
C3D3:49 40         181           EOR   #$40          ;$40-$5F=>0-$1f
C3D5:89 60         182           BIT   #$60
C3D7:F0 02   C3DB  183           BEQ   STORE1
C3D9:49 40         184           EOR   #$40
C3DB:2C 1F C0      185 STORE1    BIT   RD80VID       ;80 columns?
C3DE:10 19   C3F9  186           BPL   STORE5        ;=>no, store char
C3E0:48            187           PHA                 ;save (shifted) char
C3E1:8D 01 C0      188           STA   SET80COL      ;hit 80 store
C3E4:98            189           TYA                 ;get proper Y
C3E5:45 20         190           EOR   WNDLFT        C=1 if char in main ram
C3E7:4A            191           LSR   A
C3E8:B0 04   C3EE  192           BCS   STORE2        ;=>yes, main RAM
C3EA:AD 55 C0      193           LDA   TXTPAGE2      ;else flip in aux RAM
C3ED:C8            194           INY                 ;do this for odd left, aux bytes
C3EE:98            195 STORE2    TYA                 ;divide pos'n by 2
C3EF:4A            196           LSR   A
C3F0:A8            197           TAY
C3F1:68            198           PLA                 ;get (shifted) char
C3F2:91 28         199 STORE3    STA   (BASL),Y      ;stuff it
C3F4:2C 54 C0      200           BIT   TXTPAGE1      ;else restore page1
C3F7:7A            201 STORE4    PLY                 ;restore real Y
C3F8:60            202           RTS                 ;und exit
C3F9:              203 *
C3F9:91 28         204 STORE5    STA   (BASL),Y      ;do 40 column store
C3FB:7A            205           PLY                 ;restore Y
C3FC:60            206           RTS                 ;and exit
C3FD:       0003   207           DS    $C400-*,$00
C400:               45           include equates2    ;Equates for PC and Slinky
```

```
08 EQUATES2       slinky equates                     20-OCT-86  06:41 PAGE 20

C400:               2 ************************************************
C400:               3 * slinky equates
C400:               4 ************************************************
C400:       0101    5 revnum    equ   $101           ;revision 1.0.1
C400:       0011    6 pcrevnum  equ   $11            ;smartport rev 1.1

C400:               8 * prodos equates

C400:       0042   10 cmmand    equ   $42            ;command to be executed
C400:       0043   11 unit      equ   $43            ;D<6-4> = slot
C400:       0044   12 buffer    equ   $44            ;pointer to 512 byte data buffer
C400:       0046   13 block     equ   $46            ;block number

C400:              15 * protocol converter equates

C400:       0043   17 pparm     equ   $43            ;parameter count
C400:       0044   18 punit     equ   $44            ;unit number
C400:       0045   19 pbuff     equ   $45            ;two byte buffer pointer
C400:       0047   20 pstat     equ   $47            ;status / control code
C400:       0047   21 pblock    equ   $47            ;block number
C400:       0047   22 pcount    equ   $47            ;byte count
C400:       0049   23 paddr     equ   $49            ;address for read
C400:       004A   24 tempptr   equ   $4A            ;pointer to params must be last 2 zp byte
C400:       000A   25 zused     equ   tempptr-cmmand+2 ;zero page bytes used

C400:              27 * prodos commands

C400:       0000   29 prostat   equ   0             ;status command
C400:       0001   30 proread   equ   1             ;read command
C400:       0002   31 prowrit   equ   2             ;write command
C400:       0003   32 proform   equ   3             ;format command

C400:              34 * DOS equates

C400:       0048   36 iobpl     equ   $48            ;pointer to IOB
C400:       0049   37 iobph     equ   $49
C400:       0001   38 ibslot    equ   1              ;slot * 16
C400:       0002   39 ibdrvn    equ   2              ;drive 1 or 2
C400:       0004   40 ibtrk     equ   4              ;track number
C400:       0005   41 ibsect    equ   5              ;sector number
C400:       0008   42 ibbufp    equ   8              ;buffer pointer
C400:       000C   43 ibcmd     equ   12             ;command
C400:       000D   44 ibstat    equ   13             ;status
C400:       0080   45 doserr    equ   $80            ;DOS I/O error
C400:       9D1E   46 dosinit   equ   $9D1E          ;DOS init vector use addr-1
C400:       A6C3   47 dossyn    equ   $A6C4-1        ;DOS syntax error
C400:       BD00   48 rwts      equ   $BD00          ;RWTS entry point

C400:              50 * error codes

C400:       0001   52 badcmd    equ   $01            ;bad command
C400:       0004   53 badpcnt   equ   $04            ;bad parameter count
C400:       0011   54 badunit   equ   $11            ;bad unit number
C400:       0021   55 badctl    equ   $21            ;bad control / status code
C400:       0027   56 ioerr     equ   $27            ;other I/O error
C400:       0028   57 nderr     equ   $28            ;no device error
C400:       002D   58 badblk    equ   $2D            ;bad block or address
```

```
C400:                    60 * prodos boot equates

C400:       0002         62 bootjmp   equ  2          ;make jmp to entry
C400:       0000         63 bootblk   equ  0          ;reads block 0
C400:       0800         64 bootbuf   equ  $800       ;into $800
C400:       FABA         65 autoscan  equ  $FABA      ;re-entry point to auto boot
C400:       FF59         66 monitor   equ  $FF59      ;go to monitor if boot fails

C400:                    68 * scratch area equates

C400:       0478         70 sizetemp  equ  $478       ;holds # blocks
C400:       04F8         71 error     equ  $4F8       ;error flag
C400:       0578         72 xval      equ  $578       ;value to be returned in X
C400:       05F8         73 yval      equ  $5F8       ;value to be returned in Y
C400:       0678         74 sl.lcstate equ $678       ;language card state
C400:       0778         75 sl.devno  equ  $778       ;slot * 16 ($n0) + $88
C400:       07F8         76 sl.mslot  equ  $7F8       ;$C0 + slot ($Cn)

C400:                    78 * slot ram equates

C400:       03B8         80 sl.scrn1  equ  $478-$C0
C400:       0438         81 sl.scrn2  equ  $4F8-$C0
C400:       04B8         82 sl.scrn3  equ  $578-$C0
C400:       0538         83 sl.scrn4  equ  $5F8-$C0
C400:       05B8         84 sl.scrn5  equ  $678-$C0
C400:       0638         85 sl.scrn6  equ  $6F8-$C0
C400:       06B8         86 sl.scrn7  equ  $778-$C0
C400:       0738         87 sl.scrn8  equ  $7F8-$C0
C400:       03B8         88 numbanks  equ  sl.scrn1   ;number of 64K banks on card
C400:       06B8         89 powerup   equ  sl.scrn7   ;powerup byte
C400:                    90 ;power2 equ sl.scrn8

C400:                    92 * hardware equates, must be in $BF00 to avoid double access

C400:       BFF8         94 addrl     equ  $BFF8      ;address pointer
C400:       BFF9         95 addrm     equ  $BFF9      ;auto incs after every data access
C400:       BFFA         96 addrh     equ  $BFFA
C400:       BFFB         97 data      equ  $BFFB      ;data pointed to

C400:                    99 * other interface equates

C400:       BF00        101 proflag   equ  $BF00      ;0 = Pascal, $4C = ProDOS, other = DOS
C400:       00AA        102 nameflg   equ  $AA        ;value unused in any catalog
C400:       00FC        103 sizeflg   equ  $FC        ;block size flag
C400:       00FD        104 zers      equ  $FD        ;catalog skip flag
C400:       00FE        105 skpfe     equ  $FE        ;skip FE bytes in catalog
C400:       0004        106 slot      equ  $04        ;slot #
C400:       DC00        107 diagcode  equ  $DC00      ;location of diagnostic code
C400:       2000        108 diagdest  equ  $2000      ;location of diagnostics in ram
C400:       1FFF        109 diagstart equ  diagdest-1 ;start location of diagnostics
C400:                    46           include slinky  ;ram card at $C400
```



```
C400:                     2 ************************************************
C400:                     3 * slinky boot code
C400:                     4 ************************************************

C400:C9 20                6           cmp  #$20        ;Boot entry point
C402:C9 00                7           cmp  #$00        ;Signature byte stuff
C404:C9 03                8           cmp  #$03
C406:C9 00                9           cmp  #0
C408:B0 04   C40E        10           bcs  boot4       ;Always taken
C40A:A0 05               11           ldy  #5          ;Diagnostics entry point
C40C:D0 54   C462        12           bne  dos24
C40E:                    13 * Here is the boot code
C40E:                    14 * Reads in block 0 into $800 and executes at $801
C40E:        C40E        15 boot4     equ  *
C40E:78                  16           sei              ;No interrupts if booting
C40F:A5 39               17           lda  kswh        ;Save IN#

C411:                    19 ************************************************
C411:                    20 * the following two bytes must be $90 and $4B in locations $C411 and
C411:                    21 * and $C412 respectively.  the bcc ($90) is never taken by the
C411:                    22 * slinky code and the $4B is used to duplicate the mouse entry
C411:                    23 * point as found in slot 7.  this 'fix' enables some programs
C411:                    24 * to still work correctly.  (tim, you owe me a beer for this one!)
C411:                    25 ************************************************

C411:90 4B               27           dfb  $90,$4B     ;bcc is never taken

C413:48                  29           pha
C414:20 89 FE            30           jsr  setkbd      ;Reset the hooks
C417:20 93 FE            31           jsr  setvid
C41A:68                  32           pla
C41B:20 4C C7            33           jsr  slboot      ;Go get boot block
C41E:AE 01 08            34           ldx  bootbuf+1
C421:F0 05   C428        35           beq  btok4       ;boot not okay
C423:A2 40               36           ldx  #4*$10      ;X=n0
C425:4C 01 08            37           jmp  bootbuf+1

C428:                    39 * discontinue boot sequence if not power on reset or forced cold start

C428:A5 00               41 btok4     lda  loc0
C42A:D0 0D   C439        42           bne  btok4.1
C42C:A5 01               43           lda  loc1
C42E:C9 C4               44           cmp  #4+$C0
C430:D0 07   C439        45           bne  btok4.1
C432:A9 C6               46           lda  #$c6        ;should be $C6 now
C434:85 01               47           sta  loc1
C436:6C 00 00            48           jmp  (loc0)      ;try slot 6 instead

C439:A9 17               50 btok4.1   lda  #23          ;go to bottom of screen
C43B:85 25               51           sta  cv

C43D:BD DB C1            53 btok4.2   lda  utsmsg,x    ;'unable to start from memory card.'
C440:F0 06   C448        54           beq  btok4.3     ;skip if done
C442:20 ED FD            55           jsr  cout
C445:E8                  56           inx
C446:D0 F5   C43D        57           bne  btok4.2

C448:4C 00 E0            59 btok4.3   jmp  basic       ;drop into basic
```

```
C44B:4C 1C C7    61          jmp   xsetmou     ;should be at $C44B
```



```
C44E:                63 ************************************************
C44E:                64 * Entry point for prodos driver
C44E:                65 ************************************************

C44E:4C 54 C4        67 entry4    jmp   ent4         ;Jump to ProDOS
C451:4C 94 C4        68           jmp   pconv4       ;Jump to PC
C454:A9 28           69 ent4      lda   #nderr       ;Assume wrong drive
C456:A6 43           70           ldx   unit
C458:30 2A   C484    71           bmi   rats4        ;Error!!!
C45A:A9 01           72           lda   #badcmd      ;Assume bad command
C45C:A4 42           73           ldy   cmmand       ;Get command

C45E:                75 * the 'cpy #4' below use to be a 'cmp #4'.  because of this
C45E:                76 * change, revnum will be 1.0.1

C45E:C0 04           78           cpy   #4
C460:B0 22   C484    79           bge   rats4        ;Branch if bad command
C462:A2 0A           80 dos24     ldx   #zused       ;Save zp DOS jumps in here
C464:B5 41           81 zsave4    lda   cmmand-1,x
C466:48              82           pha
C467:CA              83           dex
C468:D0 FA   C464    84           bne   zsave4
C46A:98              85           tya                ;Add 20 to command for table look up
C46B:18              86           clc
C46C:69 14           87           adc   #20

C46E:                89 ************************************************
C46E:                90 * This code is common to PC and ProDOS and DOS entry points
C46E:                91 ************************************************

C46E:20 52 C7        93 doit4     jsr   slxeq        ;Go do command
C471:A2 00           94 done4     ldx   #0           ;Restore zp
C473:68              95 rsloop4   pla
C474:95 42           96           sta   cmmand,x
C476:E8              97           inx
C477:E0 0A           98           cpx   #zused
C479:90 F8   C473    99           blt   rsloop4
C47B:AE 78 05       100           ldx   xval         ;Set X and Y to whatever
C47E:AC F8 05       101           ldy   yval
C481:AD F8 04       102           lda   error        ;Was there an error?
C484:C9 01          103 rats4     cmp   #1           ;C = 1 if not 0
C486:09 00          104           ora   #0           ;Set n,z flags
C488:60             105           rts

C489:A9 01          107 pccmd4    lda   #badcmd      ;Bad command
C48B:D0 02   C48F   108           bne   pcerr4
C48D:A9 11          109 pcbad4    lda   #badunit     ;Bad protocol unit
C48F:8D F8 04       110 pcerr4    sta   error        ;Save the error code
C492:D0 DD   C471   111           bne   done4
```

```
C494:                  113 ************************************************
C494:                  114 * Protocol converter entry point
C494:                  115 ************************************************

C494:68                117 pconv4   pla                 ;Pull the return address
C495:A8                118          tay
C496:C9 FD             119          cmp   #$FD          ;C = 1 if carry in +3
C498:68                120          pla                  ;High byte of return address
C499:AA                121          tax
C49A:69 00             122          adc   #0
C49C:48                123          pha
C49D:98                124          tya
C49E:69 03             125          adc   #3            ;C = 0 from previous add
C4A0:48                126          pha                  ;Return address + 3 now pushed
C4A1:A5 4B             127          lda   tempptr+1     ;Save zero page
C4A3:48                128          pha
C4A4:86 4B             129          stx   tempptr+1
C4A6:A2 09             130          ldx   #zused-1
C4A8:B5 41             131 pcsvzp4  lda   cmmand-1,x    ;Save rest of zp
C4AA:48                132          pha
C4AB:CA                133          dex
C4AC:D0 FA   C4A8      134          bne   pcsvzp4
C4AE:84 4A             135          sty   tempptr
C4B0:A0 03             136          ldy   #3            ;Get (tempptr) + 3, +2, +1
C4B2:99 41 00          137 pcgtp4   sta   cmmand-1,y    ;First store don't care
C4B5:B1 4A             138 pcskp4   lda   (tempptr),y
C4B7:88                139          dey
C4B8:D0 F8   C4B2      140          bne   pcgtp4
C4BA:AA                141          tax                  ;x = user command
C4BB:                  142 ;                             Command = ptr to parm list.  y = 0
C4BB:A0 08             143          ldy   #8            ;Max # of param bytes
C4BD:B1 42             144 pcparms4 lda   (cmmand),y    ;Get the parms
C4BF:99 43 00          145          sta   cmmand+1,y    ;Note last store is on top of pointer
C4C2:88                146          dey
C4C3:10 F8   C4BD      147          bpl   pcparms4
C4C5:46 44             148          lsr   punit         ;Unit 0 or 1?
C4C7:D0 C4   C48D      149          bne   pcbad4
C4C9:8A                150          txa                  ;A = Command
C4CA:2A                151          rol   A             ;Get unit into low bit
C4CB:C9 14             152          cmp   #20           ;C = 1 if invalid command
C4CD:B0 BA   C489      153          bcs   pccmd4        ;Branch if bad command
C4CF:90 9D   C46E      154          bcc   doit4         ;Always taken
```



```
C4D1:                  156 ************************************************
C4D1:                  157 * DOS entry point stuff
C4D1:                  158 ************************************************

C4D1:84 48             160 dosent4  sty   iobpl         ;Store pointer to IOB
C4D3:85 49             161          sta   iobph
C4D5:A0 01             162          ldy   #ibslot       ;Get slot
C4D7:B1 48             163          lda   (iobpl),y
C4D9:C9 40             164          cmp   #4*$10        ;Is it us?
C4DB:F0 03   C4E0      165          beq   dosslt4
C4DD:4C 04 BD          166          jmp   rwts+4        ;Back to rwts
C4E0:A0 04             167 dosslt4  ldy   #4            ;Command for DOS
C4E2:20 62 C4          168          jsr   dos24         ;Go do it and come back
C4E5:F0 02   C4E9      169          beq   dosok4        ;A = Error code
C4E7:A9 80             170          lda   #doserr       ;DOS io error code
C4E9:A0 0D             171 dosok4   ldy   #ibstat
C4EB:91 48             172          sta   (iobpl),y     ;Put error code in status
C4ED:60                173          rts

C4EE:        000C      175          ds    $C4FA-*,0     ;pad with 0's

C4FA:11                177          dfb   pcrevnum      ;Smartport revision number
C4FB:01                178          dfb   $01           ;Mark ram card
C4FC:00 00             179          dw    0             ;Number of blocks = 0 for status call
C4FE:4F                180          dfb   $4F
C4FF:4E                181          dfb   >entry4
C500:                   47          include misc        ;Miscellaneous junk
C500:        008E        1          ds    $C58E-*,0
```

```
10 MISC           slinky entry points              20-OCT-86  06:41 PAGE 27

C58E:              3 ************************************************
C58E:              4 * maktbl - makes a deniblizing table for the disk II boot
C58E:              5 ************************************************

C58E:A2 03         7 MAKTBL    LDX   #$03
C590:A0 00         8           LDY   #0
C592:86 3C         9 TBLLOOP   STX   BOOTTMP
C594:8A           10           TXA
C595:0A           11           ASL   A
C596:24 3C        12           BIT   BOOTTMP
C598:F0 10   C5AA 13           BEQ   NOPATRN
C59A:05 3C        14           ORA   BOOTTMP
C59C:49 FF        15           EOR   #$FF
C59E:29 7E        16           AND   #$7E
C5A0:B0 08   C5AA 17 TBLLOOP2 BCS   NOPATRN
C5A2:4A           18           LSR   A
C5A3:D0 FB   C5A0 19           BNE   TBLLOOP2
C5A5:98           20           TYA
C5A6:9D 56 03     21           STA   DNIBL,X
C5A9:C8           22           INY
C5AA:E8           23 NOPATRN   INX
C5AB:10 E5   C592 24           BPL   TBLLOOP
C5AD:A9 08        25           LDA   #$08
C5AF:85 27        26           STA   $27
C5B1:A0 7F        27           LDY   #$7F
C5B3:60           28           RTS

C5B4:             30 ************************************************
C5B4:             31 * getup - get char from input buffer, iny and upshift it
C5B4:             32 ************************************************

C5B4:B9 00 02     34 getup     lda   in,y          ;get character
C5B7:C8           35           iny
C5B8:4C 99 C3     36           jmp   upshift0

C5BB:             38 ************************************************
C5BB:             39 * this is who we is in 9 letters
C5BB:             40 ************************************************

C5BB:             42           MSB   ON
C5BB:C1 F0 F0 EC  43 apple2c   asc   'Apple        //c'
C5C4:             44           MSB   OFF

C5C4:             46 ************************************************
C5C4:             47 * showinst - disassemble an instruction and adjust the PC
C5C4:             48 ************************************************

C5C4:20 D0 F8     50 showinst  jsr   instdsp
C5C7:20 53 F9     51           jsr   pcadj
C5CA:85 3A        52           sta   pcl
C5CC:84 3B        53           sty   pch
C5CE:60           54           rts

C5CF:             56 ************************************************
C5CF:             57 * xmbasic - basic call to the mouse
C5CF:             58 ************************************************

C5CF:5A           60 xmbasic   phy
```

```
10 MISC           slinky entry points              20-OCT-86  06:41 PAGE 28

C5D0:B0 1C   C5EE 61           bcs   gobasicin     ;input?
C5D2:A0 C7        62           ldy   #<mbasic      ;input from $C400?
C5D4:C4 39        63           cpy   kswh
C5D6:D0 04   C5DC 64           bne   xmbout
C5D8:A4 38        65           ldy   kswl
C5DA:F0 12   C5EE 66           beq   gobasicin
C5DC:DA           67 xmbout    phx                  ;save X too
C5DD:48           68           pha
C5DE:29 7F        69           and   #$7F           ;we don't care about high bit
C5E0:C9 02        70           cmp   #2
C5E2:B0 06   C5EA 71           bge   mbbad          ;only 0,1 valid
C5E4:20 1C C7     72           jsr   xsetmou
C5E7:20 3A C7     73           jsr   xmhome
C5EA:68           74 mbbad     pla
C5EB:FA           75           plx
C5EC:7A           76           ply
C5ED:60           77           rts
C5EE:4C 9D C7     78 gobasicin jmp   swbasicin     ;go to input routine

C5F1:       0004  80           ds    $C5F5-*,0      ;more disk stuff
C5F5:             48           include boot         ;Disk II boot @$C600
C5F5:       000B   1           ds    $C600-*,0      ;Disk II in slot 6
```

```
C600:                    4 ****************************************
C600:                    5 *
C600:                    6 * Disk II boot stuff
C600:                    7 * jumps to slot 5 if boot fails
C600:                    8 *
C600:                    9 ****************************************
C600:A2 20              10          LDX   #$20
C602:A0 00              11          LDY   #$00
C604:64 03              12          STZ   $03
C606:64 3C              13          STZ   $3C
C608:A9 60              14          LDA   #$60
C60A:AA                 15          TAX
C60B:86 2B              16 DRV2ENT  STX   SLOTZ
C60D:85 4F              17          STA   BOOTDEV
C60F:5A                 18          PHY                 ;Y=1 IF DRIVE 2 BOOT, ELSE Y=0
C610:BD 8E C0           19          LDA   $C08E,X
C613:BD 8C C0           20          LDA   $C08C,X
C616:7A                 21          PLY
C617:B9 EA C0           22          LDA   $C0EA,Y       ;SELECT DRIVE 1 OR 2
C61A:BD 89 C0           23          LDA   $C089,X
C61D:A0 50              24          LDY   #$50
C61F:BD 80 C0           25 SEEKZERO LDA   $C080,X
C622:98                 26          TYA
C623:29 03              27          AND   #$03
C625:0A                 28          ASL   A
C626:05 2B              29          ORA   SLOTZ
C628:AA                 30          TAX
C629:BD 81 C0           31          LDA   $C081,X
C62C:A9 56              32          LDA   #$56
C62E:20 A8 FC           33          JSR   WAIT
C631:88                 34          DEY
C632:10 EB   C61F       35          BPL   SEEKZERO
C634:85 26              36          STA   $26
C636:85 3D              37          STA   $3D
C638:85 41              38          STA   $41
C63A:20 8E C5           39          jsr   maktbl
C63D:64 03              40 EXTENT1  STZ   $03
C63F:18                 41 RDADR    CLC
C640:08                 42          PHP
C641:28                 43 RETRY1   PLP
C642:A6 2B              44 RDDHDR   LDX   SLOTZ          ;RESTORE X TO $60
C644:C6 03              45          DEC   $03           ;UPDATE RETRY COUNT
C646:D0 0E   C656       46          BNE   RDHD0         ;BRANCH IF NOT OUT OF RETRIES
C648:BD 88 C0           47 FUGIT    LDA   $C088,X       ;SHUT OFF DISK AND QUIT!
C64B:A5 01              48          lda   loc1          ;auto boot form slot 6?
C64D:C9 C6              49          cmp   #$c6
C64F:D0 A4   C5F5       50          bne   bootfail
C651:4C 00 C5           51          jmp   $c500         ;maybe slot 5 will talk to us

C654:        0002       53          ds    $C656-*,0     ;Keep alignment
C656:08                 54 RDHD0    PHP
C657:88                 55 RETRY    DEY
C658:D0 04   C65E       56          BNE   RDHD1
C65A:F0 E5   C641       57          BEQ   RETRY1
C65C:80 DF   C63D       58 EXTENT   BRA   EXTENT1       ;Blows up if this is moved too
C65E:                   59 * * * * * * * * * * * * * * * * * * * * * * *
C65E:                   60 * The following code is sacred in it's  *
C65E:                   61 * present form.  To change it would     *
```



```
C65E:                   62 * cause volcanos to errupt, the ground  *
C65E:                   63 * to shake, and ProDOS not to boot!     *
C65E:                   64 * * * * * * * * * * * * * * * * * * * * * * *
C65E:BD 8C C0           65 RDHD1    LDA   $C08C,X
C661:10 FB   C65E       66          BPL   RDHD1
C663:49 D5              67 ISMRK1   EOR   #$D5
C665:D0 F0   C657       68          BNE   RETRY
C667:BD 8C C0           69 RDHD2    LDA   $C08C,X
C66A:10 FB   C667       70          BPL   RDHD2
C66C:C9 AA              71          CMP   #$AA
C66E:D0 F3   C663       72          BNE   ISMRK1
C670:EA                 73          NOP
C671:BD 8C C0           74 RDHD3    LDA   $C08C,X
C674:10 FB   C671       75          BPL   RDHD3
C676:C9 96              76          CMP   #$96
C678:F0 09   C683       77          BEQ   RDSECT
C67A:28                 78          PLP
C67B:90 C2   C63F       79          BCC   RDADR
C67D:49 AD              80          EOR   #$AD
C67F:F0 25   C6A6       81          BEQ   RDATA
C681:D0 BC   C63F       82          BNE   RDADR
C683:A0 03              83 RDSECT   LDY   #$03
C685:85 40              84 RDSEC1   STA   $40
C687:BD 8C C0           85 RDSEC2   LDA   $C08C,X
C68A:10 FB   C687       86          BPL   RDSEC2
C68C:2A                 87          ROL   A
C68D:85 3C              88          STA   BOOTTMP
C68F:BD 8C C0           89 RDSEC3   LDA   $C08C,X
C692:10 FB   C68F       90          BPL   RDSEC3
C694:25 3C              91          AND   BOOTTMP
C696:88                 92          DEY
C697:D0 EC   C685       93          BNE   RDSEC1
C699:28                 94          PLP
C69A:C5 3D              95          CMP   $3D
C69C:D0 A1   C63F       96          BNE   RDADR
C69E:A5 40              97          LDA   $40
C6A0:C5 41              98          CMP   $41
C6A2:D0 9B   C63F       99 BADRD1   BNE   RDADR
C6A4:B0 9C   C642      100          BCS   RDDHDR
C6A6:A0 56             101 RDATA    LDY   #$56
C6A8:84 3C             102 RDAT0    STY   BOOTTMP
C6AA:BC 8C C0          103 RDAT1    LDY   $C08C,X
C6AD:10 FB   C6AA      104          BPL   RDAT1
C6AF:59 D6 02          105          EOR   DNIBL-$80,Y
C6B2:A4 3C             106          LDY   BOOTTMP
C6B4:88                107          DEY
C6B5:99 00 03          108          STA   NBUF1,Y
C6B8:D0 EE   C6A8      109          BNE   RDAT0
C6BA:84 3C             110 RDAT2    STY   BOOTTMP
C6BC:BC 8C C0          111 RDAT3    LDY   $C08C,X
C6BF:10 FB   C6BC      112          BPL   RDAT3
C6C1:59 D6 02          113          EOR   DNIBL-$80,Y
C6C4:A4 3C             114          LDY   BOOTTMP
C6C6:91 26             115          STA   ($26),Y
C6C8:C8                116          INY
C6C9:D0 EF   C6BA      117          BNE   RDAT2
C6CB:BC 8C C0          118 RDAT4    LDY   $C08C,X
C6CE:10 FB   C6CB      119          BPL   RDAT4
```

```
11 BOOT          Disk II boot code                20-OCT-86  06:41 PAGE 31

C6D0:59 D6 02     120           EOR   DNIBL-$80,Y
C6D3:D0 CD  C6A2  121 BADREAD   BNE   BADRD1
C6D5:A0 00        122           LDY   #$00
C6D7:A2 56        123 DENIBL    LDX   #$56
C6D9:CA           124 DENIB1    DEX
C6DA:30 FB  C6D7  125           BMI   DENIBL
C6DC:B1 26        126           LDA   ($26),Y
C6DE:5E 00 03     127           LSR   NBUF1,X
C6E1:2A           128           ROL   A
C6E2:5E 00 03     129           LSR   NBUF1,X
C6E5:2A           130           ROL   A
C6E6:91 26        131           STA   ($26),Y
C6E8:C8           132           INY
C6E9:D0 EE  C6D9  133           BNE   DENIB1
C6EB:             134 * * * * * * * * * * * * * * * * * *
C6EB:             135 * Code beyond this point is not *
C6EB:             136 * sacred... It may be perverted *
C6EB:             137 * in any manner by any pervert. *
C6EB:             138 * * * * * * * * * * * * * * * * * *
C6EB:E6 27        139           INC   $27
C6ED:E6 3D        140           INC   $3D
C6EF:A5 3D        141           LDA   $3D
C6F1:CD 00 08     142           CMP   $0800
C6F4:A6 4F        143           LDX   BOOTDEV
C6F6:90 DB  C6D3  144           BCC   BADREAD
C6F8:4C 01 08     145           JMP   $0801
C6FB:       0005  146           DS    $C700-*,0     ;Last byte must be 0
C700:              49           include mouse       ;Equates for the mouse
```

```
12 MOUSE         Mouse firmware                   20-OCT-86  06:41 PAGE 32

C700:              2          msb   ON
C700:              3 ****************************************
C700:              4 *
C700:              5 * Mouse firmware for the Chels
C700:              6 *
C700:              7 *  by Rich Williams
C700:              8 *  July, 1983
C700:              9 *
C700:             10 ****************************************



C700:             12 ****************************************
C700:             13 *
C700:             14 * Equates
C700:             15 *
C700:             16 ****************************************



C700:             18 * Input bounds are in scratch area
C700:       0478  19 moutemp   equ   $478          ;Temporary storage
C700:       0478  20 minl      equ   $478
C700:       04F8  21 maxl      equ   $4F8
C700:       0578  22 minh      equ   $578
C700:       05F8  23 maxh      equ   $5F8
C700:             24 * Mouse bounds in slot 5 screen area
C700:       047D  25 minxl     equ   $47D
C700:       04FD  26 minyl     equ   $4FD
C700:       057D  27 minxh     equ   $57D
C700:       05FD  28 minyh     equ   $5FD
C700:       067D  29 maxxl     equ   $67D
C700:       06FD  30 maxyl     equ   $6FD
C700:       077D  31 maxxh     equ   $77D
C700:       07FD  32 maxyh     equ   $7FD
C700:             33 * Mouse holes in slot 7 screen area
C700:       047F  34 mouxl     equ   $47F          ;X position low byte
C700:       04FF  35 mouyl     equ   $4FF          ;Y position low byte
C700:       057F  36 mouxh     equ   $57F          ;X position high byte
C700:       05FF  37 mouyh     equ   $5FF          ;Y position high byte
C700:       067F  38 mouarm    equ   $67F          ;Arm interrupts from movement or button
C700:       077F  39 moustat   equ   $77F          ;Mouse status
C700:             40 * Moustat provides the following
C700:             41 *  D7= Button pressed
C700:             42 *  D6= Status of button on last read
C700:             43 *  D5= Moved since last read
C700:             44 *  D4= Reserved
C700:             45 *  D3= Interrupt from VBL
C700:             46 *  D2= Interrupt from button
C700:             47 *  D1= Interrupt from movement
C700:             48 *  D0= Reserved
C700:       07FF  49 moumode   equ   $7FF          ;Mouse mode
C700:             50 *  D7 = 1 if user wants control of mouse interrupts
C700:             51 *  D6-D4= Unused
C700:             52 *  D3= VBL active
C700:             53 *  D2= VBL interrupt on button
C700:             54 *  D1= VBL interrupt on movement
C700:             55 *  D0= Mouse active
```

```
C700:          0020   56 movarm    equ   $20
C700:          000C   57 vblmode   equ   $0C
C700:          0004   58 butmode   equ   $04          ;D2 mask
C700:          0002   59 movmode   equ   $02          ;D1 mask



C700:                 61 * Hardware addresses
C700:          C015   62 mouxint   equ   $C015        ;D7 = x interrupt
C700:          C017   63 mouyint   equ   $C017        ;D7 = y interrupt
C700:          C019   64 vblint    equ   $C019        ;D7 = vbl interrupt
C700:          C078   65 ioudsbl   equ   $C078        ;Disable iou access
C700:          C079   66 iouenbl   equ   $C079        ;Enable iou access
C700:          C048   67 mouclr    equ   $C048        ;Clear mouse interrupt
C700:          C058   68 iou       equ   $C058        ;IOU interrupt switches
C700:          C058   69 moudsbl   equ   $C058        ;Disable mouse interrupts
C700:          C059   70 mouenbl   equ   $C059        ;Enable mouse interrupts
C700:          C063   71 moubut    equ   $C063        ;D7 = Mouse button
C700:          C066   72 mouxl     equ   $C066        ;D7 = X1
C700:          C067   73 mouyl     equ   $C067        ;D7 = Y1
C700:          C070   74 vblclr    equ   $C070        ;Clear VBL interrupt
C700:                 75 *
C700:                 76 * Other addresses
C700:                 77 *
C700:          0200   78 inbuf     equ   $200         ;Input buffer
C700:          0214   79 binl      equ   inbuf+20     ;Temp for binary conversion
C700:          0215   80 binh      equ   inbuf+21
C700:                 50          include mcode.x
```



```
C700:                  2 ****************************************
C700:                  3 *
C700:                  4 * Entry points for mouse firmware
C700:                  5 *
C700:                  6 ****************************************
C700:80 05   C707      7 mbasic    bra   outent
C702:A2 03             8 pnull     ldx   #3
C704:60                9           rts                  ;Null for pascal entry
C705:38               10 inent     sec                  ;Signature bytes
C706:90               11           dfb   $90
C707:18               12 outent    clc
C708:4C CF C5         13           jmp   xmbasic        ;Go do basic entry
C70B:01               14           dfb   $01            ;More signature stuff
C70C:20               15           dfb   $20
C70D:02               16           dfb   >pnull
C70E:02               17           dfb   >pnull
C70F:02               18           dfb   >pnull
C710:02               19           dfb   >pnull
C711:00               20           dfb   $0
C712:1C               21           dfb   >xsetmou       ;SETMOUSE
C713:22               22           dfb   >xmtstint      ;SERVEMOUSE
C714:28               23           dfb   >xmread        ;READMOUSE
C715:2E               24           dfb   >xmclear       ;CLEARMOUSE
C716:1A               25           dfb   >noerror       ;POSMOUSE
C717:34               26           dfb   >xmclamp       ;CLAMPMOUSE
C718:3A               27           dfb   >xmhome        ;HOMEMOUSE
C719:40               28           dfb   >initmouse     ;INITMOUSE
C71A:                 29 ;dfb >pnull
C71A:                 30 ;dfb >goxmint

C71A:18               32 noerror   clc
C71B:60               33           rts

C71C:8D 28 C0         35 xsetmou   sta   rombank
C71F:4C C2 C6         36           jmp   sw.setmou      ;do the real thing

C722:8D 28 C0         38 xmtstint  sta   rombank
C725:4C CD C6         39           jmp   sw.mtstint     ;do the real thing

C728:8D 28 C0         41 xmread    sta   rombank
C72B:4C D8 C6         42           jmp   sw.mread       ;do the real thing

C72E:8D 28 C0         44 xmclear   sta   rombank
C731:4C E3 C6         45           jmp   sw.mclear      ;do the real thing

C734:8D 28 C0         47 xmclamp   sta   rombank
C737:4C EE C6         48           jmp   sw.mclamp      ;do the real thing

C73A:8D 28 C0         50 xmhome    sta   rombank
C73D:4C F9 C6         51           jmp   sw.mhome       ;do the real thing

C740:8D 28 C0         53 initmouse sta   rombank
C743:4C 04 C7         54           jmp   sw.initmouse   ;do the real thing

C746:8D 28 C0         56           sta   rombank
C749:4C 9A CF         57           jmp   m.oveirq       ;from other side

C74C:8D 28 C0         59 slboot    sta   rombank
```

```
C74F:4C B4 C6      60           jmp   swsl.bt       ;other side

C752:8D 28 C0      62 slxeq     sta   rombank
C755:DA            63           phx                 ;
C756:20 16 C8      64           jsr   getlc         ;
C759:5A            65           phy                 ;
C75A:8C 78 06      66           sty   sl.lcstate    ;
C75D:20 00 D8      67           jsr   execute       ;
C760:4C 0E C8      68           jmp   fixlc         ;

C763:        001D  70           ds    $c780-*,$00
C780:              51           include switcher    ;Bank switcher @ $C780
```



```
C780:        0000  2            ds    $C780-*,0
C780:              3  ****************************************
C780:              4  *
C780:              5  * Code for switching between banks
C780:              6  * This code appears in both banks of the rom
C780:              7  *
C780:              8  ****************************************
C780:8D 28 C0      9 swrti      sta   rombank       ;RTI to the other bank
C783:40           10            rti
C784:8D 28 C0     11 swrts      sta   rombank       ;RTS to the other bank
C787:60           12 swrtsop    rts
C788:8D 28 C0     13 swreset    sta   rombank       ;Reset routine
C78B:4C 62 FA     14            jmp   reset
C78E:8D 28 C0     15            sta   rombank       ;Interrupt routine
C791:2C 87 C7     16            bit   swrtsop       ;Set V = 1 for other bank
C794:4C 04 C8     17            jmp   irqent
C797:8D 28 C0     18 swpcnv     sta   rombank       ;Protocol converter
C79A:4C F1 C7     19            jmp   swsthk3       ;Jump to sethooks from other side
C79D:8D 28 C0     20 swbasicin  sta   rombank       ;Mouse BASIC routines
C7A0:4C F6 C7     21            jmp   swzzqt3       ;Jump to zzquit from other side
C7A3:8D 28 C0     22 swsttm     sta   rombank       ;Set terminal mode
C7A6:4C F1 C7     23            jmp   swsttm3
C7A9:8D 28 C0     24 swcmd      sta   rombank       ;Serial port command processor
C7AC:4C 06 C8     25            jmp   swcmd3
C7AF:8D 28 C0     26 swaux      sta   rombank       ;Moveaux
C7B2:4C 4E C3     27            jmp   moveaux
C7B5:8D 28 C0     28 swxfer     sta   rombank
C7B8:4C 97 C3     29            jmp   xfer
C7BB:8D 28 C0     30 swmint     sta   rombank       ;Mouse interrupt handler
C7BE:4C 00 C1     31            jmp   mouseint
C7C1:8D 28 C0     32 banger     sta   rombank
C7C4:4C 8E D4     33            jmp   diags
C7C7:8D 28 C0     34 swatalk    sta   rombank       ;Jump to appletalk
C7CA:4C 80 C5     35            jmp   atalk
C7CD:8D 28 C0     36 swser3     sta   rombank       ;Jump to serout3
C7D0:4C 4F C2     37            jmp   serout3
C7D3:8D 28 C0     38 swgetst    sta   rombank       ;Jump to getstat
C7D6:4C AC C2     39            jmp   getstat
C7D9:8D 28 C0     40 swread     sta   rombank       ;Jump to xrdser
C7DC:4C C3 C2     41            jmp   xrdser
C7DF:8D 28 C0     42 swgetb     sta   rombank       ;Jump to getbuf
C7E2:4C F7 C2     43            jmp   getbuf
C7E5:8D 28 C0     44 swzznm     sta   rombank
C7E8:4C C5 D4     45            jmp   zznm
C7EB:8D 28 C0     46 swxfgo     sta   rombank       ;Jump to users xfer dest
C7EE:6C ED 03     47            jmp   ($3ED)
C7F1:20 23 CE     48 swsthk3    jsr   sethooks
C7F4:80 8E  C784  49            bra   swrts
C7F6:20 4D CE     50 swzzqt3    jsr   zzquit
C7F9:80 89  C784  51            bra   swrts

C7FB:D6           53            dfb   $D6           ;should be at $C7FB

C7FC:        0003 55            ds    $C7FF-*,0
C7FF:00           56            dfb   0             ;Appletalk version number
C800:             52            include irqbuf      ;Interrupt stuff @$C800
```

```
C800:                    3 ****************************************
C800:                    4 *
C800:                    5 * NEWIRQ - The main (only) IRQ handling routines
C800:                    6 * IRQENT - Entry point from alternate rom bank
C800:                    7 *
C800:                    8 *
C800:                    9 * This routine saves the memory state of the machine,
C800:                   10 * checks for an internal interrupt, and then calls the user's
C800:                   11 * interrupt handler at $3FE.
C800:                   12 * The memory state is encoded as follows:
C800:                   13 *  D7 = 1 if Alernate zero page / stack
C800:                   14 *  D6 = 1 if 80 store and page 2
C800:                   15 *  D5 = 1 if Read aux
C800:                   16 *  D4 = 1 if Write Aux
C800:                   17 *  D3 = 1 if L.C. enabled
C800:                   18 *  D2 = 1 if L.C. and $D000 bank 1
C800:                   19 *  D1 = 1 if L.C. and $D000 bank 2
C800:                   20 *  D0 = 1 if Alternate rom bank
C800:                   21 *
C800:                   22 * New changes in the interrupt handler are marked with a +
C800:                   23 *
C800:                   24 ****************************************
C800:4C 9E C1           25          jmp   p1init        ;Pascal 1.0 Initialization
C803:        C803       26 NEWIRQ   EQU   *             ;+
C803:B8                 27          CLV                  ;+ V=0 for main bank
C804:        C804       28 IRQENT   EQU   *             ;+ Entry pt from other bank assumes V=1
C804:48                 29          PHA                  ;+ Save A on stack, not $45
C805:DA                 30          PHX                  ;+ X too
C806:BA                 31          TSX                  ;+ Save stack pointer
C807:68                 32          PLA                  ;+ Skip past X
C808:68                 33          PLA                  ;+ And A
C809:68                 34          PLA                  ;+ Here is the status Oh boy!
C80A:9A                 35          TXS                  ;+ Fix the stack pointer
C80B:5A                 36          PHY                  ;Save Y too
C80C:AE 66 C0           37          LDX   MOUX1         ;Get mouse info
C80F:AC 67 C0           38          LDY   MOUY1         ;As soon as we can
C812:D8                 39          CLD                  ;+ No decimal mode please
C813:29 10              40          AND   #$10          ;+ Test break bit
C815:C9 10              41          CMP   #$10          ;+ C=1 if break. V unchanged
C817:AD 18 C0           42          LDA   RD80COL       ;TEST FOR 80-STORE WITH
C81A:2D 1C C0           43          AND   RDPAGE2       ; PAGE 2 TEXT.
C81D:29 80              44          AND   #$80          ; MAKE IT ZERO OR $80
C81F:F0 05   C826       45          BEQ   IRQ2
C821:8D 54 C0           46          STA   TXTPAGE1
C824:A9 40              47          LDA   #$40          ;SET PAGE 2 RESET BIT.
C826:50 02   C82A       48 IRQ2     BVC   IRQ21         ;+ Which Rombank?
C828:09 01              49          ORA   #01           ;+ Mark other bank
C82A:2C 13 C0           50 IRQ21    BIT   RDRAMRD
C82D:10 05   C834       51          BPL   IRQ3          ;BRANCH IF MAIN RAM READ
C82F:8D 02 C0           52          STA   RDMAINRAM     ;ELSE, SWITCH IT IN
C832:09 20              53          ORA   #$20          ; AND RECORD THE EVENT!
C834:2C 14 C0           54 IRQ3     BIT   RDRAMWRT      ;DO THE SAME FOR RAM WRITE.
C837:10 05   C83E       55          BPL   IRQ4
C839:8D 04 C0           56          STA   WRMAINRAM
C83C:09 10              57          ORA   #$10
C83E:B0 08   C848       58 IRQ4     BCS   IRQ5          ;Branch if break
C840:48                 59          PHA                  ;Save machine states so far...
C841:20 BB C7           60          JSR   SWMINT        ;+ Go Test Mouse & ACIA
```



```
C844:90 3C   C882       61          BCC   IRQLCOK       ;+ Branch if it was.  LC unchanged!
C846:68                 62          PLA                  ;Restore states recorded so far
C847:18                 63          CLC                  ;Reset break/interrupt handler
C848:2C 12 C0           64 IRQ5     BIT   RDLCRAM       ;DETERMINE IF LANGUAGE CARD ACTIVE
C84B:80 03   C850       65          bra   passkip1      ;Skip around pascal 1.0 stuff
C84D:        0000       66          ds    $C84D-*,$00
C84D:4C A8 C1           67          jmp   plread
C850:        C850       68 passkip1 equ   *
C850:10 0C   C85E       69          BPL   IRQ7
C852:09 0C              70          ORA   #$C           ;SET TWO BITS SO RESTORED
C854:2C 11 C0           71          BIT   RDLCBNK2      ; LANGUAGE CARD IS WRITE ENABLED
C857:10 02   C85B       72          BPL   IRQ6          ;BRANCH IF NOT PAGE 2 OF $D000
C859:49 06              73          EOR   #$6           ;ENABLE READ FOR PAGE 2 ON EXIT
C85B:8D 81 C0           74 IRQ6     STA   ROMIN
C85E:2C 16 C0           75 IRQ7     BIT   RDALTZP       ;LAST...AND VERY IMPORTANT!
C861:10 0D   C870       76          BPL   IRQ8          ; UNLESS IT IS NOT ENABLED
C863:BA                 77          TSX                  ;SAVE CURRENT STACK POINTER
C864:8E 01 01           78          STX   $101          ;AT BOTTOM OF STACK
C867:AE 00 01           79          LDX   $100          ;GET MAIN STACK POINTER
C86A:9A                 80          TXS
C86B:8D 08 C0           81          STA   SETSTDZP
C86E:09 80              82          ORA   #$80
C870:B0 35   C8A7       83 IRQ8     BCS   GOBREAK
C872:48                 84          PHA
C873:A9 C8              85          LDA   #<IRQDONE
C875:48                 86          PHA
C876:A9 7F              87          LDA   #>IRQDONE     ;SAVE RETURN IRQ ADDR
C878:48                 88          PHA
C879:A9 04              89          LDA   #4            ; SO WHEN INTERRUPT DOES RTI
C87B:48                 90          PHA                  ; IT RETURNS TO IRQDONE.
C87C:6C FE 03           91          JMP   ($3FE)        ;PROCESS EXTERNAL INTERRUPT


C87F:                   93 * The user's RTI returns here
C87F:                   94 * BEWARE
C87F:                   95 *  The rom must be reenabled with a LDA romin
C87F:                   96 *  This way if the LC was write protected, it still is
C87F:                   97 *   if it was write enabled, it still is
C87F:                   98 *   if it was being write enabled ( 2 ldas), it still will be
C87F:                   99 *  The restore loop uses an INC because some of the switches are read
C87F:                  100 *  and some are write.  It must be an INC abs,x since both the 6502 and
C87F:                  101 *  the 65C02 do two reads before the write (for different reasons).
C87F:AD 81 C0          102 IRQDONE LDA   ROMIN          ;+ Did some clown bank out the rom?
C882:68                103 IRQLCOK PLA                   ;Recover machine state
C883:10 07   C88C      104          BPL   IRQDN1        ;Branch if main zp was active
C885:8D 09 C0          105          STA   SETALTZP
C888:AE 01 01          106          LDX   $101          ;Restore alternate stack pointer
C88B:9A                107          TXS
C88C:A0 06             108 IRQDN1  LDY   #$06          ;+ Y = index into table of switches
C88E:10 06   C896      109 IRQDN2  BPL   IRQDN3        ;+ Branch if no change
C890:BE 86 CF          110          LDX   IRQTBLE,Y     ;+ Get soft switch address
C893:FE 00 C0          111          INC   $C000,X       ;+ Hit the switch. No page cross!!!
C896:88                112 IRQDN3  DEY
C897:30 03   C89C      113          BMI   IRQDN4        ;+ Branch if all done
C899:0A                114          ASL   A             ;Get next bit to check
C89A:D0 F2   C88E      115          BNE   IRQDN2        ;+ Fall through if all done
C89C:0A                116 IRQDN4  ASL   A             ;+ C = 1 if other rom bank
C89D:0A                117          ASL   A             ;+
```

```
C89E:7A          118          PLY
C89F:FA          119          PLX                ;RESTORE ALL REGISTERS
C8A0:68          120          PLA
C8A1:B0 01  C8A4 121          BCS   IRQDN5       ;+ Which rom bank?
C8A3:40          122          RTI                 ;DO THE REAL RTI!
C8A4:4C 80 C7    123 IRQDN5   JMP   SWRTI        ;+ Go back to the other bank


C8A7:            125 ****************************************
C8A7:            126 *
C8A7:            127 * GOBREAK- If a braek instruction has occurred, we check
C8A7:            128 * if the BRK happened in the alternate rom bank.  If it has,
C8A7:            129 * some fool may have hit the rom switch by accident and the PC is
C8A7:            130 * decremented by two, the main rom is switched in and we resume
C8A7:            131 * where we think he wanted to go
C8A7:            132 *
C8A7:            133 ****************************************
C8A7:       C8A7 134 GOBREAK  EQU   *
C8A7:30 20  C8C9 135          BMI   GBBRK        ;Give up if alt zp
C8A9:89 09       136          BIT   #9           ;From alt rom and no lang card?
C8AB:F0 1C  C8C9 137          BEQ   GBBRK        ;If not then break
C8AD:29 FE       138          AND   #$FE         ;Force main rom
C8AF:48          139          PHA                 ;Save state
C8B0:BA          140          TSX                 ;Save stack pointer
C8B1:68          141          PLA                 ;Skip State
C8B2:68          142          PLA                 ;Skip Y
C8B3:68          143          PLA                 ;Skip X
C8B4:68          144          PLA                 ;Skip A
C8B5:68          145          PLA                 ;Skip P
C8B6:68          146          PLA                 ;> address
C8B7:7A          147          PLY                 ;< address
C8B8:C0 C1       148          CPY   #$C1         ;In the ROM?
C8BA:90 0B  C8C7 149          BCC   GBNOTROM     ;Branch if not
C8BC:E9 02       150          SBC   #2           ;PC = PC - 2
C8BE:B0 01  C8C1 151          BCS   GBNOC
C8C0:88          152          DEY                 ;Borrow from high byte
C8C1:5A          153 GBNOC    PHY                 ;Push new address
C8C2:48          154          PHA
C8C3:9A          155          TXS                 ;Fix stack pointer
C8C4:4C 7F C8    156          JMP   IRQDONE
C8C7:9A          157 GBNOTROM TXS                 ;Fix stack pointer
C8C8:68          158          PLA                 ;Get state back
C8C9:4C 47 FA    159 GBBRK    JMP   NEWBRK       ;Go do the break
```



```
C8CC:            162 *   The following routine is for reading key-
C8CC:            163 * board from buffers or directly.
C8CC:            164 *   Type-ahead buffering only occurs for non auto-
C8CC:            165 * repeat keypresses.  When a key is pressed for
C8CC:            166 * auto-repeat the buffer is first emptied, then the
C8CC:            167 * repeated characters are returned.
C8CC:            168 *   The minus flag is used to indicate if a keystroke
C8CC:            169 * is being returned.
C8CC:            170 *

C8CC:AD 00 C0    172 XRKBD1   LDA   KBD          ;test keyboard directly
C8CF:10 04  C8D5 173          BPL   XRDKBD       ;loop if buffered since test.
C8D1:8D 10 C0    174          STA   KBDSTRB      ;Clear keyboard strobe.
C8D4:60          175 XNOKEY   RTS                ;Minus flag indicates valid character

C8D5:20 E6 C8    177 XRDKBD   JSR   XBITKBD      ;is keyboard input ready?
C8D8:10 FA  C8D4 178          BPL   XNOKEY       ;Branch if not.
C8DA:90 F0  C8CC 179          BCC   XRKBD1       ;Branch if direct KBD input.
C8DC:5A          180          PHY                ;Save Y
C8DD:A0 80       181          LDY   #$80         ;Y=$80 for keyboard buffer
C8DF:20 DF C7    182          JSR   SWGETB       ;Get data from buffer
C8E2:7A          183          PLY
C8E3:09 00       184          ORA   #0           ;Set minus flag
C8E5:60          185          RTS

C8E6:2C FA 05    187 XBITKBD  BIT   TYPHED       ;This routine replaces "BIT KBD"
C8E9:10 10  C8FB 188          BPL   XBKB2        ; instructions so as to function with
C8EB:            189 ;                             type-ahead
C8EB:38          190          SEC                ;anticipate data in buffer is ready
C8EC:08          191          PHP                ;save carry and minus flags
C8ED:48          192          PHA                ;preserve accumulator
C8EE:AD FF 06    193          LDA   TRKEY
C8F1:CD FC 05    194          CMP   TWKEY        ;is there data to be read?
C8F4:F0 03  C8F9 195          BEQ   XBKB1        ;branch if type-ahead buffer empty
C8F6:68          196          PLA
C8F7:28          197          PLP
C8F8:60          198          RTS                ;Carry and minus flag already set.
C8F9:            199 *
C8F9:68          200 XBKB1    PLA
C8FA:28          201          PLP                ;restore ACC and Status
C8FB:2C 00 C0    202 XBKB2    BIT   KBD          ;test KBD Directly
C8FE:18          203          CLC                ;indicate direct test
C8FF:60          204          RTS


C900:            206 ****************************************
C900:            207 *
C900:            208 * PADDLE patch
C900:            209 * This routine returns the mouse position instead of
C900:            210 * the paddle if the mouse is on
C900:            211 *
C900:            212 ****************************************
C900:       C900 213 mpaddle  equ   *
C900:AD FF 07    214          lda   moumode      ;Is the mouse active?
C903:C9 01       215          cmp   #01          ;Only transparent mode
C905:F0 06  C90D 216          beq   pdon
C907:AD 70 C0    217          lda   vblclr       ;Fire the strobe
```

```
C90A:4C 21 FB      218          jmp   $FB21
C90D:        C90D  219 pdon     equ   *
C90D:E0 01         220          cpx   #1          ;C=1 if X=1
C90F:6A            221          ror   A           ;A=80 or 0
C910:A8            222          tay
C911:B9 7F 05      223          lda   mouxh,y     ;Get high byte
C914:F0 02   C918  224          beq   pdok
C916:A9 FF         225          lda   #$FF
C918:19 7F 04      226 pdok     ora   mouxl,y
C91B:A8            227          tay
C91C:60            228          rts
C91D:               53          include mini      ;Mini assembler & step routines
```



```
C91D:                3 ************************************
C91D:                4 *
C91D:                5 * Apple //c Mini Assembler
C91D:                6 *
C91D:                7 * Got mnemonic, check address mode
C91D:                8 *
C91D:                9 ************************************
C91D:20 3B CA       10 AMOD1    JSR   NNBL         ;get next non-blank
C920:84 34          11          STY   YSAV         ;save Y
C922:DD BA F9       12          CMP   CHAR1,X
C925:D0 13   C93A   13          BNE   AMOD2
C927:20 3B CA       14          JSR   NNBL         ;get next non-blank
C92A:DD B4 F9       15          CMP   CHAR2,X
C92D:F0 0D   C93C   16          BEQ   AMOD3
C92F:BD B4 F9       17          LDA   CHAR2,X      ;done yet?
C932:F0 07   C93B   18          BEQ   AMOD4
C934:C9 A4          19          CMP   #$A4         ;if "$" then done
C936:F0 03   C93B   20          BEQ   AMOD4
C938:A4 34          21          LDY   YSAV         ;restore Y
C93A:18             22 AMOD2    CLC
C93B:88             23 AMOD4    DEY
C93C:26 44          24 AMOD3    ROL   A5L          ;shift bit into format
C93E:E0 03          25          CPX   #$03
C940:D0 0D   C94F   26          BNE   AMOD6
C942:20 A7 FF       27          JSR   GETNUM
C945:A5 3F          28          LDA   A2H          ;get high byte of address
C947:F0 01   C94A   29          BEQ   AMOD5        ;=>
C949:E8             30          INX
C94A:86 35          31 AMOD5    STX   YSAV1
C94C:A2 03          32          LDX   #$03
C94E:88             33          DEY
C94F:86 3D          34 AMOD6    STX   A1H
C951:CA             35          DEX
C952:10 C9   C91D   36          BPL   AMOD1
C954:60             37          RTS


C955:               39 *
C955:               40 *
C955:               41 * Calculate offset byte for relative addresses
C955:               42 *
C955:E9 81          43 REL      SBC   #$81         ;calc relative address
C957:4A             44          LSR   A
C958:D0 14   C96E   45          BNE   GOERR        ;bad branch
C95A:A4 3F          46          LDY   A2H
C95C:A6 3E          47          LDX   A2L
C95E:D0 01   C961   48          BNE   REL1
C960:88             49          DEY                ;point to offset
C961:CA             50 REL1     DEX                 ;displacement - 1
C962:8A             51          TXA
C963:18             52          CLC
C964:E5 3A          53          SBC   PCL          ;subtract current PCL
C966:85 3E          54          STA   A2L          ;and save as displacement
C968:10 01   C96B   55          BPL   REL2         ;check page
C96A:C8             56          INY
C96B:98             57 REL2     TYA                 ;get page
C96C:E5 3B          58          SBC   PCH          ;check page
```

```
C96E:D0 57   C9C7   59 GOERR    BNE   MINIERR       ;display error
C970:               60 *
C970:               61 * Move instruction to memory
C970:               62 *
C970:A4 2F          63 MOVINST  LDY   LENGTH        ;get instruction length
C972:B9 3D 00       64 MOV1     LDA   A1H,Y         ;get a byte
C975:91 3A          65          STA   (PCL),Y       ;and move it
C977:88             66          DEY
C978:10 F8   C972   67          BPL   MOV1
C97A:               68 *
C97A:               69 * Display instruction
C97A:               70 *
C97A:20 48 F9       71          JSR   PRBLNK        ;print blanks to make ProDOS work
C97D:20 1A FC       72          JSR   UP            ;move up 2 lines
C980:20 1A FC       73          JSR   UP
C983:       C983    74 DISLIN   EQU   *
C983:20 C4 C5       75          JSR   SHOWINST      ;Display line & get next instruction
C986:       C986    76 GETINST1 EQU   *             ;Get the next instruction
C986:A9 A1          77          LDA   #$A1          ;! for prompt
C988:85 33          78          STA   PROMPT
C98A:20 67 FD       79          JSR   GETLNZ        ;Get a line
C98D:80 49   C9D8   80          BRA   DOINST        ;Go do the instruction
C98F:               81 *
C98F:               82 * Compare disassembly of all known opcodes with
C98F:               83 * the one typed in until a match is found
C98F:               84 *
C98F:A5 3D          85 GETOP    LDA   A1H           ;get opcode
C991:20 8E F8       86          JSR   INSDS2        ;determine mnemonic index
C994:AA             87          TAX                  ;X = index
C995:BD 00 FA       88          LDA   MNEMR,X       ;get right half of index
C998:C5 42          89          CMP   A4L           ;does it match entry?
C99A:D0 21   C9BD   90          BNE   NXTOP         ;=>try next opcode
C99C:BD C0 F9       91          LDA   MNEML,X       ;get left half of index

C99F:80 0C   C9AD   93          bra   plskip        ;Skip past pascal stuff
C9A1:       0009    94          ds    $C9AA-*,0     ;Hello I'm the pascal 1.0 entry point
C9AA:4C B4 C1       95          jmp   plwrite       ;Just getting in the way
C9AD:       C9AD    96 plskip   equ   *

C9AD:C5 43          98          CMP   A4H           ;does it match entry?
C9AF:D0 0C   C9BD   99          BNE   NXTOP         ;=>no, try next opcode
C9B1:A5 44         100          LDA   A5L           ;found opcode, check address mode
C9B3:A4 2E         101          LDY   FORMAT        ;get addr. mode format for that opcode
C9B5:C0 9D         102          CPY   #$9D          ;is it relative?
C9B7:F0 9C   C955  103          BEQ   REL           ;=>yes, calc relative address
C9B9:C5 2E         104          CMP   FORMAT        ;does mode match?
C9BB:F0 B3   C970  105          BEQ   MOVINST       ;=>yes, move instruction to memory
C9BD:C6 3D         106 NXTOP    DEC   A1H           ;else try next opcode
C9BF:D0 CE   C98F  107          BNE   GETOP         ;=>go try it
C9C1:E6 44         108          INC   A5L           ;else try next format
C9C3:C6 35         109          DEC   YSAV1
C9C5:F0 C8   C98F  110          BEQ   GETOP         ;=>go try next format
C9C7:              111 *
C9C7:              112 * Point to the error with a caret, beep, and fall
C9C7:              113 * into the mini-assembler.
C9C7:              114 *
C9C7:A4 34         115 MINIERR  LDY   YSAV          ;get position
C9C9:98            116 ERR2     TYA
```



```
C9CA:AA            117          TAX
C9CB:       C9CB   118 ERR3     EQU   *
C9CB:20 4A F9      119          JSR   PRBL2
C9CE:A9 DE         120          LDA   #$DE          ;^ to point to error
C9D0:20 ED FD      121          JSR   COUT
C9D3:20 3A FF      122          JSR   BELL          ;Beep cause we're mad
C9D6:80 AE   C986  123          BRA   GETINST1      ;try again
C9D8:              124 *
C9D8:              125 * Read a line of input.  If prefaced with " ", decode
C9D8:              126 * mnemonic. If "$" do monitor command.  Otherwise parse
C9D8:              127 * hex address before decoding mnemonic.
C9D8:              128 *
C9D8:20 C7 FF      129 DOINST   JSR   ZMODE         ;clear mode
C9DB:AD 00 02      130          LDA   $200          ;get first char in line
C9DE:C9 A0         131          CMP   #$A0          ;if blank,
C9E0:F0 12   C9F4  132          BEQ   DOLIN         ;=>go attempt disassembly
C9E2:C9 8D         133          CMP   #$8D          ;is it return?
C9E4:D0 01   C9E7  134          BNE   GETI1         ;=>no, continue
C9E6:60            135          RTS                 ;else return to Monitor
C9E7:              136 *
C9E7:20 A7 FF      137 GETI1    JSR   GETNUM        ;parse hexadecimal input
C9EA:C9 93         138          CMP   #$93          ;look for "ADDR:"
C9EC:D0 DB   C9C9  139 GOERR2   BNE   ERR2          ;no ":", display error
C9EE:8A            140          TXA                 ;X nonzero if address entered
C9EF:F0 D8   C9C9  141          BEQ   ERR2          ;no "ADDR", display error
C9F1:              142 *
C9F1:20 78 FE      143          JSR   A1PCLP        ;move address to PC
C9F4:A9 03         144 DOLIN    LDA   #$03          ;get starting opcode
C9F6:85 3D         145          STA   A1H           ;and save
C9F8:20 3B CA      146 NXTCH    JSR   NNBL          ;get next non-blank
C9FB:0A            147          ASL   A             ;validate entry
C9FC:E9 BE         148          SBC   #$BE
C9FE:C9 C2         149          CMP   #$C2
CA00:90 C7   C9C9  150          BCC   ERR2          ;=>flag bad mnemonic
CA02:              151 *
CA02:              152 * Form mnemonic for later comparison
CA02:              153 *
CA02:0A            154          ASL   A
CA03:0A            155          ASL   A
CA04:A2 04         156          LDX   #$04
CA06:0A            157 NXTMN    ASL   A
CA07:26 42         158          ROL   A4L
CA09:26 43         159          ROL   A4H
CA0B:CA            160          DEX
CA0C:10 F8   CA06  161          BPL   NXTMN
CA0E:C6 3D         162          DEC   A1H           ;decrement mnemonic count
CA10:F0 F4   CA06  163          BEQ   NXTMN
CA12:10 E4   C9F8  164          BPL   NXTCH
CA14:A2 05         165          LDX   #$5           ;index into address mode tables
CA16:20 1D C9      166          JSR   AMOD1         ;do this elsewhere
CA19:A5 44         167          LDA   A5L           ;get format
CA1B:0A            168          ASL   A
CA1C:0A            169          ASL   A
CA1D:05 35         170          ORA   YSAV1
CA1F:C9 20         171          CMP   #$20
CA21:B0 06   CA29  172          BCS   AMOD7
CA23:A6 35         173          LDX   YSAV1         ;get our format
CA25:F0 02   CA29  174          BEQ   AMOD7
```

```
CA27:09 80        175          ORA   #$80
CA29:85 44        176 AMOD7    STA   A5L          ;update format
CA2B:84 34        177          STY   YSAV         ;update position
CA2D:B9 00 02     178          LDA   $0200,Y      ;get next character
CA30:C9 BB        179          CMP   #$BB         ;is it a ";"?
CA32:F0 04   CA38 180          BEQ   AMOD8        ;=>yes, skip comment
CA34:C9 8D        181          CMP   #$8D         ;is it carriage return
CA36:D0 B4   C9EC 182          BNE   GOERR2
CA38:4C 8F C9     183 AMOD8    JMP   GETOP        ;get next opcode



CA3B:             185 ****************************************
CA3B:             186 *
CA3B:             187 * NNBL - Gets a non blank character for the mini assembler
CA3B:             188 *
CA3B:             189 ****************************************
CA3B:       CA3B  190 nnbl     equ   *
CA3B:20 B4 C5     191          jsr   getup        ;Get next upshifted character
CA3E:C9 A0        192          cmp   #$A0         ;Blank?
CA40:F0 F9   CA3B 193          beq   nnbl
CA42:60           194          rts
```



```
CA43:             196 ****************************************
CA43:             197 *
CA43:             198 * Step and trace routines
CA43:             199 *
CA43:             200 ****************************************
CA43:       CA43  201 step     equ   *
CA43:2C 61 C0     202          bit   butn0        ;Open apple =  slow step
CA46:10 08   CA50 203          bpl   xqnobt0
CA48:A2 07        204          ldx   #7           ;Wait about a second
CA4A:20 A8 FC     205 xqwait   jsr   wait
CA4D:CA           206          dex
CA4E:D0 FA   CA4A 207          bne   xqwait
CA50:2C 62 C0     208 xqnobt0  bit   butn1
CA53:30 51   CAA6 209          bmi   xbrk         ;Closed apple = break
CA55:20 75 FE     210          jsr   a1pc         ;If user specified an address, move it
CA58:18           211          clc
CA59:20 0D CB     212          jsr   godsp        ;Disassemble one instruction
CA5C:68           213          pla                ;At (PCL,H)
CA5D:85 2C        214          sta   rtnl         ;Adjust to user stack
CA5F:68           215          pla
CA60:85 2D        216          sta   rtnh         ;Save return address
CA62:A2 08        217          ldx   #$08
CA64:BD 04 CB     218 xqinit   lda   initbl-1,x   ;Init XEQ area
CA67:95 3C        219          sta   xqt,x
CA69:CA           220          dex
CA6A:D0 F8   CA64 221          bne   xqinit
CA6C:A1 3A        222          lda   (pcl,x)
CA6E:F0 36   CAA6 223          beq   xbrk         ;Special if break
CA70:A4 2F        224          ldy   length
CA72:C9 20        225          cmp   #$20
CA74:F0 4A   CAC0 226          beq   xjsr         ;Do JSR, RTS, JMP, JMP (), JMP (,X), RTI
CA76:C9 60        227          cmp   #$60
CA78:F0 36   CAB0 228          beq   xrts
CA7A:C9 4C        229          cmp   #$4C
CA7C:F0 4A   CAC8 230          beq   xjmp
CA7E:C9 6C        231          cmp   #$6C
CA80:F0 47   CAC9 232          beq   xjmpat
CA82:C9 7C        233          cmp   #$7C
CA84:F0 5D   CAE3 234          beq   xjmpatx
CA86:C9 40        235          cmp   #$40
CA88:F0 22   CAAC 236          beq   xrti
CA8A:C9 80        237          cmp   #$80         ;Make bra turn into bpl
CA8C:D0 02   CA90 238          bne   xqntbra
CA8E:A9 10        239          lda   #$10
CA90:29 1F        240 xqntbra  and   #$1F
CA92:49 14        241          eor   #$14
CA94:C9 04        242          cmp   #$04
CA96:F0 02   CA9A 243          beq   xq2          ;Copy user inst to xeq area
CA98:B1 3A        244 xq1      lda   (pcl),y      ;Change rel branch
CA9A:99 3C 00     245 xq2      sta   xqt,y        ;displacement to 4 for jmp to branch
CA9D:88           246          dey                ;or jump to nbranch
CA9E:10 F8   CA98 247          bpl   xq1
CAA0:20 3F FF     248          jsr   restore      ;Restore user reg contents
CAA3:4C 3C 00     249          jmp   xqt          ;Xeq user op from ram
CAA6:A9 64        250 xbrk     lda   #>mon-1      ;Print registers and go to monitor
CAA8:A2 FF        251          ldx   #<mon-1
CAAA:80 2D   CAD9 252          bra   rtnjmp2      ;Display regs & go to monitor
CAAC:18           253 xrti     clc
```

```
16 MINI            65C02 Mini assembler                20-OCT-86  06:41 PAGE 47

CAAD:68             254           pla                 ;Simulate rti by getting status
CAAE:85 48          255           sta   status        ;from stack then doing rts
CAB0:68             256 xrts      pla                  ;Pop PC (not pc - 1)!
CAB1:85 3A          257           sta   pcl
CAB3:68             258           pla
CAB4:85 3B          259 pcinc2    sta   pch           ;Update Pc by 1 (Len = 0)
CAB6:A5 2F          260 pcinc3    lda   length        ;Update pc by length
CAB8:20 56 F9       261           jsr   pcadj3
CABB:84 3B          262           sty   pch
CABD:18             263           clc
CABE:90 11   CAD1   264           bcc   newpcl
CAC0:18             265 xjsr      clc
CAC1:20 54 F9       266           jsr   pcadj2
CAC4:5A             267           phy                  ;Push pc onto stack for jsr
CAC5:48             268           pha
CAC6:A0 02          269           ldy   #$02
CAC8:18             270 xjmp      clc
CAC9:B1 3A          271 xjmpat    lda   (pcl),y
CACB:AA             272           tax                  ;Load pc for jmp, (jmp) simulate
CACC:88             273           dey
CACD:B1 3A          274           lda   (pcl),y
CACF:86 3B          275           stx   pch
CAD1:85 3A          276 newpcl    sta   pcl
CAD3:B0 F3   CAC8   277           bcs   xjmp
CAD5:A6 2D          278 rtnjmp    ldx   rtnh
CAD7:A5 2C          279           lda   rtnl
CAD9:DA             280 rtnjmp2   phx
CADA:48             281           pha
CADB:A9 27          282           lda   #39           ;Move over
CADD:85 24          283           sta   ch
CADF:38             284           sec
CAE0:4C 0D CB       285           jmp   godsp
CAE3:18             286 xjmpatx   clc                  ;JMP (,X)
CAE4:A5 3A          287           lda   pcl           ;Add x to address
CAE6:65 46          288           adc   xreg
CAE8:85 3A          289           sta   pcl
CAEA:90 02   CAEE   290           bcc   xjxnoc
CAEC:E6 3B          291           inc   pch
CAEE:38             292 xjxnoc    sec                 ;C = 1 for indirect jump
CAEF:80 D8   CAC9   293           bra   xjmpat
CAF1:18             294 branch    clc                  ;Branch taken
CAF2:A0 01          295           ldy   #$01          ;Add len+2 to PC
CAF4:B1 3A          296           lda   (pcl),y
CAF6:20 56 F9       297           jsr   pcadj3
CAF9:85 3A          298           sta   pcl
CAFB:98             299           tya
CAFC:38             300           sec
CAFD:B0 B5   CAB4   301           bcs   pcinc2
CAFF:20 4A FF       302 nbrnch    jsr   save          ;Normal return from xeq
CB02:38             303           sec
CB03:B0 B1   CAB6   304           bcs   pcinc3        ;Go update PC

CB05:               306 ****************************************
CB05:               307 *
CB05:               308 * This is the table that is moved into zero page
CB05:               309 * when stepping and tracing
CB05:               310 *
CB05:               311 ****************************************
```

```
16 MINI            65C02 Mini assembler                20-OCT-86  06:41 PAGE 48

CB05:EA             312 initbl    nop
CB06:EA             313           nop
CB07:4C FF CA       314           jmp   nbrnch
CB0A:4C F1 CA       315           jmp   branch



CB0D:               317 ****************************************
CB0D:               318 *
CB0D:               319 * GODSP - Saves hooks, calls display routine and fixes hooks
CB0D:               320 * C = 0 instruction display
CB0D:               321 * C = 1 register display
CB0D:               322 * used by step and trace
CB0D:               323 *
CB0D:               324 ****************************************
CB0D:        CB0D   325 godsp     equ   *
CB0D:A5 36          326           lda   cswl
CB0F:48             327           pha
CB10:A5 37          328           lda   cswh          ;Save output hook
CB12:48             329           pha
CB13:A9 F0          330           lda   #>cout1
CB15:85 36          331           sta   cswl
CB17:A9 FD          332           lda   #<cout1
CB19:85 37          333           sta   cswh
CB1B:B0 05   CB22   334           bcs   godreg        ;Which display?
CB1D:20 D0 F8       335           jsr   instdsp
CB20:80 03   CB25   336           bra   goddone
CB22:20 DA FA       337 godreg    jsr   rgdsp1
CB25:68             338 goddone   pla
CB26:85 37          339           sta   cswh
CB28:68             340           pla
CB29:85 36          341           sta   cswl
CB2B:60             342           rts
CB2C:                54           INCLUDE SCROLLING   ;More Video stuff @$CB30
```

```
CB2C:                    3 ;align this for fools with illegal entry points
CB2C:          0004      4          ds   $CB30-*,0
CB30:                    5 *
CB30:                    6 * SCROLLIT scrolls the screen either up or down, depending
CB30:                    7 * on the value of X.  It scrolls within windows with even
CB30:                    8 * or odd edges for both 40 and 80 columns.  It can scroll
CB30:                    9 * windows down to 1 characters wide.
CB30:                   10 *
CB30:DA                 11 SCROLLDN  PHX                 ;save X
CB31:A2 00              12           LDX   #0            ;direction = down
CB33:80 03    CB38      13           BRA   SCROLLIT      ;do scroll
CB35:                   14 *
CB35:DA                 15 SCROLLUP  PHX                 ;save X
CB36:A2 01              16           LDX   #1            ;direction = up
CB38:A4 21              17 SCROLLIT  LDY   WNDWDTH       ;get width of screen window
CB3A:2C 1F C0           18           BIT   RD80VID       ;in 40 or 80 columns?
CB3D:10 18    CB57      19           BPL   GETST         ;=>40, determine starting line
CB3F:8D 01 C0           20           STA   SET80COL      ;make sure this is enabled
CB42:98                 21           TYA                 ;get WNDWDTH for test
CB43:4A                 22           LSR   A             ;divide by 2 for 80 column index
CB44:A8                 23           TAY                 ;and save
CB45:A5 20              24           LDA   WNDLFT        ;test oddity of right edge
CB47:4A                 25           LSR   A             ;by rotating low bit into carry
CB48:B8                 26           CLV                 ;V=0 if left edge even
CB49:90 03    CB4E      27           BCC   CHKRT         ;=>check right edge
CB4B:2C C1 CB           28           BIT   SEV1          ;V=1 if left edge odd
CB4E:2A                 29 CHKRT     ROL   A             ;restore WNDLFT
CB4F:45 21              30           EOR   WNDWDTH       ;get oddity of right edge
CB51:4A                 31           LSR   A             ;C=1 if right edge even
CB52:70 03    CB57      32           BVS   GETST         ;if odd left, don't DEY
CB54:B0 01    CB57      33           BCS   GETST         ;if even right, don't DEY
CB56:88                 34           DEY                 ;if right edge odd, need one less
CB57:8C F8 05           35 GETST     STY   TEMPY         ;save window width
CB5A:AD 1F C0           36           LDA   RD80VID       ;N=1 if 80 columns
CB5D:08                 37           PHP                 ;save N,Z,V
CB5E:A5 22              38           LDA   WNDTOP        ;assume scroll from top
CB60:E0 00              39           CPX   #0            ;up or down?
CB62:D0 03    CB67      40           BNE   SETDBAS       ;=>up
CB64:A5 23              41           LDA   WNDBTM        ;down, start scrolling at bottom
CB66:3A                 42           DEC   A             ;really need one less
CB67:                   43 *
CB67:8D 78 05           44 SETDBAS   STA   TEMPA         ;save current line
CB6A:20 24 FC           45           JSR   VTABZ         ;calculate base with window width
CB6D:                   46 *
CB6D:A5 28              47 SCRLIN    LDA   BASL          ;current line is destination
CB6F:85 2A              48           STA   BAS2L
CB71:A5 29              49           LDA   BASH
CB73:85 2B              50           STA   BAS2H
CB75:                   51 *
CB75:AD 78 05           52           LDA   TEMPA         ;get current line
CB78:E0 00              53           CPX   #0            ;going up?
CB7A:D0 07    CB83      54           BNE   SETUP2        =>up, inc current line
CB7C:C5 22              55           CMP   WNDTOP        ;down. Reached top yet?
CB7E:F0 39    CBB9      56           BEQ   SCRL3         ;yes! clear top line, exit
CB80:3A                 57           DEC   A             ;no, go up a line
CB81:80 05    CB88      58           BRA   SETSRC        ;set source for scroll
CB83:1A                 59 SETUP2    INC   A             ;up, inc current line
CB84:C5 23              60           CMP   WNDBTM        ;at bottom yet?
```



```
CB86:B0 31    CBB9      61           BCS   SCRL3         ;yes! clear bottom line, exit
CB88:                   62 *
CB88:8D 78 05           63 SETSRC    STA   TEMPA         ;save new current line
CB8B:20 24 FC           64           JSR   VTABZ         ;get base for new current line
CB8E:AC F8 05           65           LDY   TEMPY         ;get width for scroll
CB91:28                 66           PLP                 ;get status for scroll
CB92:08                 67           PHP                 ;N=1 if 80 columns
CB93:10 1F    CBB4      68           BPL   SKPRT         ;=>only do 40 columns
CB95:AD 55 C0           69           LDA   TXTPAGE2      ;scroll aux page first (even bytes)
CB98:98                 70           TYA                 ;test Y
CB99:F0 07    CBA2      71           BEQ   SCRLFT        ;if Y=0, only scroll one byte
CB9B:B1 28              72 SCRLEVEN  LDA   (BASL),Y
CB9D:91 2A              73           STA   (BAS2L),Y
CB9F:88                 74           DEY
CBA0:D0 F9    CB9B      75           BNE   SCRLEVEN      ;do all but last even byte
CBA2:70 04    CBA8      76 SCRLFT    BVS   SKPLFT        ;odd left edge, skip this byte
CBA4:B1 28              77           LDA   (BASL),Y
CBA6:91 2A              78           STA   (BAS2L),Y
CBA8:AD 54 C0           79 SKPLFT    LDA   TXTPAGE1      ;now do main page (odd bytes)
CBAB:AC F8 05           80           LDY   TEMPY         ;restore width
CBAE:B0 04    CBB4      81           BCS   SKPRT         ;even right edge, skip this byte
CBB0:B1 28              82 SCRLODD   LDA   (BASL),Y
CBB2:91 2A              83           STA   (BAS2L),Y
CBB4:88                 84 SKPRT     DEY
CBB5:10 F9    CBB0      85           BPL   SCRLODD
CBB7:80 B4    CB6D      86           BRA   SCRLIN        ;scroll next line
CBB9:                   87 *
CBB9:20 A0 FC           88 SCRL3     JSR   CLRLIN        ;clear current line
CBBC:20 22 FC           89           JSR   VTAB          ;restore original cursor line
CBBF:28                 90           PLP                 ;pull status off stack
CBC0:FA                 91           PLX                 ;restore X
CBC1:60                 92 SEV1      RTS                 ;done!!!
```

```
CBC2:                 94 *
CBC2:                 95 * DOCLR is called by CLREOL.  It decides whether
CBC2:                 96 * to do a (quick) 40 or 80 column clear to end of line.
CBC2:                 97 *
CBC2:2C 1F C0         98 DOCLR    BIT   RD80VID       ;40 or 80 column clear?
CBC5:30 13   CBDA     99          BMI   CLR80         ;=>clear 80 columns
CBC7:91 28           100 CLR40    STA   (BASL),Y
CBC9:C8              101          INY
CBCA:C4 21           102          CPY   WNDWDTH
CBCC:90 F9   CBC7    103          BCC   CLR40
CBCE:60              104          RTS
CBCF:                105 *
CBCF:DA              106 CLRHALF  PHX                  ;clear right half of screen
CBD0:A2 D8           107          LDX   #$D8          ;for SCRN48
CBD2:A0 14           108          LDY   #20
CBD4:A5 32           109          LDA   INVFLG
CBD6:29 A0           110          AND   #$A0
CBD8:80 17   CBF1    111          BRA   CLR2          ;=>jump into middle
CBDA:                112 *
CBDA:DA              113 CLR80    PHX                  ;preserve X
CBDB:48              114          PHA                  ;and blank
CBDC:98              115          TYA                  ;get count for CH
CBDD:48              116          PHA                  ;save for left edge check
CBDE:38              117          SEC                  ;count=WNDWDTH-Y-1
CBDF:E5 21           118          SBC   WNDWDTH
CBE1:AA              119          TAX                  ;save CH counter
CBE2:98              120          TYA                  ;div CH by 2 for half pages
CBE3:4A              121          LSR   A
CBE4:A8              122          TAY
CBE5:68              123          PLA                   ;restore original CH
CBE6:45 20           124          EOR   WNDLFT         ;get starting page
CBE8:6A              125          ROR   A
CBE9:B0 03   CBEE    126          BCS   CLR0
CBEB:10 01   CBEE    127          BPL   CLR0
CBED:C8              128          INY                  ;iff WNDLFT odd, starting byte odd
CBEE:68              129 CLR0     PLA                  ;get blankity blank
CBEF:B0 0B   CBFC    130          BCS   CLR1           ;starting page is 1 (default)
CBF1:2C 55 C0        131 CLR2     BIT   TXTPAGE2       ;else do page 2
CBF4:91 28           132          STA   (BASL),Y
CBF6:2C 54 C0        133          BIT   TXTPAGE1       ;now do page 1
CBF9:E8              134          INX
CBFA:F0 06   CC02    135          BEQ   CLR3           ;all done
CBFC:91 28           136 CLR1     STA   (BASL),Y
CBFE:C8              137          INY                  ;forward 2 columns
CBFF:E8              138          INX                  ;next CH
CC00:D0 EF   CBF1    139          BNE   CLR2           ;not done yet
CC02:FA              140 CLR3     PLX                  ;restore X
CC03:60              141          RTS                  ;and exit
CC04:                142 *
CC04:9C FA 05        143 CLRPORT  STZ   TYPHED         ;disable typeahead
CC07:9C F9 05        144          STZ   EXTINT2        ;and external interrupts
CC0A:60              145          RTS
```



```
CC0B:                147 *
CC0B:                148 * PASINVERT is used by Pascal to display the cursor. Pascal
CC0B:                149 * normally leaves the cursor on the screen at all times. It
CC0B:                150 * is fleetingly removed while a character is displayed, then
CC0B:                151 * promptly redisplayed.  CTL-F and CTL-E, respectively,
CC0B:                152 * disable and enable display of the cursor when printed using
CC0B:                153 * the Pascal 1.1 entry point (PWRITE).  Screen I/O is
CC0B:                154 * significantly faster when the cursor is disabled.  This
CC0B:                155 * feature is supported by Pascal 1.2 and later.
CC0B:                156 *
CC0B:AD FB 04        157 PASINVERT LDA  VMODE          ;Called by pascal to
CC0E:29 10           158          AND   #M.CURSOR      ;display cursor
CC10:D0 0A   CC1C    159          BNE   INVX           ;=>cursor off, don't invert
CC12:        CC12    160 INVERT   EQU   *
CC12:20 1D CC        161          JSR   PICKY          ;load Y and get char
CC15:48              162          PHA
CC16:49 80           163          EOR   #$80           ;FLIP INVERSE/NORMAL
CC18:20 B3 C3        164          JSR   STORY          ;stuff onto screen
CC1B:68              165          PLA                  ;for RDCHAR
CC1C:60              166 INVX     RTS
CC1D:                167 *
CC1D:                168 * PICK lifts a character from the screen in either
CC1D:                169 * 40 or 80 columns from the current cursor position.
CC1D:                170 * If the alternate character set is switched in,
CC1D:                171 * character codes $0-$1F are returned as $40-$5F (which
CC1D:                172 * is what must have been originally printed to the location).
CC1D:                173 *
CC1D:5A              174 PICKY    PHY                  ;save Y
CC1E:20 9D CC        175          JSR   GETCUR         ;get newest cursor into Y
CC21:AD 1F C0        176          LDA   RD80VID        ;80 columns?
CC24:10 17   CC3D    177          BPL   PICK1          ;=>no
CC26:8D 01 C0        178          STA   SET80COL       ;force 80STORE if 80 columns
CC29:98              179          TYA
CC2A:45 20           180          EOR   WNDLFT         ;C=1 if char in main RAM
CC2C:6A              181          ROR   A              ;get low bit into carry
CC2D:B0 04   CC33    182          BCS   PICK2          ;=>store in main memory
CC2F:AD 55 C0        183          LDA   TXTPAGE2       ;else switch in page 2
CC32:C8              184          INY                  ;for odd left, aux bytes
CC33:98              185 PICK2    TYA                  ;divide pos'n by 2
CC34:4A              186          LSR   A
CC35:A8              187          TAY                  ;and use as offset into line
CC36:B1 28           188          LDA   (BASL),Y       ;pick character
CC38:8D 54 C0        189          STA   TXTPAGE1       ;80 columns, switch in
CC3B:80 02   CC3F    190          BRA   PICK3          ;skip 40 column pick
CC3D:B1 28           191 PICK1    LDA   (BASL),Y       ;pick 40 column char
CC3F:2C 1E C0        192 PICK3    BIT   ALTCHARSET     ;only allow if alt set
CC42:10 06   CC4A    193          BPL   PICK4
CC44:C9 20           194          CMP   #$20
CC46:B0 02   CC4A    195          BCS   PICK4
CC48:09 40           196          ORA   #$40
CC4A:7A              197 PICK4    PLY                  ;restore real Y
CC4B:60              198          RTS
CC4C:                199 *
CC4C:                200 * SHOWCUR displays either a checkerboard cursor, a solid
CC4C:                201 * rectangle, or the current cursor character, depending
CC4C:                202 * on the value of the CURSOR location.  0=inverse cursor,
CC4C:                203 * $FF=checkerboard cursor, anything else is displayed
CC4C:                204 * after being anded with inverse mask.
```

```
CC4C:                 205 *
CC4C:AC FB 07         206 SHOWCUR   LDY   CURSOR        ;what's my type?
CC4F:D0 02   CC53     207           BNE   NOTINV        ;=>not inverse
CC51:80 BF   CC12     208           BRA   INVERT        ;else invert the char (exit)
CC53:                 209 *
CC53:                 210 * Exit with char in accumulator
CC53:                 211 *
CC53:20 1D CC         212 NOTINV    JSR   PICKY         ;get char on screen
CC56:48               213           PHA                 ;preserve it
CC57:8D 7B 07         214           STA   NXTCUR        ;save for update
CC5A:98               215           TYA                 ;test for checkerboard
CC5B:C8               216           INY
CC5C:F0 0D   CC6B     217           BEQ   NOTINV2       ;=>checkerboard, display it
CC5E:7A               218           PLY                 ;test char
CC5F:5A               219           PHY
CC60:30 09   CC6B     220           BMI   NOTINV2       ;don't need inverse
CC62:AD 1E C0         221           LDA   ALTCHARSET    ;mask = $7F if alternate
CC65:09 7F            222           ORA   #$7F          ; character set,
CC67:4A               223           LSR   A             ;$3F if normal char set
CC68:2D FB 07         224 NOTINV1   AND   CURSOR        ;form char to display
CC6B:20 B3 C3         225 NOTINV2   JSR   STORY         ;and display it
CC6E:68               226           PLA                 ;restore real char
CC6F:60               227           RTS
CC70:                 228 *
CC70:                 229 * The UPDATE routine increments the random seed.
CC70:                 230 * If a certain value is reached and we are in Apple II
CC70:                 231 * mode, the blinking check cursor is updated.  If a
CC70:                 232 * key has been pressed, the old char is replaced on the
CC70:                 233 * screen, and we return with BMI.
CC70:                 234 *
CC70:                 235 * NOTE: this routine used by COMM firmware!!
CC70:                 236 *
CC70:48               237 UPDATE    PHA                 ;save char
CC71:E6 4E            238           INC   RNDL          ;update seed
CC73:D0 1E   CC93     239           BNE   UD2           ;check for key
CC75:A5 4F            240           LDA   RNDH
CC77:E6 4F            241           INC   RNDH
CC79:45 4F            242           EOR   RNDH
CC7B:29 10            243           AND   #$10          ;need to update cursor?
CC7D:F0 14   CC93     244           BEQ   UD2           ;=>no, check for key
CC7F:AD FB 07         245           LDA   CURSOR        ;what cursor are we using?
CC82:F0 0F   CC93     246           BEQ   UD2           ;=>//e cursor, leave alone
CC84:5A               247           PHY                 ;+ Save Y
CC85:20 1D CC         248           JSR   PICKY         ;get the character into A
CC88:AC 7B 07         249           LDY   NXTCUR        ;get next character
CC8B:8D 7B 07         250           STA   NXTCUR        ;save next next character
CC8E:98               251           TYA
CC8F:20 B3 C3         252           JSR   STORY         ;and print it
CC92:7A               253           PLY                 ;+
CC93:68               254 UD2       PLA                 ;get real char
CC94:20 E6 C8         255           JSR   XBITKBD       ;was a key pressed?
CC97:10 26   CCBF     256           BPL   GETCURX       ;=>no key pressed
CC99:4C C3 CF         257 CLRKBD    JMP   CLRKBD2       ;+ restore old key look for key and exit
CC9C:EA               258           NOP                 ;+ Keep code alignedkey
CC9D:                 259 *
CC9D:                 260 * ON CURSORS.  Whenever the horizontal cursor position is
CC9D:                 261 * needed, a call to GETCUR is done. This is the equivalent
CC9D:                 262 * of a LDY CH.  This returns the current cursor for II and
```



```
CC9D:                 263 * //e mode, which may have been poked as either CH or OURCH.
CC9D:                 264 *
CC9D:                 265 * It also forces CH and OLDCH to 0 if 80 column mode active.
CC9D:                 266 * This prevents LDY CH, STA (BASL),Y from trashing non screen
CC9D:                 267 * memory. It works just like the //e.
CC9D:                 268 *
CC9D:                 269 * All routines that update the cursor's horizontal position
CC9D:                 270 * are here.  This ensures that the newest value of the cursor
CC9D:                 271 * is always used, and that 80 column CH is always 0.
CC9D:                 272 *
CC9D:                 273 * GETCUR only affects the Y register
CC9D:                 274 *
CC9D:A4 24            275 GETCUR    LDY   CH            ;if CH=OLDCH, then
CC9F:CC 7B 04         276           CPY   OLDCH         ;OURCH is valid
CCA2:D0 03   CCA7     277           BNE   GETCUR1       ;=>else CH must have been changed
CCA4:AC 7B 05         278           LDY   OURCH         ;use OURCH
CCA7:C4 21            279 GETCUR1   CPY   WNDWDTH       ;is the value too big
CCA9:90 02   CCAD     280           BCC   GETCUR2       ;=>no, fits just fine
CCAB:A0 00            281           LDY   #0            ;else force CH to 0
CCAD:                 282 *
CCAD:                 283 * GETCUR2 is commonly used to set the current cursor
CCAD:                 284 * position when Y can be used.
CCAD:                 285 *
CCAD:8C 7B 05         286 GETCUR2   STY   OURCH         ;update real cursor
CCB0:2C 1F C0         287           BIT   RD80VID       ;80 columns?
CCB3:10 02   CCB7     288           BPL   GETCUR3       ;=>no, set all cursors
CCB5:A0 00            289           LDY   #0            ;yes, peg CH to 0
CCB7:84 24            290 GETCUR3   STY   CH
CCB9:8C 7B 04         291           STY   OLDCH
CCBC:AC 7B 05         292           LDY   OURCH         ;get cursor
CCBF:60               293 GETCURX   RTS                 ;and fly...
CCC0:                  55           INCLUDE ESCAPE
```

```
CCC0:                    2 * START AN ESCAPE SEQUENCE:
CCC0:                    3 *  WE HANDLE THE FOLLOWING ONES:
CCC0:                    4 *    @ - HOME & CLEAR
CCC0:                    5 *    A - Cursor right
CCC0:                    6 *    B - Cursor left
CCC0:                    7 *    C - Cursor down
CCC0:                    8 *    D - Cursor up
CCC0:                    9 *    E - CLR TO EOL
CCC0:                   10 *    F - CLR TO EOS
CCC0:                   11 *    I, Up Arrow - CURSOR UP (stay escape)
CCC0:                   12 *    J, Lft Arrow - CURSOR LEFT (stay escape)
CCC0:                   13 *    K, Rt Arrow - CURSOR RIGHT (stay escape)
CCC0:                   14 *    M, Dn Arrow - CURSOR DOWN  (stay escape)
CCC0:                   15 *    4 - GOTO 40 COLUMN MODE
CCC0:                   16 *    8 - GOTO 80 COLUMN MODE
CCC0:                   17 * CTL-D- Disable the printing of control chars
CCC0:                   18 * CTL-E- Enable the printing of control chars
CCC0:                   19 * CTL-Q- QUIT (PR#0/IN#0)
CCC0:                   20 *
CCC0:B9 0C CD           21 ESC3     LDA   ESCCHAR,Y     ;GET CHAR TO "PRINT"
CCC3:5A                 22          PHY                 ;save index
CCC4:20 58 CD           23          JSR   CTLCHAR       ;execute character
CCC7:7A                 24          PLY                 ;restore index
CCC8:C0 08              25          CPY   #YHI          ;If Y<YHI, stay escape
CCCA:B0 21   CCED       26          BCS   ESCRDKEY      ;=>exit escape mode
CCCC:                   27 *
CCCC:                   28 * This is the entry point called by RDKEY iff escapes
CCCC:                   29 * are enabled and an escape is encountered.  The next
CCCC:                   30 * keypress is read and processed.  If it is a key that
CCCC:                   31 * terminates escape mode, a new key is read by ESCRDKEY.
CCCC:                   32 * If escape mode should not be terminated, NEWESC is
CCCC:                   33 * called again.
CCCC:                   34 *
CCCC:20 1D CC           35 NEWESC   JSR   PICKY         ;get current character
CCCF:48                 36          PHA                 ;and save it
CCD0:29 80              37          AND   #$80          ;save invert bit
CCD2:49 AB              38          EOR   #$AB          ;make it inverted "+"
CCD4:20 B3 C3           39          JSR   STORY         ;and pop it on the screen
CCD7:20 E6 C8           40 ESC0     JSR   XBITKBD       ;check for keystroke
CCDA:10 FB   CCD7       41          BPL   ESC0
CCDC:68                 42          PLA                 ;get old char
CCDD:20 99 CC           43          JSR   CLRKBD        ;restore char, get key
CCE0:20 9B C3           44          JSR   UPSHIFT       ;upshift esc char
CCE3:A0 13              45 ESC1     LDY   #ESCNUM       ;COUNT/INDEX
CCE5:D9 F8 CC           46 ESC2     CMP   ESCTAB,Y      ;IS IT A VALID ESCAPE?
CCE8:F0 D6   CCC0       47          BEQ   ESC3          =>yes
CCEA:88                 48          DEY
CCEB:10 F8   CCE5       49          BPL   ESC2          ;TRY 'EM ALL...
CCED:                   50 *
CCED:                   51 * End of escape sequence, read next character.
CCED:                   52 * This is initially called by RDCHAR which is usually called
CCED:                   53 * by GETLN to read characters with escapes enabled.
CCED:                   54 *
CCED:A9 08              55 ESCRDKEY LDA   #M.CTL        ;enable escape sequences
CCEF:1C FB 04           56          TRB   VMODE
CCF2:20 0C FD           57          JSR   RDKEY         ;read char with escapes
CCF5:4C 44 FD           58          JMP   NOESCAPE      ;got the key, disable escapes
CCF8:                   59 *
```



```
CCF8:                   60 * When in escape mode, the characters in ESCTAB (high)
CCF8:                   61 * bits set), are mapped into the characters in ESCCHAR.
CCF8:                   62 * These characters are then executed by a call to CTLCHAR.
CCF8:                   63 *
CCF8:                   64 * CTLCHAR looks up a character in the table starting at
CCF8:                   65 * CTLTAB.  It uses the current index as an index into the
CCF8:                   66 * table of routine addresses, CTLADR.  If the character is
CCF8:                   67 * not in the table, a call to VIDOUT1 is done in case the
CCF8:                   68 * character is BS, LF, CR, or BEL.
CCF8:                   69 *
CCF8:                   70 * NOTE: CTLON and CTLOFF are not accessible except through
CCF8:                   71 * and escape sequence
CCF8:                   72 *
CCF8:                   73          MSB   ON            ;high bit on
CCF8:        CCF8       74 ESCTAB   EQU   *
CCF8:CA                 75          ASC   'J'           ;left (stay esc)
CCF9:88                 76          DFB   $88           ;left arrow (stay esc)
CCFA:CD                 77          ASC   'M'           ;down (stay esc)
CCFB:8B                 78          DFB   $8B           ;up arrow (stay esc)
CCFC:95                 79          DFB   $95           ;right arrow (stay esc)
CCFD:8A                 80          DFB   $8A           ;down arrow (stay esc)
CCFE:C9                 81          ASC   'I'           ;up (stay esc)
CCFF:CB                 82          ASC   'K'           ;right (stay esc)
CD00:        0008       83 YHI      EQU   *-ESCTAB
CD00:C2                 84          ASC   'B'           ;left
CD01:C3                 85          ASC   'C'           ;down
CD02:C4                 86          ASC   'D'           ;up
CD03:C1                 87          ASC   'A'           ;right
CD04:C0                 88          ASC   '@'           ;formfeed
CD05:C5                 89          ASC   'E'           ;clear EOL
CD06:C6                 90          ASC   'F'           ;clear EOS
CD07:B4                 91          ASC   '4'           ;40 column mode
CD08:B8                 92          ASC   '8'           ;80 column mode
CD09:91                 93          DFB   $91           ;CTL-Q = QUIT
CD0A:84                 94          DFB   $84           ;CTL-D ;ctl char disable
CD0B:85                 95          DFB   $85           ;CTL-E ;ctl char enable
CD0C:                   96 *
CD0C:        0013       97 ESCNUM   EQU   *-ESCTAB-1
CD0C:                   98 *
CD0C:        CD0C       99 ESCCHAR  EQU   *             ;list of escape chars
CD0C:88                100          DFB   $88           ;J: BS (stay esc)
CD0D:88                101          DFB   $88           ;<-:BS (stay esc)
CD0E:8A                102          DFB   $8A           ;M: LF (stay esc)
CD0F:9F                103          DFB   $9F           ;UP:US (stay esc)
CD10:9C                104          DFB   $9C           ;->:FS (stay esc)
CD11:8A                105          DFB   $8A           ;DN: LF (stay esc)
CD12:9F                106          DFB   $9F           ;I: UP (stay esc)
CD13:9C                107          DFB   $9C           ;K: RT (stay esc)
CD14:88                108          DFB   $88           ;ESC-B = BS
CD15:        CD15      109 CTLTAB    EQU   *             ;list of control characters
CD15:8A                110          DFB   $8A           ;ESC-C = DN
CD16:9F                111          DFB   $9F           ;ESC-D = UP
CD17:9C                112          DFB   $9C           ;ESC-A = RT
CD18:8C                113          DFB   $8C           ;@: Formfeed
CD19:9D                114          DFB   $9D           ;E: CLREOL
CD1A:8B                115          DFB   $8B           ;F: CLREOP
CD1B:91                116          DFB   $91           ;SET40
CD1C:92                117          DFB   $92           ;SET80
```

```
CD1D:95              118           DFB   $95           ;QUIT
CD1E:04              119           DFB   $04           ;Disable controls (escape only)
CD1F:05              120           DFB   $05           ;Enable controls (escape only)
CD20:                121 * escape chars end here
CD20:85              122           DFB   $85           ;X.CUR.ON
CD21:86              123           DFB   $86           ;X.CUR.OFF
CD22:8E              124           DFB   $8E           ;Normal
CD23:8F              125           DFB   $8F           ;Inverse
CD24:96              126           DFB   $96           ;Scroll down
CD25:97              127           DFB   $97           ;Scroll up
CD26:98              128           DFB   $98           ;mouse chars off
CD27:99              129           DFB   $99           ;home cursor
CD28:9A              130           DFB   $9A           ;clear line
CD29:9B              131           DFB   $9B           ;mouse chars on
CD2A:                132 *
CD2A:        0014    133 CTLNUM    EQU   *-CTLTAB-1
CD2A:                134 *
CD2A:        CD2A    135 CTLADR    EQU   *
CD2A:66 FC           136           DW    LF            ;move cursor down
CD2C:1A FC           137           DW    UP            ;move cursor up
CD2E:A0 FB           138           DW    NEWADV        ;forward a space
CD30:58 FC           139           DW    HOME          ;home cursor, clear screen
CD32:9C FC           140           DW    CLREOL        ;clear to end of line
CD34:42 FC           141           DW    CLREOP        ;clear to end of page
CD36:C0 CD           142           DW    SET40         ;set 40 column mode
CD38:BE CD           143           DW    SET80         ;set 80 column mode
CD3A:45 CE           144           DW    QUIT          ;Quit video firmware
CD3C:91 CD           145           DW    CTLOFF        ;disable //e control chars
CD3E:95 CD           146           DW    CTLON         ;enable //e control chars
CD40:89 CD           147           DW    X.CUR.ON      ;turn on cursor (pascal)
CD42:8D CD           148           DW    X.CUR.OFF     ;turn off cursor (pascal)
CD44:B0 CD           149           DW    X.SO          ;normal video
CD46:B7 CD           150           DW    X.SI          ;inverse video
CD48:30 CB           151           DW    SCROLLDN      ;scroll down a line
CD4A:35 CB           152           DW    SCROLLUP      ;scroll up a line
CD4C:9F CD           153           DW    MOUSOFF       ;disable mouse characters
CD4E:A5 CD           154           DW    HOMECUR       ;move cursor home
CD50:A0 FC           155           DW    CLRLIN        ;clear current line
CD52:99 CD           156           DW    MOUSON        ;enable mouse characters
CD54:                157 *
CD54:                158           MSB   ON
CD54:                159 *
CD54:                160 * CTLCHAR executes the control character in the
CD54:                161 * accumulator.  If it is called by Pascal, the character
CD54:                162 * is always executed.  If it is called by the video
CD54:                163 * firmware, the character is executed if M.CTL is set
CD54:                164 * and M.CTL2 is clear.
CD54:                165 *
CD54:                166 * Note: This routine is only called if the video firmware
CD54:                167 * is active. The Monitor ROM calls VIDOUT1 if the video
CD54:                168 * firmware is inactive.
CD54:                169 *
CD54:2C C1 CB        170 CTLCHAR0  BIT   SEV1          ;set V (use M.CTL)
CD57:50              171           DFB   $50           ;BVC opcode (never taken)
CD58:                172 *
CD58:B8              173 CTLCHAR   CLV                 ;Always do control character
CD59:DA              174           PHX                 ;save X
CD5A:8D F8 04        175           STA   TEMP1         ;temp save of A
```



```
CD5D:20 04 FC        176           JSR   VIDOUT1      ;try to execute CR, LF, BS, or BEL
CD60:CD F8 04        177           CMP   TEMP1         ;if acc has changed
CD63:D0 0A   CD6F    178           BNE   CTLDONE       ;then function done
CD65:A2 14           179           LDX   #CTLNUM       ;number of CTL chars
CD67:DD 15 CD        180 FNDCTL    CMP   CTLTAB,X      ;is it in table
CD6A:F0 05   CD71    181           BEQ   CTLGO         ;=>yes, should we execute?
CD6C:CA              182           DEX                 ;else check next
CD6D:10 F8   CD67    183           BPL   FNDCTL        ;=>try next one
CD6F:FA              184 CTLDONE   PLX                 ;restore X
CD70:60              185           RTS                 ;and return
CD71:                186 *
CD71:48              187 CTLGO     PHA                 ;save A
CD72:50 0C   CD80    188           BVC   CTLGO1        ;V clear, always do (pascal,escape)
CD74:AD FB 04        189           LDA   VMODE         ;controls are enabled iff
CD77:29 28           190           AND   #M.CTL+M.CTL2 ; M.CTL = 1 and
CD79:49 08           191           EOR   #M.CTL        ; M.CTL2 = 0
CD7B:F0 03   CD80    192           BEQ   CTLGO1        ;=>they're enabled!!
CD7D:68              193 CGO       PLA                 ;restore A
CD7E:FA              194           PLX                 ;restore X
CD7F:60              195           RTS                 ;and return
CD80:                196 *
CD80:8A              197 CTLGO1    TXA                 ;double X as index
CD81:0A              198           ASL   A             ;into address table
CD82:AA              199           TAX
CD83:68              200           PLA                 ;restore A
CD84:20 A4 FC        201           JSR   CTLDO         ;execute the char
CD87:FA              202           PLX                 ;restore X
CD88:60              203           RTS                 ;and return
CD89:                204 *
CD89:                205 * X.CUR.ON = Allow Pascal cursor display
CD89:                206 * X.CUR.OFF = Disable Pascal cursor display
CD89:                207 * Cursor is not displayed during call, so it will
CD89:                208 * be right when "redisplayed".
CD89:                209 * Note: Though these commands are executed from BASIC,
CD89:                210 * they have no effect on firmware operation.
CD89:                211 *
CD89:A9 10           212 X.CUR.ON  LDA   #M.CURSOR     ;clear cursor bit
CD8B:80 0E   CD9B    213           BRA   CLRIT
CD8D:                214 *
CD8D:A9 10           215 X.CUR.OFF LDA   #M.CURSOR     ;set cursor bit
CD8F:80 10   CDA1    216           BRA   SETIT
CD91:                217 *
CD91:                218 * The control characters other than CR,LF,BEL,BS
CD91:                219 * are normally enabled when video firmware is active.
CD91:                220 * They can be disabled and enabled using the ESC-D
CD91:                221 * and ESC-E escape sequences.
CD91:                222 *
CD91:A9 20           223 CTLOFF    LDA   #M.CTL2       ;disable control characters
CD93:80 0C   CDA1    224           BRA   SETIT         ;by setting M.CTL2
CD95:                225 *
CD95:A9 20           226 CTLON     LDA   #M.CTL2       ;enable control characters
CD97:80 02   CD9B    227           BRA   CLRIT         ;by clearing M.CTL2
CD99:                228 *
CD99:                229 * Enable mouse text by clearing M.MOUSE
CD99:                230 *
CD99:A9 01           231 MOUSON    LDA   #M.MOUSE
CD9B:1C FB 04        232 CLRIT     TRB   VMODE
CD9E:60              233           RTS
```

```
CD9F:                   *
CD9F:                   * Disable mouse text by setting M.MOUSE
CD9F:                   *
CD9F:A9 01              MOUSOFF LDA   #M.MOUSE
CDA1:0C FB 04           SETIT   TSB   VMODE
CDA4:60                 RTS
CDA5:                   *
CDA5:                   * EXECUTE HOME:
CDA5:                   *
CDA5:20 E9 FE           HOMECUR JSR   CLRCH           ;move cursors to far left
CDA8:A8                 TAY                           ;(probably not needed)
CDA9:A5 22              LDA   WNDTOP                  ;and to top of window
CDAB:85 25              STA   CV
CDAD:4C 88 FC           JMP   NEWVTABZ                ;then set base address, OURCV
CDB0:                   *
CDB0:                   * EXECUTE "NORMAL VIDEO"
CDB0:                   *
CDB0:20 84 FE           X.SO    JSR   SETNORM         ;set INVFLG to $FF
CDB3:A9 04              LDA   #M.VMODE                ;then clear inverse mode bit
CDB5:80 E4   CD9B       BRA   CLRIT
CDB7:                   *
CDB7:                   * EXECUTE "INVERSE VIDEO"
CDB7:                   *
CDB7:20 80 FE           X.SI    JSR   SETINV          ;set INVFLG to $3F
CDBA:A9 04              LDA   #M.VMODE                ;then set inverse mode bit
CDBC:80 E3   CDA1       BRA   SETIT
CDBE:                   *
CDBE:                   * EXECUTE '40COL MODE' or '80COL MODE':
CDBE:                   *
CDBE:38                 SET80   SEC                   ;flag an 80 column window
CDBF:90                 DFB   $90                     ;BCC opcode (never taken)
CDC0:18                 SET40   CLC                   ;flag a 40 column window
CDC1:2C FB 04           BIT   VMODE                   ;but...is it pascal?
CDC4:10 54   CE1A       BPL   SETX                    ;=>yes, don't execute
CDC6:08                 PHP                           ;save window size
CDC7:20 1B CE           JSR   HOOKITUP                ;COPYROM if needed, set I/O hooks
CDCA:28                 PLP                           ;and get 40/80
CDCB:80 08   CDD5       BRA   WIN0                    ;=>set window
CDCD:                   *
CDCD:                   * CHK80 is called by PR#0 to convert to 40 if it was
CDCD:                   * 80. Otherwise the window is left ajar.
CDCD:                   *
CDCD:2C 1F C0           CHK80   BIT   RD80VID         ;don't set 40 if
CDD0:10 48   CE1A       BPL   SETX                    ;already 40
CDD2:                   *
CDD2:18                 WIN40   CLC                   ;flag 40 column window
CDD3:B0                 DFB   $B0                     ;BCS opcode (never taken)
CDD4:38                 WIN80   SEC                   ;flag 80 column window
CDD5:64 22              WIN0    STZ   WNDTOP          ;set window top now
CDD7:2C 1A C0           BIT   RDTEXT                  ;for text or mixed
CDDA:30 04   CDE0       BMI   WIN1                    ;=>text
CDDC:A9 14              LDA   #20
CDDE:85 22              STA   WNDTOP                  ;used by 80<->40 conversion
CDE0:2C 1F C0           WIN1    BIT   RD80VID         ;80 columns now?
CDE3:08                 PHP                           ;save 80 or 40
CDE4:B0 07   CDED       BCS   WIN2                    ;=>80: convert if 40
CDE6:10 0A   CDF2       BPL   WIN3                    ;=>40: no convert
CDE8:20 53 CE           JSR   SCRN84                  ;80: convert to 40
```



```
CDEB:80 05   CDF2       BRA   WIN3            ;done converting
CDED:30 03   CDF2       BMI   WIN3            ;=>80: no convert
CDEF:20 80 CE           JSR   SCRN48          ;40: convert to 80
CDF2:20 9D CC           JSR   GETCUR          ;determine absolute CH
CDF5:98                 TYA                   ;in case the window setting
CDF6:18                 CLC                   ;was different
CDF7:65 20              ADC   WNDLFT
CDF9:28                 PLP                   ;pin to right edge if
CDFA:B0 06   CE02       BCS   WIN4            ;80 to 40 leaves cursor
CDFC:C9 28              CMP   #40             ;off the screen
CDFE:90 02   CE02       BCC   WIN4
CE00:A9 27              LDA   #39
CE02:20 EC FE           WIN4    JSR   SETCUR  ;set new cursor
CE05:A5 25              LDA   CV              ;set new base address
CE07:20 C1 FB           JSR   BASCALC         ;for left = 0 (always)
CE0A:                   *
CE0A:64 20              WNDREST STZ   WNDLFT  ;Called by INIT and Pascal
CE0C:A9 18              LDA   #$18            ;and bottom
CE0E:85 23              STA   WNDBTM
CE10:A9 28              LDA   #$28            ;set left,width,bottom
CE12:2C 1F C0           BIT   RD80VID         ;set width to 80 if 80 columns
CE15:10 01   CE18       BPL   WIN5
CE17:0A                 ASL   A
CE18:85 21              WIN5    STA   WNDWDTH ;set width
CE1A:60                 SETX    RTS           ;exit used by SET40/80
CE1B:                   *
CE1B:                   * Turn on video firmware:
CE1B:                   *
CE1B:                   * This routine is used by BASIC init, ESC-4, ESC-8
CE1B:                   * It copies the Monitor ROM to the language card
CE1B:                   * if necessary; it sets the input and output hooks to
CE1B:                   * $C30x; it sets all switches for video firmware operation
CE1B:                   *
CE1B:2C 7B 06           HOOKITUP BIT  VFACTV  ;don't touch hooks
CE1E:10 11   CE31       BPL   VIDMODE         ;if video firmware already active
CE20:20 38 C3           JSR   COPYROM         ;Copy ROM to LC?
CE23:A9 05              SETHOOKS LDA  #>C3KEYIN ;set up $C300 hooks
CE25:85 38              STA   KSWL
CE27:A9 07              LDA   #>C3COUT1
CE29:85 36              STA   CSWL
CE2B:A9 C3              LDA   #<C3COUT1
CE2D:85 39              STA   KSWH
CE2F:85 37              STA   CSWH
CE31:                   *
CE31:                   * Now set the video firmware active
CE31:                   *
CE31:9C FB 07           VIDMODE STZ   CURSOR  ;set a solid inverse cursor
CE34:A9 08              LDA   #M.CTL          ;preserve M.CTL bit
CE36:2D FB 04           AND   VMODE
CE39:09 81              ORA   #M.PASCAL+M.MOUSE ;no pascal,mouse
CE3B:                   *
CE3B:                   * Pascal calls here to set its mode
CE3B:                   *
CE3B:8D FB 04           PVMODE  STA   VMODE   ;set mode bits
CE3E:9C 7B 06           STZ   VFACTV          ;say video firmware active
CE41:8D 0F C0           STA   SETALTCHAR      ;and set alternate char set
CE44:60                 QX      RTS
CE45:                   *
```

```
CE45:                   * QUIT converts the screen from 80 to 40 if necessary,
CE45:                   * sets a 40 column window, and restores the normal I/O
CE45:                   * hooks (COUT1 and KEYIN).
CE45:                   *
CE45:2C FB 04           QUIT    BIT   VMODE       ;no quitting from pascal
CE48:10 FA   CE44               BPL   QX
CE4A:20 D2 CD                   JSR   WIN40       ;first, do an escape 4
CE4D:20 89 FE           ZZQUIT  JSR   SETKBD      ;do a IN#0  (used by COMM)
CE50:4C 93 FE                   JMP   SETVID      ;and a PR#0
```



```
CE53:                   *
CE53:                   * SCRN84 and SCRN48 convert screens between 40 & 80 cols.
CE53:                   * WNDTOP must be set up to indicate the last line to
CE53:                   * be done.  All registers are trashed.
CE53:                   *
CE53:A2 17              SCRN84  LDX   #23          ;start at bottom of screen
CE55:8D 01 C0                   STA   SET80COL     ;allow page 2 access
CE58:8A                 SCR1    TXA                ;calc base for line
CE59:20 C1 FB                   JSR   BASCALC
CE5C:A0 27                      LDY   #39          ;start at right of screen
CE5E:5A                 SCR2    PHY                ;save 40 index
CE5F:98                         TYA                ;div by 2 for 80 column index
CE60:4A                         LSR   A
CE61:B0 03   CE66               BCS   SCR3
CE63:2C 55 C0                   BIT   TXTPAGE2     ;even column, do page 2
CE66:A8                 SCR3    TAY                ;get 80 index
CE67:B1 28                      LDA   (BASL),Y     ;get 80 char
CE69:2C 54 C0                   BIT   TXTPAGE1     ;restore page1
CE6C:7A                         PLY                ;get 40 index
CE6D:91 28                      STA   (BASL),Y
CE6F:88                         DEY
CE70:10 EC   CE5E               BPL   SCR2         ;do next 40 byte
CE72:CA                         DEX                ;do next line
CE73:30 04   CE79               BMI   SCR4         ;=>done with setup
CE75:E4 22                      CPX   WNDTOP       ;at top yet?
CE77:B0 DF   CE58               BCS   SCR1
CE79:8D 00 C0           SCR4    STA   CLR80COL     ;clear 80STORE for 40 columns
CE7C:8D 0C C0                   STA   CLR80VID     ;clear 80VID for 40 columns
CE7F:60                         RTS
CE80:                   *
CE80:A2 17              SCRN48  LDX   #23          ;start at bottom of screen
CE82:8A                 SCR5    TXA                ;set base for current line
CE83:20 C1 FB                   JSR   BASCALC
CE86:A0 00                      LDY   #0           ;start at left of screen
CE88:8D 01 C0                   STA   SET80COL     ;enable page2 store
CE8B:B1 28              SCR6    LDA   (BASL),Y     ;get 40 column char
CE8D:5A                 SCR8    PHY                ;save 40 column index
CE8E:48                         PHA                ;save char
CE8F:98                         TYA                ;div 2 for 80 column index
CE90:4A                         LSR   A
CE91:B0 03   CE96               BCS   SCR7         ;save on page1
CE93:8D 55 C0                   STA   TXTPAGE2
CE96:A8                 SCR7    TAY                ;get 80 column index
CE97:68                         PLA                ;now save character
CE98:91 28                      STA   (BASL),Y
CE9A:8D 54 C0                   STA   TXTPAGE1     ;flip page1
CE9D:7A                         PLY                ;restore 40 column index
CE9E:C8                         INY                ;move to the right
CE9F:C0 28                      CPY   #40          ;at right yet?
CEA1:90 E8   CE8B               BCC   SCR6         ;=>no, do next column
CEA3:20 CF CB                   JSR   CLRHALF      ;clear half of screen
CEA6:CA                         DEX                ;else do next line of screen
CEA7:30 04   CEAD               BMI   SCR9         ;=>done with top line
CEA9:E4 22                      CPX   WNDTOP       ;at top yet?
CEAB:B0 D5   CE82               BCS   SCR5
CEAD:8D 0D C0           SCR9    STA   SET80VID     ;convert to 80 columns
CEB0:60                         RTS
CEB1:                    56     INCLUDE PASCAL     ;Pascal support stuff
```

```
19 PASCAL          Video firmware Pascal stuff          20-OCT-86  06:41 PAGE 63

CEB1:AA              3 PSTATUS   TAX                 ;is request code = 0?
CEB2:F0 08   CEBC    4           BEQ   PIORDY        ;=>yes, ready for output
CEB4:CA              5           DEX                 ;check for any input
CEB5:D0 07   CEBE    6           BNE   PSTERR        ;=>bad request, return error
CEB7:20 E6 C8        7           JSR   XBITKBD       ;test keyboard
CEBA:10 04   CEC0    8           BPL   PNOTRDY       ;=>no keystroked
CEBC:38              9 PIORDY    SEC                 ;good return
CEBD:60             10           RTS
CEBE:A2 03          11 PSTERR    LDX   #3            ;else flag error
CEC0:18             12 PNOTRDY   CLC
CEC1:60             13           RTS
CEC2:               14 *
CEC2:               15 * PASCAL OUTPUT:
CEC2:               16 *
CEC2:        CEC2   17 PWRITE    EQU   *
CEC2:09 80          18           ORA   #$80          ;turn on high bit
CEC4:AA             19           TAX                 ;save character
CEC5:20 54 CF       20           JSR   PSETUP2       ;SETUP ZP STUFF, don't set ROM
CEC8:A9 08          21           LDA   #M.GOXY       ;ARE WE DOING GOTOXY?
CECA:2C FB 04       22           BIT   VMODE
CECD:D0 2B   CEFA   23           BNE   GETX          ;=>Doing X or Y?
CECF:8A             24           TXA                 ;now check for control char
CED0:89 60          25           BIT   #$60          ;is it control?
CED2:F0 45   CF19   26           BEQ   PCTL          ;=>yes, do control
CED4:AC 7B 05       27           LDY   OURCH         ;get horizontal position
CED7:24 32          28           BIT   INVFLG        ;check for inverse
CED9:30 02   CEDD   29           BMI   PWR1          ;normal, go store it
CEDB:29 7F          30           AND   #$7F
CEDD:20 C1 C3       31 PWR1      JSR   STORE         ;now store it (erasing cursor)
CEE0:C8             32           INY                 ;INC CH
CEE1:8C 7B 05       33           STY   OURCH
CEE4:C4 21          34           CPY   WNDWDTH
CEE6:90 0C   CEF4   35           BCC   PWRET
CEE8:20 60 C3       36           JSR   SETROM
CEEB:20 E9 FE       37           JSR   CLRCH         ;set cursor position to 0
CEEE:20 66 FC       38           JSR   LF
CEF1:20 54 C3       39 PWRITERET JSR   RESETLC
CEF4:20 0B CC       40 PWRET     JSR   PASINVERT     ;display new cursor
CEF7:A2 00          41 PRET      LDX   #$0           ;return with no error
CEF9:60             42           RTS
CEFA:               43 *
CEFA:               44 *  HANDLE GOTOXY STUFF:
CEFA:               45 *
CEFA:        CEFA   46 GETX      EQU   *
CEFA:20 0B CC       47           JSR   PASINVERT     ;turn off cursor
CEFD:8A             48           TXA                 ;get character
CEFE:38             49           SEC
CEFF:E9 A0          50           SBC   #160          ;MAKE BINARY
CF01:2C FB 06       51           BIT   XCOORD        ;doing X?
CF04:30 2A   CF30   52           BMI   PSETX         ;=>yes, set it
CF06:               53 *
CF06:               54 * Set Y and do the GOTOXY
CF06:               55 *
CF06:        CF06   56 GETY      EQU   *
CF06:8D FB 05       57           STA   OURCV
CF09:20 71 CF       58           JSR   PASCALC       ;calc base addr
CF0C:AC FB 06       59           LDY   XCOORD
CF0F:20 AD CC       60           JSR   GETCUR2       ;set proper cursors
```

```
19 PASCAL          Video firmware Pascal stuff          20-OCT-86  06:41 PAGE 64

CF12:A9 08          61           LDA   #M.GOXY       ;turn off gotoxy
CF14:1C FB 04       62           TRB   VMODE
CF17:80 DB   CEF4   63           BRA   PWRET         ;=>DONE (ALWAYS TAKEN)
CF19:               64 *
CF19:20 0B CC       65 PCTL      JSR   PASINVERT     ;turn off cursor
CF1C:8A             66           TXA                 ;get char
CF1D:C9 9E          67           CMP   #$9E          ;is it gotoXY?
CF1F:F0 08   CF29   68           BEQ   STARTXY       ;=>yes, start it up
CF21:20 60 C3       69           JSR   SETROM        ;must switch in ROM for controls
CF24:20 58 CD       70           JSR   CTLCHAR       ;EXECUTE IT IF POSSIBLE
CF27:80 C8   CEF1   71           BRA   PWRITERET     ;=>display new cursor, exit
CF29:               72 *
CF29:               73 * START THE GOTOXY SEQUENCE:
CF29:               74 *
CF29:        CF29   75 STARTXY   EQU   *
CF29:A9 08          76           LDA   #M.GOXY
CF2B:0C FB 04       77           TSB   VMODE         ;turn on gotoxy
CF2E:A9 FF          78           LDA   #$FF          ;set XCOORD to -1
CF30:8D FB 06       79 PSETX     STA   XCOORD        ;set X
CF33:80 BF   CEF4   80           BRA   PWRET         ;=>display cursor and exit
CF35:               81 *
CF35:               82 * PASCAL INPUT:
CF35:               83 *
CF35:20 54 CF       84 PASREAD   JSR   PSETUP2       ;SETUP ZP STUFF
CF38:20 D5 C8       85 GKEY      JSR   XRDKBD        ;key pressed?
CF3B:10 FB   CF38   86           BPL   GKEY          ;=>not yet
CF3D:29 7F          87           AND   #$7F          ;DROP HI BIT
CF3F:80 B6   CEF7   88           BRA   PRET          ;good exit
CF41:               89 *
CF41:               90 * PASCAL INITIALIZATION:
CF41:               91 *
CF41:        CF41   92 PINIT     EQU   *
CF41:A9 01          93           LDA   #M.MOUSE      ;Set mode to pascal
CF43:20 3B CE       94           JSR   PVMODE        ;without mouse characters
CF46:20 51 CF       95           JSR   PSETUP        ;setup zero page for pascal
CF49:20 D4 CD       96           JSR   WIN80         ;do 40->80 convert
CF4C:20 58 FC       97           JSR   HOME          ;home and clear screen
CF4F:80 A0   CEF1   98           BRA   PWRITERET     ;display cursor, set OURCH,OURCV...
CF51:               99 *
CF51:        CF51  100 PSETUP    EQU   *
CF51:20 60 C3      101           JSR   SETROM        ;save LC state, set ROM read
CF54:64 22         102 PSETUP2   STZ   WNDTOP        ;set top to 0
CF56:20 0A CE      103           JSR   WNDREST       ;init either 40 or 80 window
CF59:A9 FF         104           LDA   #$FF          ;assume normal text
CF5B:85 32         105           STA   INVFLG
CF5D:A9 04         106           LDA   #M.VMODE      ;is it
CF5F:2C FB 04      107           BIT   VMODE
CF62:F0 02   CF66  108           BEQ   PS1           ;=>yes
CF64:46 32         109           LSR   INVFLG        ;no, make flag inverse
CF66:AC 7B 05      110 PS1       LDY   OURCH
CF69:20 AD CC      111           JSR   GETCUR2       ;set all cursors
CF6C:AD FB 05      112           LDA   OURCV
CF6F:85 25         113           STA   CV
CF71:              114 *
CF71:              115 * Put BASCALC here so we don't have to switch
CF71:              116 * in the ROMs for each character output.
CF71:              117 *
CF71:0A            118 PASCALC   ASL   A
```

```
CF72:A8          119           TAY             ;calc base addr in BASL,H
CF73:4A          120           LSR   A         ;for given line no.
CF74:4A          121           LSR   A
CF75:29 03       122           AND   #$03      ; 0<=line no.<=$17
CF77:09 04       123           ORA   #$4       ; arg=000ABCDE, generate
CF79:85 29       124           STA   BASH      ; BASH=000001CD
CF7B:98          125           TYA             ;and
CF7C:6A          126           ROR   A         ; BASL=EABAB000
CF7D:29 98       127           AND   #$98
CF7F:85 28       128 PASCLC2   STA   BASL
CF81:0A          129           ASL   A
CF82:0A          130           ASL   A
CF83:04 28       131           TSB   BASL
CF85:60          132           RTS
CF86:             57           include moremisc   ;More random junk
```



```
CF86:             2 ************************************************
CF86:             3 * here are more miscellaneous routines stuffed here in a
CF86:             4 * valiant effort to make other code align properly.
CF86:             5 ************************************************

CF86:             7 * various tables

CF86:83 8B 8B     9 irqtble   dfb   >lcbank2,>lcbank1,>lcbank1
CF89:05 03 55    10           dfb   >wrcardram,>rdcardram,>txtpage2

CF8C:9E 0B 40 50 12 comtbl    dfb   $9E,$0B,$40,$50,$16,$0B,$01,$00

CF94:CD C1 D8 D9 14 rtbl      asc   'MAXYPS'

CF9A:            16 ************************************************
CF9A:            17 * MOVEIRQ - This routine transfers the roms interrupt vector into
CF9A:            18 *          both language cards
CF9A:            19 ************************************************

CF9A:20 A0 CF    21 m.oveirq  jsr   moveirq
CF9D:4C 84 C7    22           jmp   swrts

CFA0:20 60 C3    24 moveirq   JSR   SETROM       ;Read ROM and Write to RAM
CFA3:AD 16 C0    25           LDA   RDALTZP      ;Which language card?
CFA6:0A          26           ASL   A            ;C=1 if alternate card
CFA7:A0 01       27           LDY   #1           ;Move two bytes
CFA9:B9 FE FF    28 MIRQLP    LDA   IRQVECT,Y    ;Get byte from ROM
CFAC:8D 09 C0    29           STA   SETALTZP     ;Set alternate card
CFAF:99 FE FF    30           STA   IRQVECT,Y    ;Store it in the RAM card
CFB2:8D 08 C0    31           STA   SETSTDZP     ;Set main card
CFB5:99 FE FF    32           STA   IRQVECT,Y
CFB8:88          33           DEY
CFB9:10 EE  CFA9 34           BPL   MIRQLP       ;Go do the second byte
CFBB:90 03  CFC0 35           BCC   MIRQSTD      ;Is the card set right?
CFBD:8D 09 C0    36           STA   SETALTZP     ;No, it wasn't
CFC0:4C 54 C3    37 MIRQSTD   JMP   RESETLC      ;Clean up & go home

CFC3:            39 ************************************************
CFC3:            40 * CLRKBD2 - Moved here from scrolling routines
CFC3:            41 ************************************************

CFC3:5A          43 clrkbd2   phy                 ;Now preserves Y
CFC4:20 B3 C3    44           jsr   story
CFC7:7A          45           ply
CFC8:4C D5 C8    46           jmp   xrdkbd

CFCB:            48 ************************************************
CFCB:            49 * LOOKASC - addition to monitor input routine, if a quote (')
CFCB:            50 * in input, the ascii of the next is input like a hex number
CFCB:            51 ************************************************

CFCB:B0 11  CFDE 53 lookasc   bcs   ladig        ;Was char a hex digit?
CFCD:C9 A0       54           cmp   #$A0         ;Is it a quote
CFCF:D0 13  CFE4 55           bne   ladone       ;Done if not
CFD1:B9 00 02    56           lda   inbuf,y      ;Get next char
CFD4:A2 07       57           ldx   #7           ;for shifting asc into A2L and A2H
CFD6:C9 8D       58           cmp   #$8D         ;Was it a cr?
CFD8:F0 07  CFE1 59           beq   lacr         ;Go handle cr
```

```
20 MOREMISC       Video firmware Pascal stuff        20-OCT-86  06:41 PAGE 67

CFDA:C8             60           iny                 ;Advance index into inbuf
CFDB:4C 90 FF       61           jmp   nxtbit        ;Go shift it in
CFDE:4C 8A FF       62 ladig     jmp   dig
CFE1:4C A7 FF       63 lacr      jmp   getnum
CFE4:60             64 ladone    rts
CFE5:       001B    58           ds    $D000-*,0
----- NEXT OBJECT FILE NAME IS FIRM.1
F800:       F800    59           ORG   F8ORG
F800:               60           INCLUDE AUTOST1     ;F8 monitor rom
```

```
21 AUTOST1        Apple //c F8 monitor firmware      20-OCT-86  06:41 PAGE 68

F800:4A             3 PLOT       LSR   A             ;Y-COORD/2
F801:08             4            PHP                 ;SAVE LSB IN CARRY
F802:20 47 F8       5            JSR   GBASCALC      ;CALC BASE ADR IN GBASL,H
F805:28             6            PLP                 ;RESTORE LSB FROM CARRY
F806:A9 0F          7            LDA   #$0F          ;MASK $0F IF EVEN
F808:90 02   F80C   8            BCC   RTMASK
F80A:69 E0          9            ADC   #$E0          ;MASK $F0 IF ODD
F80C:85 2E         10 RTMASK     STA   MASK
F80E:B1 26         11 PLOT1      LDA   (GBASL),Y     ;DATA
F810:45 30         12            EOR   COLOR         ; XOR COLOR
F812:25 2E         13            AND   MASK          ;  AND MASK
F814:51 26         14            EOR   (GBASL),Y     ;   XOR DATA
F816:91 26         15            STA   (GBASL),Y     ;    TO DATA
F818:60            16            RTS
F819:              17 *
F819:20 00 F8      18 HLINE      JSR   PLOT          ;PLOT SQUARE
F81C:C4 2C         19 HLINE1     CPY   H2            ;DONE?
F81E:B0 11   F831  20            BCS   RTS1          ; YES, RETURN
F820:C8            21            INY                 ; NO, INCR INDEX (X-COORD)
F821:20 0E F8      22            JSR   PLOT1         ;PLOT NEXT SQUARE
F824:90 F6   F81C  23            BCC   HLINE1        ;ALWAYS TAKEN
F826:69 01         24 VLINEZ     ADC   #$01          ;NEXT Y-COORD
F828:48            25 VLINE      PHA                 ; SAVE ON STACK
F829:20 00 F8      26            JSR   PLOT          ; PLOT SQUARE
F82C:68            27            PLA
F82D:C5 2D         28            CMP   V2            ;DONE?
F82F:90 F5   F826  29            BCC   VLINEZ        ; NO, LOOP.
F831:60            30 RTS1       RTS
F832:              31 *
F832:A0 2F         32 CLRSCR     LDY   #$2F          ;MAX Y, FULL SCRN CLR
F834:D0 02   F838  33            BNE   CLRSC2        ;ALWAYS TAKEN
F836:A0 27         34 CLRTOP     LDY   #$27          ;MAX Y, TOP SCRN CLR
F838:84 2D         35 CLRSC2     STY   V2            ;STORE AS BOTTOM COORD
F83A:              36 ;                              FOR VLINE CALLS
F83A:A0 27         37            LDY   #$27          ;RIGHTMOST X-COORD (COLUMN)
F83C:A9 00         38 CLRSC3     LDA   #$00          ;TOP COORD FOR VLINE CALLS
F83E:85 30         39            STA   COLOR         ;CLEAR COLOR (BLACK)
F840:20 28 F8      40            JSR   VLINE         ;DRAW VLINE
F843:88            41            DEY                 ;NEXT LEFTMOST X-COORD
F844:10 F6   F83C  42            BPL   CLRSC3        ;LOOP UNTIL DONE.
F846:60            43            RTS
F847:              44 *
F847:48            45 GBASCALC   PHA                 ;FOR INPUT  00DEFGH
F848:4A            46            LSR   A
F849:29 03         47            AND   #$03
F84B:09 04         48            ORA   #$04          ;GENERATE GBASH=000001FG
F84D:85 27         49            STA   GBASH
F84F:68            50            PLA                 ;AND GBASL=HDEDE000
F850:29 18         51            AND   #$18
F852:90 02   F856  52            BCC   GBCALC
F854:69 7F         53            ADC   #$7F
F856:85 26         54 GBCALC     STA   GBASL
F858:0A            55            ASL   A
F859:0A            56            ASL   A
F85A:05 26         57            ORA   GBASL
F85C:85 26         58            STA   GBASL
F85E:60            59            RTS
F85F:              60 *
```

```
F85F:A5 30          61 NXTCOL   LDA   COLOR          ;INCREMENT COLOR BY 3
F861:18             62          CLC
F862:69 03          63          ADC   #$03
F864:29 0F          64 SETCOL   AND   #$0F          ;SETS COLOR=17*A MOD 16
F866:85 30          65          STA   COLOR
F868:0A             66          ASL   A             ;BOTH HALF BYTES OF COLOR EQUAL
F869:0A             67          ASL   A
F86A:0A             68          ASL   A
F86B:0A             69          ASL   A
F86C:05 30          70          ORA   COLOR
F86E:85 30          71          STA   COLOR
F870:60             72          RTS
F871:               73 *
F871:4A             74 SCRN     LSR   A             ;READ SCREEN Y-COORD/2
F872:08             75          PHP                 ;SAVE LSB (CARRY)
F873:20 47 F8       76          JSR   GBASCALC      ;CALC BASE ADDRESS
F876:B1 26          77          LDA   (GBASL),Y     ;GET BYTE
F878:28             78          PLP                 ;RESTORE LSB FROM CARRY
F879:90 04   F87F   79 SCRN2    BCC   RTMSKZ        ;IF EVEN, USE LO H
F87B:4A             80          LSR   A
F87C:4A             81          LSR   A
F87D:4A             82          LSR   A             ;SHIFT HIGH HALF BYTE DOWN
F87E:4A             83          LSR   A
F87F:29 0F          84 RTMSKZ   AND   #$0F          ;MASK 4-BITS
F881:60             85          RTS
F882:               86 *
F882:A6 3A          87 INSDS1   LDX   PCL           ;PRINT PCL,H
F884:A4 3B          88          LDY   PCH
F886:20 96 FD       89          JSR   PRYX2
F889:20 48 F9       90          JSR   PRBLNK        ;FOLLOWED BY A BLANK
F88C:A1 3A          91          LDA   (PCL,X)       ;GET OPCODE
F88E:A8             92 INSDS2   TAY                  ;Lable moved down 1
F88F:4A             93          LSR   A             ;EVEN/ODD TEST
F890:90 05   F897   94          BCC   IEVEN
F892:6A             95          ROR   A             ;BIT 1 TEST
F893:B0 0C   F8A1   96          BCS   ERR           ;XXXXXX11 INVALID OP
F895:29 87          97          AND   #$87          ;MASK BITS
F897:4A             98 IEVEN    LSR   A             ;LSB INTO CARRY FOR L/R TEST
F898:AA             99          TAX
F899:BD 62 F9      100          LDA   FMT1,X        ;GET FORMAT INDEX BYTE
F89C:20 79 F8      101          JSR   SCRN2         ;R/L H-BYTE ON CARRY
F89F:D0 04   F8A5  102          BNE   GETFMT
F8A1:A0 FC         103 ERR      LDY   #$FC          ;SUBSTITUTE $FC FOR INVALID OPS
F8A3:A9 00         104          LDA   #$00          ;SET PRINT FORMAT INDEX TO 0
F8A5:AA            105 GETFMT   TAX
F8A6:BD A6 F9      106          LDA   FMT2,X        ;INDEX INTO PRINT FORMAT TABLE
F8A9:85 2E         107          STA   FORMAT        ;SAVE FOR ADR FIELD FORMATTING
F8AB:29 03         108          AND   #$03          ;MASK FOR 2-BIT LENGTH
F8AD:              109 ; (0=1 BYTE, 1=2 BYTE, 2=3 BYTE)
F8AD:85 2F         110          STA   LENGTH
F8AF:20 35 FC      111          JSR   NEWOPS       ;get index for new opcodes
F8B2:F0 18   F8CC  112          BEQ   GOTONE       ;found a new op (or no op)
F8B4:29 8F         113          AND   #$8F          ;MASK FOR 1XXX1010 TEST
F8B6:AA            114          TAX                 ; SAVE IT
F8B7:98            115          TYA                 ;OPCODE TO A AGAIN
F8B8:A0 03         116          LDY   #$03
F8BA:E0 8A         117          CPX   #$8A
F8BC:F0 0B   F8C9  118          BEQ   MNNDX3
```



```
F8BE:4A            119 MNNDX1   LSR   A
F8BF:90 08   F8C9  120          BCC   MNNDX3        ;FORM INDEX INTO MNEMONIC TABLE
F8C1:4A            121          LSR   A
F8C2:4A            122 MNNDX2   LSR   A             ;  1) 1XXX1010 => 00101XXX
F8C3:09 20         123          ORA   #$20          ;  2) XXXYYY01 => 00111XXX
F8C5:88            124          DEY                 ;  3) XXXYYY10 => 00110XXX
F8C6:D0 FA   F8C2  125          BNE   MNNDX2        ;  4) XXXYY100 => 00100XXX
F8C8:C8            126          INY                 ;  5) XXXXX000 => 000XXXXX
F8C9:88            127 MNNDX3   DEY
F8CA:D0 F2   F8BE  128          BNE   MNNDX1
F8CC:60            129 GOTONE   RTS
F8CD:              130 *
F8CD:FF FF FF      131          DFB   $FF,$FF,$FF
F8D0:              132 *
F8D0:20 82 F8      133 INSTDSP  JSR   INSDS1        ;GEN FMT, LEN BYTES
F8D3:48            134          PHA                 ;SAVE MNEMONIC TABLE INDEX
F8D4:B1 3A         135 PRNTOP   LDA   (PCL),Y
F8D6:20 DA FD      136          JSR   PRBYTE
F8D9:A2 01         137          LDX   #$01          ;PRINT 2 BLANKS
F8DB:20 4A F9      138 PRNTBL   JSR   PRBL2
F8DE:C4 2F         139          CPY   LENGTH        ;PRINT INST (1-3 BYTES)
F8E0:C8            140          INY                 ;IN A 12 CHR FIELD
F8E1:90 F1   F8D4  141          BCC   PRNTOP
F8E3:A2 03         142          LDX   #$03          ;CHAR COUNT FOR MNEMONIC INDEX
F8E5:C0 04         143          CPY   #$04
F8E7:90 F2   F8DB  144          BCC   PRNTBL
F8E9:68            145          PLA                 ;RECOVER MNEMONIC INDEX
F8EA:A8            146          TAY
F8EB:B9 C0 F9      147          LDA   MNEML,Y
F8EE:85 2C         148          STA   LMNEM         ;FETCH 3-CHAR MNEMONIC
F8F0:B9 00 FA      149          LDA   MNEMR,Y       ;  (PACKED INTO 2-BYTES)
F8F3:85 2D         150          STA   RMNEM
F8F5:A9 00         151 PRMN1    LDA   #$00
F8F7:A0 05         152          LDY   #$05
F8F9:06 2D         153 PRMN2    ASL   RMNEM         ;SHIFT 5 BITS OF CHARACTER INTO A
F8FB:26 2C         154          ROL   LMNEM
F8FD:2A            155          ROL   A             ;  (CLEARS CARRY)
F8FE:88            156          DEY
F8FF:D0 F8   F8F9  157          BNE   PRMN2
F901:69 BF         158          ADC   #$BF          ;ADD "?" OFFSET
F903:20 ED FD      159          JSR   COUT          ;OUTPUT A CHAR OF MNEM
F906:CA            160          DEX
F907:D0 EC   F8F5  161          BNE   PRMN1
F909:20 48 F9      162          JSR   PRBLNK        ;OUTPUT 3 BLANKS
F90C:A4 2F         163          LDY   LENGTH
F90E:A2 06         164          LDX   #$06          ;CNT FOR 6 FORMAT BITS
F910:E0 03         165 PRADR1   CPX   #$03
F912:F0 1C   F930  166          BEQ   PRADR5        ;IF X=3 THEN ADDR.
F914:06 2E         167 PRADR2   ASL   FORMAT
F916:90 0E   F926  168          BCC   PRADR3
F918:BD B9 F9      169          LDA   CHAR1-1,X
F91B:20 ED FD      170          JSR   COUT
F91E:BD B3 F9      171          LDA   CHAR2-1,X
F921:F0 03   F926  172          BEQ   PRADR3
F923:20 ED FD      173          JSR   COUT
F926:CA            174 PRADR3   DEX
F927:D0 E7   F910  175          BNE   PRADR1
F929:60            176          RTS
```

```
F92A:                177 *
F92A:88              178 PRADR4   DEY
F92B:30 E7   F914    179          BMI   PRADR2
F92D:20 DA FD        180          JSR   PRBYTE
F930:A5 2E           181 PRADR5   LDA   FORMAT
F932:C9 E8           182          CMP   #$E8          ;HANDLE REL ADR MODE
F934:B1 3A           183          LDA   (PCL),Y       ;SPECIAL (PRINT TARGET,
F936:90 F2   F92A    184          BCC   PRADR4        ;  NOT OFFSET)
F938:20 56 F9        185 RELADR   JSR   PCADJ3
F93B:AA              186          TAX                 ;PCL,PCH+OFFSET+1 TO A,Y
F93C:E8              187          INX
F93D:D0 01   F940    188          BNE   PRNTYX        ;+1 TO Y,X
F93F:C8              189          INY
F940:98              190 PRNTYX   TYA
F941:20 DA FD        191 PRNTAX   JSR   PRBYTE        ;OUTPUT TARGET ADR
F944:8A              192 PRNTX    TXA                 ;  OF BRANCH AND RETURN
F945:4C DA FD        193          JMP   PRBYTE
F948:                194 *
F948:A2 03           195 PRBLNK   LDX   #$03          ;BLANK COUNT
F94A:A9 A0           196 PRBL2    LDA   #$A0          ;LOAD A SPACE
F94C:20 ED FD        197 PRBL3    JSR   COUT          ;OUTPUT A BLANK
F94F:CA              198          DEX
F950:D0 F8   F94A    199          BNE   PRBL2         ;LOOP UNTIL COUNT=0
F952:60              200          RTS
F953:                201 *
F953:38              202 PCADJ    SEC                 ;0=1 BYTE, 1=2 BYTE,
F954:A5 2F           203 PCADJ2   LDA   LENGTH        ;  2=3 BYTE
F956:A4 3B           204 PCADJ3   LDY   PCH
F958:AA              205          TAX                 ;TEST DISPLACEMENT SIGN
F959:10 01   F95C    206          BPL   PCADJ4        ;  (FOR REL BRANCH)
F95B:88              207          DEY                 ;EXTEND NEG BY DECR PCH
F95C:65 3A           208 PCADJ4   ADC   PCL
F95E:90 01   F961    209          BCC   RTS2          ;PCL+LENGTH(OR DISPL)+1 TO A
F960:C8              210          INY                 ;  CARRY INTO Y (PCH)
F961:60              211 RTS2     RTS
F962:                212 *
F962:                213 ; FMT1 BYTES:   XXXXXXY0 INSTRS
F962:                214 ; IF Y=0        THEN RIGHT HALF BYTE
F962:                215 ; IF Y=1        THEN LEFT HALF BYTE
F962:                216 ;                 (X=INDEX)
F962:                217 *
F962:0F              218 FMT1     DFB   $0F
F963:22              219          DFB   $22
F964:FF              220          DFB   $FF
F965:33              221          DFB   $33
F966:CB              222          DFB   $CB
F967:62              223          DFB   $62
F968:FF              224          DFB   $FF
F969:73              225          DFB   $73
F96A:03              226          DFB   $03
F96B:22              227          DFB   $22
F96C:FF              228          DFB   $FF
F96D:33              229          DFB   $33
F96E:CB              230          DFB   $CB
F96F:66              231          DFB   $66
F970:FF              232          DFB   $FF
F971:77              233          DFB   $77
F972:0F              234          DFB   $0F
```



```
F973:20              235          DFB   $20
F974:FF              236          DFB   $FF
F975:33              237          DFB   $33
F976:CB              238          DFB   $CB
F977:60              239          DFB   $60
F978:FF              240          DFB   $FF
F979:70              241          DFB   $70
F97A:0F              242          DFB   $0F
F97B:22              243          DFB   $22
F97C:FF              244          DFB   $FF
F97D:39              245          DFB   $39
F97E:CB              246          DFB   $CB
F97F:66              247          DFB   $66
F980:FF              248          DFB   $FF
F981:7D              249          DFB   $7D
F982:0B              250          DFB   $0B
F983:22              251          DFB   $22
F984:FF              252          DFB   $FF
F985:33              253          DFB   $33
F986:CB              254          DFB   $CB
F987:A6              255          DFB   $A6
F988:FF              256          DFB   $FF
F989:73              257          DFB   $73
F98A:11              258          DFB   $11
F98B:22              259          DFB   $22
F98C:FF              260          DFB   $FF
F98D:33              261          DFB   $33
F98E:CB              262          DFB   $CB
F98F:A6              263          DFB   $A6
F990:FF              264          DFB   $FF
F991:87              265          DFB   $87
F992:01              266          DFB   $01
F993:22              267          DFB   $22
F994:FF              268          DFB   $FF
F995:33              269          DFB   $33
F996:CB              270          DFB   $CB
F997:60              271          DFB   $60
F998:FF              272          DFB   $FF
F999:70              273          DFB   $70
F99A:01              274          DFB   $01
F99B:22              275          DFB   $22
F99C:FF              276          DFB   $FF
F99D:33              277          DFB   $33
F99E:CB              278          DFB   $CB
F99F:60              279          DFB   $60
F9A0:FF              280          DFB   $FF
F9A1:70              281          DFB   $70
F9A2:24              282          DFB   $24
F9A3:31              283          DFB   $31
F9A4:65              284          DFB   $65
F9A5:78              285          DFB   $78
F9A6:                286 ; ZZXXXY01 INSTR'S
F9A6:00              287 FMT2     DFB   $00           ;ERR
F9A7:21              288          DFB   $21           ;IMM
F9A8:81              289          DFB   $81           ;Z-PAGE
F9A9:82              290          DFB   $82           ;ABS
F9AA:59              291          DFB   $59           ;(ZPAG,X)
F9AB:4D              292          DFB   $4D           ;(ZPAG),Y
```

```
F9AC:91   293          DFB  $91        ;ZPAG,X
F9AD:92   294          DFB  $92         ;ABS,X
F9AE:86   295          DFB  $86         ;ABS,Y
F9AF:4A   296          DFB  $4A         ;(ABS)
F9B0:85   297          DFB  $85         ;ZPAG,Y
F9B1:9D   298          DFB  $9D         ;RELATIVE
F9B2:49   299          DFB  $49        ;(ZPAG)     (new)
F9B3:5A   300          DFB  $5A        ;(ABS,X)    (new)
F9B4:     301 *
F9B4:D9   302 CHAR2    DFB  $D9         ;'Y'
F9B5:00   303          DFB  $00        ; (byte F of FMT2)
F9B6:D8   304          DFB  $D8         ;'Y'
F9B7:A4   305          DFB  $A4         ;'$'
F9B8:A4   306          DFB  $A4         ;'$'
F9B9:00   307          DFB  $00
F9BA:     308 *
F9BA:AC   309 CHAR1    DFB  $AC         ;','
F9BB:A9   310          DFB  $A9         ;')'
F9BC:AC   311          DFB  $AC         ;','
F9BD:A3   312          DFB  $A3         ;'#'
F9BE:A8   313          DFB  $A8         ;'('
F9BF:A4   314          DFB  $A4         ;'$'
F9C0:1C   315 MNEML    DFB  $1C
F9C1:8A   316          DFB  $8A
F9C2:1C   317          DFB  $1C
F9C3:23   318          DFB  $23
F9C4:5D   319          DFB  $5D
F9C5:8B   320          DFB  $8B
F9C6:1B   321          DFB  $1B
F9C7:A1   322          DFB  $A1
F9C8:9D   323          DFB  $9D
F9C9:8A   324          DFB  $8A
F9CA:1D   325          DFB  $1D
F9CB:23   326          DFB  $23
F9CC:9D   327          DFB  $9D
F9CD:8B   328          DFB  $8B
F9CE:1D   329          DFB  $1D
F9CF:A1   330          DFB  $A1
F9D0:1C   331          DFB  $1C        ;BRA
F9D1:29   332          DFB  $29
F9D2:19   333          DFB  $19
F9D3:AE   334          DFB  $AE
F9D4:69   335          DFB  $69
F9D5:A8   336          DFB  $A8
F9D6:19   337          DFB  $19
F9D7:23   338          DFB  $23
F9D8:24   339          DFB  $24
F9D9:53   340          DFB  $53
F9DA:1B   341          DFB  $1B
F9DB:23   342          DFB  $23
F9DC:24   343          DFB  $24
F9DD:53   344          DFB  $53
F9DE:19   345          DFB  $19
F9DF:A1   346          DFB  $A1        ; (A) FORMAT ABOVE
F9E0:AD   347          DFB  $AD        ; TSB
F9E1:1A   348          DFB  $1A
F9E2:5B   349          DFB  $5B
F9E3:5B   350          DFB  $5B
```



```
F9E4:A5   351          DFB  $A5
F9E5:69   352          DFB  $69
F9E6:24   353          DFB  $24
F9E7:24   354          DFB  $24        ; (B) FORMAT
F9E8:AE   355          DFB  $AE
F9E9:AE   356          DFB  $AE
F9EA:A8   357          DFB  $A8
F9EB:AD   358          DFB  $AD
F9EC:29   359          DFB  $29
F9ED:8A   360          DFB  $8A
F9EE:7C   361          DFB  $7C
F9EF:8B   362          DFB  $8B        ; (C) FORMAT
F9F0:15   363          DFB  $15
F9F1:9C   364          DFB  $9C
F9F2:6D   365          DFB  $6D
F9F3:9C   366          DFB  $9C
F9F4:A5   367          DFB  $A5
F9F5:69   368          DFB  $69
F9F6:29   369          DFB  $29
F9F7:53   370          DFB  $53        ; (D) FORMAT
F9F8:84   371          DFB  $84
F9F9:13   372          DFB  $13
F9FA:34   373          DFB  $34
F9FB:11   374          DFB  $11
F9FC:A5   375          DFB  $A5
F9FD:69   376          DFB  $69
F9FE:23   377          DFB  $23         ; (E) FORMAT
F9FF:A0   378          DFB  $A0
FA00:     379 *
FA00:D8   380 MNEMR    DFB  $D8
FA01:62   381          DFB  $62
FA02:5A   382          DFB  $5A
FA03:48   383          DFB  $48
FA04:26   384          DFB  $26
FA05:62   385          DFB  $62
FA06:94   386          DFB  $94
FA07:88   387          DFB  $88
FA08:54   388          DFB  $54
FA09:44   389          DFB  $44
FA0A:C8   390          DFB  $C8
FA0B:54   391          DFB  $54
FA0C:68   392          DFB  $68
FA0D:44   393          DFB  $44
FA0E:E8   394          DFB  $E8
FA0F:94   395          DFB  $94
FA10:C4   396          DFB  $C4        ;BRA
FA11:B4   397          DFB  $B4
FA12:08   398          DFB  $08
FA13:84   399          DFB  $84
FA14:74   400          DFB  $74
FA15:B4   401          DFB  $B4
FA16:28   402          DFB  $28
FA17:6E   403          DFB  $6E
FA18:74   404          DFB  $74
FA19:F4   405          DFB  $F4
FA1A:CC   406          DFB  $CC
FA1B:4A   407          DFB  $4A
FA1C:72   408          DFB  $72
```

```
FA1D:F2          409           DFB   $F2
FA1E:A4          410           DFB   $A4
FA1F:8A          411           DFB   $8A          ; (A) FORMAT
FA20:06          412           DFB   $06          ; TSB
FA21:AA          413           DFB   $AA
FA22:A2          414           DFB   $A2
FA23:A2          415           DFB   $A2
FA24:74          416           DFB   $74
FA25:74          417           DFB   $74
FA26:74          418           DFB   $74
FA27:72          419           DFB   $72          ; (B) FORMAT
FA28:44          420           DFB   $44
FA29:68          421           DFB   $68
FA2A:B2          422           DFB   $B2
FA2B:32          423           DFB   $32
FA2C:B2          424           DFB   $B2
FA2D:72          425           DFB   $72
FA2E:22          426           DFB   $22
FA2F:72          427           DFB   $72          ; (C) FORMAT
FA30:1A          428           DFB   $1A
FA31:1A          429           DFB   $1A
FA32:26          430           DFB   $26
FA33:26          431           DFB   $26
FA34:72          432           DFB   $72
FA35:72          433           DFB   $72
FA36:88          434           DFB   $88
FA37:C8          435           DFB   $C8          ; (D) FORMAT
FA38:C4          436           DFB   $C4
FA39:CA          437           DFB   $CA
FA3A:26          438           DFB   $26
FA3B:48          439           DFB   $48
FA3C:44          440           DFB   $44
FA3D:44          441           DFB   $44
FA3E:A2          442           DFB   $A2
FA3F:C8          443           DFB   $C8          ; (E) FORMAT
FA40:            444 *
FA40:85 45       445 IRQ       STA   $45          ;+ Trash $45 for those who want it
FA42:A5 45       446           LDA   $45          ;+
FA44:4C 03 C8    447           JMP   NEWIRQ       ;+
FA47:            448 *
FA47:            449 *
FA47:            450 * NEWBRK is called by the interrupt handler which has
FA47:            451 * set the hardware to its default state and encoded
FA47:            452 * the state in the accumulator.  Software that wants
FA47:            453 * to do break processing using full system resources
FA47:            454 * can restore the machine state from this value.
FA47:            455 *
FA47:85 44       456 NEWBRK     STA   MACSTAT      ;save state of machine
FA49:7A          457           PLY                ;restore registers for save
FA4A:FA          458           PLX
FA4B:68          459           PLA
FA4C:            460 *
FA4C:28          461 BREAK     PLP                ;Note: same as old BREAK routine!!
FA4D:20 4A FF    462           JSR   SAVE         ;save reg's on BRK
FA50:68          463           PLA                ;including PC
FA51:85 3A       464           STA   PCL
FA53:68          465           PLA
FA54:85 3B       466           STA   PCH
```



```
FA56:6C F0 03    467           JMP   (BRKV)       ;call BRK HANDLER
FA59:            468 *
FA59:20 82 F8    469 OLDBRK    JSR   INSDS1       ;PRINT USER PC
FA5C:20 DA FA    470           JSR   RGDSP1       ;  AND REGS
FA5F:4C 65 FF    471           JMP   MON          ;GO TO MONITOR (NO PASS GO, NO $200!)
FA62:            472 *
FA62:D8          473 RESET     CLD                ;DO THIS FIRST THIS TIME
FA63:20 84 FE    474           JSR   SETNORM
FA66:20 2F FB    475           JSR   INIT
FA69:20 4D CE    476           JSR   ZZQUIT       ;+ Setvid & Setkbd
FA6C:20 40 C7    477           JSR   INITMOUSE    ;initialize the mouse
FA6F:20 04 CC    478           JSR   CLRPORT      ;clear port setup bytes
FA72:9C FC 04    479           STZ   ACIABUF      ;and the commahead buffer
FA75:AD 5F C0    480           LDA   SETAN3       ; AN3 = TTL HI
FA78:20 BD FA    481           JSR   RESET.X      ; initialize other devices
FA7B:2C 10 C0    482           BIT   KBDSTRB      ; CLEAR KEYBOARD
FA7E:80 05  FA85 483           BRA   BEEPSKIP     ;+ Bell already beeped
FA80:EA          484           NOP                ;+ align code
FA81:D8          485 NEWMON    CLD
FA82:20 3A FF    486           JSR   BELL         ; CAUSES DELAY IF KEY BOUNCES
FA85:AD F3 03    487 BEEPSKIP  LDA   SOFTEV+1     ;IS RESET HI
FA88:49 A5       488           EOR   #$A5         ;A FUNNY COMPLEMENT OF THE
FA8A:CD F4 03    489           CMP   PWREDUP      ; PWR UP BYTE ???
FA8D:D0 17  FAA6 490           BNE   PWRUP        ; NO SO PWRUP
FA8F:AD F2 03    491           LDA   SOFTEV       ; YES SEE IF COLD START
FA92:D0 0F  FAA3 492           BNE   NOFIX        ; HAS BEEN DONE YET?
FA94:A9 E0       493           LDA   #$E0         ; DOES SEV POINT AT BASIC?
FA96:CD F3 03    494           CMP   SOFTEV+1
FA99:D0 08  FAA3 495           BNE   NOFIX        ; YES SO REENTER SYSTEM
FA9B:A0 03       496 FIXSEV    LDY   #3           ; NO SO POINT AT WARM START
FA9D:8C F2 03    497           STY   SOFTEV       ; FOR NEXT RESET
FAA0:4C 00 E0    498           JMP   BASIC        ; AND DO THE COLD START
FAA3:            499 *
FAA3:6C F2 03    500 NOFIX     JMP   (SOFTEV)
FAA6:            501 *
FAA6:20 CA FC    502 PWRUP     JSR   COLDSTART    ;Trash memory, init ports
FAA9:       FAA9 503 SETPG3    EQU   *            ; SET PAGE 3 VECTORS
FAA9:A2 05       504           LDX   #5
FAAB:BD FC FA    505 SETPLP    LDA   PWRCON-1,X   ; WITH CNTRL B ADRS
FAAE:9D EF 03    506           STA   BRKV-1,X     ; OF CURRENT BASIC
FAB1:CA          507           DEX
FAB2:D0 F7  FAAB 508           BNE   SETPLP
FAB4:A9 C4       509           LDA   #$C4         ; LOAD HI SLOT +1
FAB6:80 5A  FB12 510           BRA   PWRUP2       ;branch around mnemonics
FAB8:            511 *
FAB8:            512 * Extension to MNEML (left mnemonics)
FAB8:            513 *
FAB8:8A          514           DFB   $8A          ;PHY
FAB9:8B          515           DFB   $8B          ;PLY
FABA:A5          516           DFB   $A5          ;STZ
FABB:AC          517           DFB   $AC          ;TRB
FABC:00          518           DFB   $00          ;???
FABD:            519 *
FABD:            520 * This extension to the monitor reset routine ($FA62)
FABD:            521 * checks for apple keys.  If both are pressed, it goes
FABD:            522 * into an exerciser mode.  If the open apple key only is
FABD:            523 * pressed, memory is selectively trashed and a cold start
FABD:            524 * is done.
```

```
                        525 *
FABD:                   526 RESET.X  LDA  #$FF
FABF:8D FB 04           527          STA  VMODE        ;initialize mode
FAC2:20 3A FF           528          JSR  BELL         ;+ Need bell delay for 3.5" drive
FAC5:20 F8 C5           529          JSR  PCNVRST      ;+ Reset protocol converter
FAC8:0E 62 C0           530          ASL  BUTN1
FACB:2C 61 C0           531          BIT  BUTN0
FACE:10 5E   FB2E       532          BPL  RTS2D
FAD0:90 D4   FAA6       533          BCC  PWRUP        ;open apple only, reboot
FAD2:4C C1 C7           534          JMP  BANGER       ;both apples, exercise 'er
FAD5:EA                 535          NOP               ;+ align code
FAD6:EA                 536          NOP               ;+
FAD7:20 8E FD           537 REGDSP   JSR  CROUT        ;DISPLAY USER REG CONTENTS
FADA:A9 44              538 RGDSP1   LDA  #$44         ;WITH LABELS
FADC:85 40              539          STA  A3L          ;Memory state now printed
FADE:A9 00              540          LDA  #$00
FAE0:85 41              541          STA  A3H
FAE2:A2 FA              542          LDX  #$FA
FAE4:A9 A0              543 RDSP1    LDA  #$A0
FAE6:20 ED FD           544          JSR  COUT
FAE9:BD 9A CE           545          LDA  RTBL-$FA,X
FAEC:20 ED FD           546          JSR  COUT
FAEF:A9 BD              547          LDA  #$BD
FAF1:20 ED FD           548          JSR  COUT
FAF4:B5 4A              549          LDA  ACC+5,X
FAF6:80 0A   FB02       550          BRA  RGDSP2       ;make room for mnemonics
FAF8:                   551 *
FAF8:                   552 * Right half of new mnemonics, indexed from MNEMR
FAF8:                   553 *
FAF8:74                 554          DFB  $74          ;PHY
FAF9:74                 555          DFB  $74          ;PLY
FAFA:76                 556          DFB  $76          ;STZ
FAFB:C6                 557          DFB  $C6          ;TRB
FAFC:00                 558          DFB  $00          ;???
FAFD:                   559 *
FAFD:59 FA              560 PWRCON   DW   OLDBRK
FAFF:00 E0 45           561          DFB  $00,$E0,$45
FB02:                   562 *
FB02:20 DA FD           563 RGDSP2   JSR  PRBYTE
FB05:E8                 564          INX
FB06:30 DC   FAE4       565          BMI  RDSP1
FB08:60                 566          RTS
FB09:                   567 *
FB09:C1 F0 F0 EC        568 TITLE    ASC  'Apple       ]['
FB11:C4                 569          DFB  $C4          ;optional filler
FB12:                   570 *
FB12:86 00              571 PWRUP2   STX  LOC0          ; SETPG3 MUST RETURN X=0
FB14:85 01              572          STA  LOC1          ; SET PTR H
FB16:20 60 FB           573          JSR  APPLEII      ;Display our banner...
FB19:6C 00 00           574          JMP  (LOC0)       ;JUMP $C600
FB1C:00                 575          BRK
FB1D:00                 576          BRK
FB1E:                   577 *
FB1E:4C 00 C9           578 PREAD    JMP  MPADDLE      ;read mouse paddle
FB21:A0 00              579          LDY  #$00         ;INIT COUNT
FB23:EA                 580          NOP               ;COMPENSATE FOR 1ST COUNT
FB24:EA                 581          NOP
FB25:BD 64 C0           582 PREAD2   LDA  PADDL0,X     ;COUNT Y-REG EVERY 12 USEC.
```



```
FB28:10 04   FB2E       583          BPL  RTS2D
FB2A:C8                 584          INY
FB2B:D0 F8   FB25       585          BNE  PREAD2       ;EXIT AT 255 MAX
FB2D:88                 586          DEY
FB2E:60                 587 RTS2D    RTS
FB2F:                    61          INCLUDE AUTOST2
```

```
FB2F:                    2 *
FB2F:A9 00               3 INIT    LDA   #$00        ;CLR STATUS FOR DEBUG SOFTWARE
FB31:85 48               4         STA   STATUS
FB33:AD 56 C0            5         LDA   LORES
FB36:AD 54 C0            6         LDA   TXTPAGE1    ;INIT VIDEO MODE
FB39:AD 51 C0            7 SETTXT  LDA   TXTSET      ;SET FOR TEXT MODE
FB3C:A9 00               8         LDA   #$00        ;FULL SCREEN WINDOW
FB3E:F0 0B   FB4B        9         BEQ   SETWND
FB40:AD 50 C0           10 SETGR   LDA   TXTCLR      ;SET FOR GRAPHICS MODE
FB43:AD 53 C0           11         LDA   MIXSET      ;LOWER 4 LINES AS TEXT WINDOW
FB46:20 36 F8           12         JSR   CLRTOP
FB49:A9 14              13         LDA   #$14
FB4B:85 22              14 SETWND  STA   WNDTOP      ;SET WINDOW
FB4D:EA                 15         NOP
FB4E:EA                 16         NOP
FB4F:20 0A CE           17         JSR   WNDREST     ;40/80 column width
FB52:80 05   FB59       18         BRA   VTAB23
FB54:                   19 *
FB54:09 80              20 DOCTL   ORA   #$80        ;controls need high bit
FB56:4C 54 CD           21         JMP   CTLCHAR0    ;execute control char
FB59:                   22 *
FB59:A9 17              23 VTAB23  LDA   #$17        ;VTAB TO ROW 23
FB5B:85 25              24 TABV    STA   CV          ;VTABS TO ROW IN A-REG
FB5D:4C 22 FC           25         JMP   VTAB        ;don't set OURCV!!
FB60:                   26 *
FB60:20 58 FC           27 APPLEII JSR   HOME        ;CLEAR THE SCRN
FB63:A0 09              28         LDY   #9
FB65:B9 BA C5           29 STITLE  LDA   APPLE2C-1,Y ;GET A CHAR
FB68:99 0D 04           30         STA   LINE1+13,Y  ;PUT IT AT TOP CENTER OF SCREEN
FB6B:88                 31         DEY
FB6C:D0 F7   FB65       32         BNE   STITLE
FB6E:60                 33         RTS
FB6F:                   34 *
FB6F:AD F3 03           35 SETPWRC LDA   SOFTEV+1    ;ROUTINE TO CALCULATE THE 'FUNNY
FB72:49 A5              36         EOR   #$A5        ;COMPLEMENT' FOR THE RESET VECTOR
FB74:8D F4 03           37         STA   PWREDUP
FB77:60                 38         RTS
FB78:                   39 *
FB78:          FB78     40 VIDWAIT EQU   *           ;CHECK FOR A PAUSE (CONTROL-S).
FB78:C9 8D              41         CMP   #$8D        ;ONLY WHEN I HAVE A CR
FB7A:D0 18   FB94       42         BNE   NOWAIT      ;NOT SO, DO REGULAR
FB7C:AC 00 C0           43         LDY   KBD         ;IS KEY PRESSED?
FB7F:10 13   FB94       44         BPL   NOWAIT      ;NO.
FB81:C0 93              45         CPY   #$93        ;YES -- IS IT CTRL-S?
FB83:D0 0F   FB94       46         BNE   NOWAIT      ;NOPE - IGNORE
FB85:2C 10 C0           47         BIT   KBDSTRB     ;CLEAR STROBE
FB88:AC 00 C0           48 KBDWAIT LDY   KBD         ;WAIT TILL NEXT KEY TO RESUME
FB8B:10 FB   FB88       49         BPL   KBDWAIT     ;WAIT FOR KEYPRESS
FB8D:C0 83              50         CPY   #$83        ;IS IT CONTROL-C?
FB8F:F0 03   FB94       51         BEQ   NOWAIT      ;YES, SO LEAVE IT
FB91:2C 10 C0           52         BIT   KBDSTRB     ;CLR STROBE
FB94:2C 7B 06           53 NOWAIT  BIT   VFACTV      ;is video firmware active?
FB97:30 64   FBFD       54         BMI   VIDOUT      ;=>no, do normal 40 column
FB99:89 60              55         BIT   #$60        ;is it a control?
FB9B:F0 B7   FB54       56         BEQ   DOCTL       ;=>yes, do it
FB9D:20 B8 C3           57         JSR   STORCH      ;print w/inverse mask
FBA0:EE 7B 05           58 NEWADV  INC   OURCH      ;advance cursor
FBA3:AD 7B 05           59         LDA   OURCH       ;and update others
```



```
FBA6:2C 1F C0           60         BIT   RD80VID     ;but only if not 80 columns
FBA9:30 05   FBB0       61         BMI   NEWADV1     ;=>80 columns, leav'em
FBAB:8D 7B 04           62         STA   OLDCH
FBAE:85 24              63         STA   CH
FBB0:80 46   FBF8       64 NEWADV1 BRA   ADV2        ;check for CR
FBB2:                   65 *
FBB2:EA                 66         NOP
FBB3:                   67 *
FBB3:06                 68 F8VERSION DFB  GOODF8      ;//e, chels ID byte
FBB4:                   69 *
FBB4:10 06   FBBC       70 DOCOUT1 BPL   DCX         ;=>video firmware active, no mask
FBB6:C9 A0              71         CMP   #$A0        ;is it control char?
FBB8:90 02   FBBC       72         BCC   DCX         ;=>yes, no mask
FBBA:25 32              73         AND   INVFLG      ;else apply inverse mask
FBBC:4C F6 FD           74 DCX     JMP   COUTZ       ;and print character
FBBF:03                 75         dfb   $03         ;revision byte
FBC0:                   76 *
FBC0:00                 77         DFB   $00         ;chels ID byte
FBC1:                   78 *
FBC1:48                 79 BASCALC PHA               ;CALC BASE ADDR IN BASL,H
FBC2:4A                 80         LSR   A           ;FOR GIVEN LINE NO.
FBC3:29 03              81         AND   #$03        ; 0<=LINE NO.<=$17
FBC5:09 04              82         ORA   #$04        ;ARG=000ABCDE, GENERATE
FBC7:85 29              83         STA   BASH        ; BASH=000001CD
FBC9:68                 84         PLA               ; AND
FBCA:29 18              85         AND   #$18        ; BASL=EABAB000
FBCC:90 02   FBD0       86         BCC   BASCLC2
FBCE:69 7F              87         ADC   #$7F
FBD0:85 28              88 BASCLC2 STA   BASL
FBD2:0A                 89         ASL   A
FBD3:0A                 90         ASL   A
FBD4:05 28              91         ORA   BASL
FBD6:85 28              92         STA   BASL
FBD8:60                 93         RTS
FBD9:                   94 *
FBD9:C9 87              95 CHKBELL CMP   #$87        ;BELL CHAR? (CONTROL-G)
FBDB:D0 12   FBEF       96         BNE   RTS2B       ; NO, RETURN.
FBDD:A9 40              97 BELL1   LDA   #$40        ; YES...
FBDF:20 A8 FC           98         JSR   WAIT        ;DELAY .01 SECONDS
FBE2:A0 C0              99         LDY   #$C0
FBE4:A9 0C             100 BELL2   LDA   #$0C        ;TOGGLE SPEAKER AT 1 KHZ
FBE6:20 A8 FC          101         JSR   WAIT        ; FOR .1 SEC.
FBE9:AD 30 C0          102         LDA   SPKR
FBEC:88                103         DEY
FBED:D0 F5   FBE4      104         BNE   BELL2
FBEF:60                105 RTS2B   RTS
FBF0:                  106 *
FBF0:A4 24             107 STORADV LDY   CH          ;get 40 column position
FBF2:91 28             108         STA   (BASL),Y    ;and store
FBF4:E6 24             109 ADVANCE INC   CH          ;increment cursor
FBF6:A5 24             110         LDA   CH
FBF8:C5 21             111 ADV2    CMP   WNDWDTH     ;BEYOND WINDOW WIDTH?
FBFA:B0 66   FC62      112         BCS   CR          ; YES, CR TO NEXT LINE.
FBFC:60                113 RTS3    RTS               ; NO, RETURN.
FBFD:                  114 *
FBFD:C9 A0             115 VIDOUT  CMP   #$A0        ;CONTROL CHAR?
FBFF:B0 EF   FBF0      116         BCS   STORADV     ; NO, OUTPUT IT.
FC01:A8                117         TAY               ;INVERSE VIDEO?
```

```
FC02:10 EC   FBF0  118          BPL   STORADV       ; YES, OUTPUT IT.
FC04:C9 8D        119 VIDOUT1   CMP   #$8D         ;CR?
FC06:F0 6B   FC73 120          BEQ   NEWCR         ;Yes, use new routine
FC08:C9 8A        121          CMP   #$8A         ;LINE FEED?
FC0A:F0 5A   FC66 122          BEQ   LF            ; IF SO, DO IT.
FC0C:C9 88        123          CMP   #$88         ;BACK SPACE? (CONTROL-H)
FC0E:D0 C9   FBD9 124          BNE   CHKBELL       ; NO, CHECK FOR BELL.
FC10:20 E2 FE     125 BS       JSR   DECCH         ;decrement all cursor H indices
FC13:10 E7   FBFC 126          BPL   RTS3          ;IF POSITIVE, OK; ELSE MOVE UP.
FC15:A5 21        127          LDA   WNDWDTH      ;get window width,
FC17:20 EB FE     128          JSR   WDTHCH       ;and set CH's to WNDWDTH-1
FC1A:A5 22        129 UP       LDA   WNDTOP        ;CURSOR V INDEX
FC1C:C5 25        130          CMP   CV
FC1E:B0 DC   FBFC 131          BCS   RTS3         ;top line, exit
FC20:C6 25        132          DEC   CV           ;not top, go up one
FC22:             133 *
FC22:80 62   FC86 134 VTAB     BRA   NEWVTAB      ;go update OURCV
FC24:20 C1 FB     135 VTABZ    JSR   BASCALC      ;calculate the base address
FC27:A5 20        136          LDA   WNDLFT       ;get the left window edge
FC29:2C 1F C0     137          BIT   RD80VID      ;80 columns?
FC2C:10 02   FC30 138          BPL   VTAB40       ;=>no, left edge ok
FC2E:4A           139          LSR   A            ;divide width by 2
FC2F:18           140          CLC                ;prepare to add
FC30:65 28        141 VTAB40   ADC   BASL         ;add width to base
FC32:85 28        142          STA   BASL
FC34:60           143 RTS4     RTS
FC35:             144 *
FC35:             145 * NEWOPS translates the opcode in the Y register
FC35:             146 * to a mnemonic table index and returns with Z=1.
FC35:             147 * If Y is not a new opcode, Z=0.
FC35:             148 *
FC35:98           149 NEWOPS    TYA                ;get the opcode
FC36:A2 16        150          LDX   #NUMOPS      ;check through new opcodes
FC38:DD FE FE     151 NEWOP1   CMP   OPTBL,X      ;does it match?
FC3B:F0 43   FC80 152          BEQ   GETINDX      ;=>yes, get new index
FC3D:CA           153          DEX
FC3E:10 F8   FC38 154          BPL   NEWOP1       ;else check next one
FC40:60           155          RTS                ;not found, exit with BNE
FC41:             156 *
FC41:00           157          BRK
FC42:             158 *
FC42:80 19   FC5D 159 CLREOP   BRA   CLREOP1       ;ESC F IS CLR TO END OF PAGE
FC44:A5 25        160 CLREOP2  LDA   CV
FC46:48           161 CLEOP1   PHA                  ;SAVE CURRENT LINE NO. ON STACK
FC47:20 24 FC     162          JSR   VTABZ          ;CALC BASE ADDRESS
FC4A:20 9E FC     163          JSR   CLEOLZ       ;CLEAR TO EOL. (SETS CARRY)
FC4D:A0 00        164          LDY   #$00          ;CLEAR FROM H INDEX=0 FOR REST
FC4F:68           165          PLA                  ;INCREMENT CURRENT LINE NO.
FC50:1A           166          INC   A
FC51:C5 23        167          CMP   WNDBTM       ;DONE TO BOTTOM OF WINDOW?
FC53:90 F1   FC46 168          BCC   CLEOP1         ; NO, KEEP CLEARING LINES.
FC55:B0 CB   FC22 169          BCS   VTAB           ; YES, TAB TO CURRENT LINE
FC57:00           170          BRK
FC58:             171 *
FC58:20 A5 CD     172 HOME     JSR   HOMECUR      ;move cursor home
FC5B:80 E7   FC44 173          BRA   CLREOP2      ;then clear to end of page
FC5D:             174 *
FC5D:20 9D CC     175 CLREOP1  JSR   GETCUR        ;load Y with proper CH
```



```
FC60:80 E2   FC44 176          BRA   CLREOP2      ;before clearing page
FC62:             177 *
FC62:80 0F   FC73 178 CR       BRA   NEWCR        ;only LF if not Pascal
FC64:00           179          BRK
FC65:00           180          BRK
FC66:             181 *
FC66:E6 25        182 LF       INC   CV            ;INCR CURSOR V. (DOWN 1 LINE)
FC68:A5 25        183          LDA   CV
FC6A:C5 23        184          CMP   WNDBTM        ;OFF SCREEN?
FC6C:90 1A   FC88 185          BCC   NEWVTABZ     ;set base+WNDLFT
FC6E:C6 25        186          DEC   CV            ;DECR CURSOR V. (BACK TO BOTTOM)
FC70:             187 *
FC70:4C 35 CB     188 SCROLL   JMP   SCROLLUP     ;scroll the screen
FC73:             189 *
FC73:20 E9 FE     190 NEWCR    JSR   CLRCH        ;set CH's to 0
FC76:2C FB 04     191          BIT   VMODE        ;is it Pascal?
FC79:10 0A   FC85 192          BPL   CRRTS        ;pascal, no LF
FC7B:20 44 FD     193          JSR   NOESCAPE     ;else clear escape mode
FC7E:80 E6   FC66 194          BRA   LF           ;then do LF
FC80:             195 *
FC80:BD 15 FF     196 GETINDX  LDA   INDX,X       ;lookup index for mnemonic
FC83:A0 00        197          LDY   #0           ;exit with BEQ
FC85:60           198 CRRTS    RTS
FC86:             199 *
FC86:A5 25        200 NEWVTAB  LDA   CV           ;update //e CV
FC88:8D FB 05     201 NEWVTABZ STA   OURCV
FC8B:80 97   FC24 202          BRA   VTABZ        ;and calc base+WNDLFT
FC8D:             203 *
FC8D:20 9D CC     204 NEWCLREOL JSR  GETCUR       ;get current cursor
FC90:A9 A0        205 NEWCLEOLZ LDA  #$A0         ;get a blank
FC92:2C 7B 06     206          BIT   VFACTV       ;if video firmware active,
FC95:30 02   FC99 207          BMI   NEWC1        ;=>don't use inverse mask
FC97:25 32        208          AND   INVFLG
FC99:4C C2 CB     209 NEWC1    JMP   DOCLR        ;go do clear
FC9C:             210 *
FC9C:80 EF   FC8D 211 CLREOL   BRA   NEWCLREOL    ;get cursor and clear
FC9E:80 F0   FC90 212 CLEOLZ   BRA   NEWCLEOLZ    ;clear from Y
FCA0:             213 *
FCA0:A0 00        214 CLRLIN   LDY   #0           ;clear entire line
FCA2:80 EC   FC90 215          BRA   NEWCLEOLZ
FCA4:             216 *
FCA4:7C 2A CD     217 CTLDO    JMP   (CTLADR,X)   ;jump to proper routine
FCA7:             218 *
FCA7:EA           219          NOP
FCA8:             220 *
FCA8:38           221 WAIT     SEC
FCA9:48           222 WAIT2    PHA
FCAA:E9 01        223 WAIT3    SBC   #$01
FCAC:D0 FC   FCAA 224          BNE   WAIT3         ;1.0204 USEC
FCAE:68           225          PLA                  ;(13+2712*A+512*A*A)
FCAF:E9 01        226          SBC   #$01
FCB1:D0 F6   FCA9 227          BNE   WAIT2
FCB3:60           228 RTS6     RTS
FCB4:             229 *
FCB4:E6 42        230 NXTA4    INC   A4L          ;INCR 2-BYTE A4
FCB6:D0 02   FCBA 231          BNE   NXTA1        ; AND A1
FCB8:E6 43        232          INC   A4H
FCBA:A5 3C        233 NXTA1    LDA   A1L          ;INCR 2-BYTE A1.
```

```
22 AUTOST2          Apple //c F8 monitor firmware      20-OCT-86  06:41 PAGE 83

FCBC:C5 3E          234          CMP   A2L          ; AND COMPARE TO A2
FCBE:A5 3D          235          LDA   A1H          ; (CARRY SET IF >=)
FCC0:E5 3F          236          SBC   A2H
FCC2:E6 3C          237          INC   A1L
FCC4:D0 02   FCC8   238          BNE   RTS4B
FCC6:E6 3D          239          INC   A1H
FCC8:60             240 RTS4B    RTS
FCC9:               241 *
FCC9:60             242 HEADR    RTS                ;don't do it
FCCA:               243 *
FCCA:A0 B0          244 COLDSTART LDY  #$B0         ;let it precess down
FCCC:64 3C          245          STZ   A1L
FCCE:A2 BF          246          LDX   #$BF         ;start from BFXX down
FCD0:86 3D          247 BLAST    STX   A1H
FCD2:A9 A0          248          LDA   #$A0         ;store blanks
FCD4:91 3C          249          STA   (A1L),Y
FCD6:88             250          DEY
FCD7:91 3C          251          STA   (A1L),Y
FCD9:CA             252          DEX                ;back down to next page
FCDA:E0 01          253          CPX   #1           ;stay away from stack
FCDC:D0 F2   FCD0   254          BNE   BLAST        ;fall into COMINIT
FCDE:               255 *
FCDE:8D 01 C0       256          STA   SET80COL     ;init ALT screen holes
FCE1:AD 55 C0       257          LDA   TXTPAGE2     ;for serial and comm ports
FCE4:A2 88          258          LDX   #$88         ;C = 1 from CPX #1
FCE6:BD 8B CF       259 COM1     LDA   COMTBL-1,X   ;XFER from rom
FCE9:90 0A   FCF5   260          BCC   COM2         ;branch if defaults ok
FCEB:DD 77 04       261          CMP   $477,X       ;test for prior setup
FCEE:18             262          CLC                ;branch if not valid
FCEF:D0 04   FCF5   263          BNE   COM2         ;If $4F8 & $4FF = TBL values
FCF1:E0 82          264          CPX   #$82
FCF3:90 06   FCFB   265          BCC   COM3
FCF5:9D 77 04       266 COM2     STA   $477,X
FCF8:CA             267          DEX                ;move all 8...
FCF9:D0 EB   FCE6   268          BNE   COM1
FCFB:AD 54 C0       269 COM3     LDA   TXTPAGE1     ;restore switches
FCFE:8D 00 C0       270          STA   CLR80COL     ;to default states
FD01:60             271          RTS
FD02:EA             272          NOP                ;+
FD03:EA             273          NOP
FD04:EA             274          NOP
FD05:EA             275          NOP
FD06:EA             276          NOP
FD07:EA             277          NOP
FD08:EA             278          NOP
FD09:EA             279          NOP
FD0A:EA             280          NOP
FD0B:EA             281          NOP
FD0C:               282 *
FD0C:A4 24          283 RDKEY    LDY   CH           ;get char at current position
FD0E:B1 28          284          LDA   (BASL),Y     ;for those who restore it
FD10:EA             285          NOP                ;if a program controls input
FD11:EA             286          NOP                ;hooks, no cursor may be displayed
FD12:EA             287          NOP
FD13:EA             288          NOP
FD14:EA             289          NOP
FD15:EA             290          NOP
FD16:EA             291          NOP
```

```
22 AUTOST2          Apple //c F8 monitor firmware      20-OCT-86  06:41 PAGE 84

FD17:EA             292          NOP
FD18:               293 *
FD18:6C 38 00       294 KEYIN0   JMP   (KSWL)       ;GO TO USER KEY-IN
FD1B:               295 *
FD1B:91 28          296 KEYIN    STA   (BASL),Y     ;erase false images
FD1D:20 4C CC       297          JSR   SHOWCUR      ;display true cursor
FD20:20 70 CC       298 DONXTCUR JSR   UPDATE       ;look for key, blink II cursor
FD23:10 FB   FD20   299          BPL   DONXTCUR     ;loop until keypress
FD25:48             300 GOTKEY   PHA                ;save character
FD26:A9 08          301          LDA   #M.CTL       ;were escapes enabled?
FD28:2C FB 04       302          BIT   VMODE
FD2B:D0 1D   FD4A   303          BNE   NOESC2       ;=>no, there is no escape
FD2D:68             304          PLA                ;yes, there may be a way out!!
FD2E:C9 9B          305          CMP   #ESC         ;escape?
FD30:D0 06   FD38   306          BNE   LOOKPICK     ;=>no escape
FD32:4C CC CC       307          JMP   NEWESC       ;=>go do escape sequence
FD35:               308 *
FD35:4C ED CC       309 RDCHAR   JMP   ESCRDKEY     ;do RDKEY with escapes
FD38:               310 *
FD38:2C 7B 06       311 LOOKPICK BIT   VFACTV       ;only process f.arrow
FD3B:30 07   FD44   312          BMI   NOESCAPE     ;if video firmware is active
FD3D:C9 95          313          CMP   #PICK        ;was it PICK? (->,CTL-U)
FD3F:D0 03   FD44   314          BNE   NOESCAPE     ;no, just return
FD41:20 1D CC       315          JSR   PICKY        ;yes, pick the character
FD44:               316 *
FD44:               317 * NOESCAPE is used by GETCOUT too.
FD44:               318 *
FD44:48             319 NOESCAPE PHA                ;save it
FD45:A9 08          320 NOESC1   LDA   #M.CTL       ;disable escape sequences
FD47:0C FB 04       321          TSB   VMODE        ;and enable controls
FD4A:68             322 NOESC2   PLA                ;by setting M.CTL
FD4B:60             323          RTS
FD4C:               324 *
FD4C:EA             325          NOP
FD4D:               326 *
FD4D:20 A6 C3       327 NOTCR    JSR   GETCOUT      ;disable controls and print
FD50:C9 88          328          CMP   #$88         ;CHECK FOR EDIT KEYS
FD52:F0 1D   FD71   329          BEQ   BCKSPC       ; - BACKSPACE
FD54:C9 98          330          CMP   #$98
FD56:F0 0A   FD62   331          BEQ   CANCEL       ; - CONTROL-X
FD58:E0 F8          332          CPX   #$F8
FD5A:90 03   FD5F   333          BCC   NOTCR1       ;MARGIN?
FD5C:20 3A FF       334          JSR   BELL         ; YES, SOUND BELL
FD5F:E8             335 NOTCR1   INX                ;ADVANCE INPUT INDEX
FD60:D0 13   FD75   336          BNE   NXTCHAR
FD62:A9 DC          337 CANCEL   LDA   #$DC         ;BACKSLASH AFTER CANCELLED LINE
FD64:20 A6 C3       338          JSR   GETCOUT
FD67:20 8E FD       339 GETLNZ   JSR   CROUT        ;OUTPUT 'CR'
FD6A:A5 33          340 GETLN    LDA   PROMPT       ;OUTPUT PROMPT CHAR
FD6C:20 ED FD       341          JSR   COUT
FD6F:A2 01          342 GETLN1   LDX   #$01         ;INIT INPUT INDEX
FD71:8A             343 BCKSPC   TXA
FD72:F0 F3   FD67   344          BEQ   GETLNZ       ;WILL BACKSPACE TO 0
FD74:CA             345          DEX
FD75:20 ED CC       346 NXTCHAR  JSR   ESCRDKEY     ;do new RDCHAR (allow escapes)
FD78:C9 95          347          CMP   #PICK        ;USE SCREEN CHAR
FD7A:D0 08   FD84   348          BNE   ADDINP       ; FOR CONTROL-U
FD7C:20 1D CC       349          JSR   PICKY        ;lift char from screen
```

```
FD7F:EA          350          NOP
FD80:EA          351          NOP
FD81:EA          352          NOP             ;no upshifting needed
FD82:EA          353          NOP
FD83:EA          354          NOP
FD84:9D 00 02    355 ADDINP   STA   IN,X      ;ADD TO INPUT BUFFER
FD87:C9 8D       356          CMP   #$8D
FD89:D0 C2  FD4D 357          BNE   NOTCR
FD8B:20 9C FC    358 CROUT1   JSR   CLREOL    ;CLR TO EOL IF CR
FD8E:A9 8D       359 CROUT    LDA   #$8D
FD90:D0 5B  FDED 360          BNE   COUT      ;(ALWAYS)
FD92:            361 *
FD92:A4 3D       362 PRA1     LDY   A1H        ;PRINT CR,A1 IN HEX
FD94:A6 3C       363          LDX   A1L
FD96:20 8E FD    364 PRYX2    JSR   CROUT
FD99:20 40 F9    365          JSR   PRNTYX
FD9C:A0 00       366          LDY   #$00
FD9E:A9 AD       367          LDA   #$AD       ;PRINT '-'
FDA0:4C ED FD    368          JMP   COUT
FDA3:            369 *
FDA3:A5 3C       370 XAM8     LDA   A1L
FDA5:09 07       371          ORA   #$07       ;SET TO FINISH AT
FDA7:85 3E       372          STA   A2L        ; MOD 8=7
FDA9:A5 3D       373          LDA   A1H
FDAB:85 3F       374          STA   A2H
FDAD:A5 3C       375 MOD8CHK  LDA   A1L
FDAF:29 07       376          AND   #$07
FDB1:D0 03  FDB6 377          BNE   DATAOUT
FDB3:20 92 FD    378 XAM      JSR   PRA1
FDB6:A9 A0       379 DATAOUT  LDA   #$A0
FDB8:20 ED FD    380          JSR   COUT       ;OUTPUT BLANK
FDBB:B1 3C       381          LDA   (A1L),Y
FDBD:20 DA FD    382          JSR   PRBYTE     ;OUTPUT BYTE IN HEX
FDC0:20 BA FC    383          JSR   NXTA1
FDC3:90 E8  FDAD 384          BCC   MOD8CHK    ;NOT DONE YET. GO CHECK MOD 8
FDC5:60          385 RTS4C    RTS              ;DONE.
FDC6:            386 *
FDC6:4A          387 XAMPM    LSR   A          ;DETERMINE IF MONITOR MODE IS
FDC7:90 EA  FDB3 388          BCC   XAM        ; EXAMINE, ADD OR SUBTRACT
FDC9:4A          389          LSR   A
FDCA:4A          390          LSR   A
FDCB:A5 3E       391          LDA   A2L
FDCD:90 02  FDD1 392          BCC   ADD
FDCF:49 FF       393          EOR   #$FF       ;FORM 2'S COMPLEMENT FOR SUBTRACT.
FDD1:65 3C       394 ADD      ADC   A1L
FDD3:48          395          PHA
FDD4:A9 BD       396          LDA   #$BD       ;PRINT '=', THEN RESULT
FDD6:20 ED FD    397          JSR   COUT
FDD9:68          398          PLA
FDDA:            399 *
FDDA:48          400 PRBYTE   PHA              ;PRINT BYTE AS 2 HEX DIGITS
FDDB:4A          401          LSR   A          ; (DESTROYS A-REG)
FDDC:4A          402          LSR   A
FDDD:4A          403          LSR   A
FDDE:4A          404          LSR   A
FDDF:20 E5 FD    405          JSR   PRHEXZ
FDE2:68          406          PLA
FDE3:            407 *
```



```
FDE3:29 0F       408 PRHEX    AND   #$0F       ;PRINT HEX DIGIT IN A-REG
FDE5:09 B0       409 PRHEXZ   ORA   #$B0       ;LSBITS ONLY.
FDE7:C9 BA       410          CMP   #$BA
FDE9:90 02  FDED 411          BCC   COUT
FDEB:69 06       412          ADC   #$06
FDED:            413 *
FDED:6C 36 00    414 COUT     JMP   (CSWL)     ;VECTOR TO USER OUTPUT ROUTINE
FDF0:            415 *
FDF0:2C 7B 06    416 COUT1    BIT   VFACTV     ;video firmware active?
FDF3:4C B4 FB    417          JMP   DOCOUT1    ;mask II mode characters
FDF6:84 35       418 COUTZ    STY   YSAV1      ;SAVE Y-REG
FDF8:48          419          PHA              ;SAVE A -REG
FDF9:20 78 FB    420          JSR   VIDWAIT    ;OUTPUT CHR AND CHECK FOR CTRL-S
FDFC:68          421          PLA              ;RESTORE A-REG
FDFD:A4 35       422          LDY   YSAV1      ;AND Y-REG
FDFF:60          423          RTS              ;RETURN TO SENDER...
FE00:            424 *
FE00:C6 34       425 BL1      DEC   YSAV
FE02:F0 9F  FDA3 426          BEQ   XAM8
FE04:            427 *
FE04:CA          428 BLANK    DEX              ;BLANK TO MON
FE05:D0 16  FE1D 429          BNE   SETMDZ     ;AFTER BLANK
FE07:C9 BA       430          CMP   #$BA       ;DATA STORE MODE?
FE09:D0 BB  FDC6 431          BNE   XAMPM      ; NO; XAM, ADD, OR SUBTRACT.
FE0B:            432 *
FE0B:85 31       433 STOR     STA   MODE       ;KEEP IN STORE MODE
FE0D:A5 3E       434          LDA   A2L
FE0F:91 40       435          STA   (A3L),Y    ;STORE AS LOW BYTE AT (A3)
FE11:E6 40       436          INC   A3L
FE13:D0 02  FE17 437          BNE   RTS5       ;INCR A3, RETURN.
FE15:E6 41       438          INC   A3H
FE17:60          439 RTS5     RTS
FE18:            440 *
FE18:A4 34       441 SETMODE  LDY   YSAV        ;SAVE CONVERTED ':', '+',
FE1A:B9 FF 01    442          LDA   IN-1,Y     ; '-', '.' AS MODE
FE1D:85 31       443 SETMDZ   STA   MODE
FE1F:60          444          RTS
FE20:            445 *
FE20:A2 01       446 LT       LDX   #$01
FE22:B5 3E       447 LT2      LDA   A2L,X      ;COPY A2 (2 BYTES) TO
FE24:95 42       448          STA   A4L,X      ; A4 AND A5
FE26:95 44       449          STA   A5L,X
FE28:CA          450          DEX
FE29:10 F7  FE22 451          BPL   LT2
FE2B:60          452          RTS
FE2C:            453 *
FE2C:B1 3C       454 MOVE     LDA   (A1L),Y    ;MOVE (A1) THRU (A2) TO (A4)
FE2E:91 42       455          STA   (A4L),Y
FE30:20 B4 FC    456          JSR   NXTA4
FE33:90 F7  FE2C 457          BCC   MOVE
FE35:60          458          RTS
FE36:            459 *
FE36:B1 3C       460 VERIFY   LDA   (A1L),Y    ;VERIFY (A1) THRU (A2)
FE38:D1 42       461          CMP   (A4L),Y    ; WITH (A4)
FE3A:F0 1C  FE58 462          BEQ   VFYOK
FE3C:20 92 FD    463          JSR   PRA1
FE3F:B1 3C       464          LDA   (A1L),Y
FE41:20 DA FD    465          JSR   PRBYTE
```

```
22 AUTOST2        Apple //c F8 monitor firmware      20-OCT-86  06:41 PAGE 87

FE44:A9 A0          466           LDA   #$A0
FE46:20 ED FD       467           JSR   COUT
FE49:A9 A8          468           LDA   #$A8
FE4B:20 ED FD       469           JSR   COUT
FE4E:B1 42          470           LDA   (A4L),Y
FE50:20 DA FD       471           JSR   PRBYTE
FE53:A9 A9          472           LDA   #$A9
FE55:20 ED FD       473           JSR   COUT
FE58:20 B4 FC       474 VFYOK     JSR   NXTA4
FE5B:90 D9   FE36   475           BCC   VERIFY
FE5D:60             476           RTS
FE5E:               477 *
FE5E:20 75 FE       478 LIST      JSR   A1PC          ;MOVE A1 (2 BYTES) TO
FE61:A9 14          479           LDA   #$14          ; PC IF SPEC'D AND
FE63:48             480 LIST2     PHA                 ; DISASSEMBLE 20 INSTRUCTIONS.
FE64:20 C4 C5       481           JSR   SHOWINST      ;Display a line
FE67:68             482           PLA
FE68:3A             483           DEC   A             ;Count down
FE69:D0 F8   FE63   484           BNE   LIST2
FE6B:60             485           RTS
FE6C:               486 *
FE6C:4C 86 C9       487 MINI      JMP   GETINST1      ;Go to the mini assembler
FE6F:C6 34          488 TRACE     DEC   YSAV          ;Stay on T for trace
FE71:4C 43 CA       489 STEPZ     JMP   STEP          ;Off to the step routine
FE74:       0001    490           ds    $FE75-*,0     ;Extra bytes
FE75:               491 *
FE75:8A             492 A1PC      TXA                 ;IF USER SPECIFIED AN ADDRESS,
FE76:F0 07   FE7F   493           BEQ   A1PCRTS       ; COPY IT FROM A1 TO PC.
FE78:B5 3C          494 A1PCLP    LDA   A1L,X         ;YEP, SO COPY IT.
FE7A:95 3A          495           STA   PCL,X
FE7C:CA             496           DEX
FE7D:10 F9   FE78   497           BPL   A1PCLP
FE7F:60             498 A1PCRTS   RTS
FE80:               499 *
FE80:A0 3F          500 SETINV    LDY   #$3F          ;SET FOR INVERSE VID
FE82:D0 02   FE86   501           BNE   SETIFLG       ; VIA COUT1
FE84:A0 FF          502 SETNORM   LDY   #$FF          ;SET FOR NORMAL VID
FE86:84 32          503 SETIFLG   STY   INVFLG
FE88:60             504           RTS
FE89:               505 *
FE89:A9 00          506 SETKBD    LDA   #$00          ;DO 'IN#0'
FE8B:85 3E          507 INPORT    STA   A2L           ;DO 'IN#AREG'
FE8D:A2 38          508 INPRT     LDX   #KSWL
FE8F:A0 1B          509           LDY   #KEYIN
FE91:D0 08   FE9B   510           BNE   IOPRT
FE93:               511 *
FE93:A9 00          512 SETVID    LDA   #$0           ;DO 'PR#0'
FE95:85 3E          513 OUTPORT   STA   A2L           ;DO 'PR#AREG'
FE97:A2 36          514 OUTPRT    LDX   #CSWL
FE99:A0 F0          515           LDY   #COUT1
FE9B:A5 3E          516 IOPRT     LDA   A2L
FE9D:29 0F          517           AND   #$0F
FE9F:D0 06   FEA7   518           BNE   NOTPRT0       ;not slot 0
FEA1:C0 1B          519           CPY   #KEYIN        ;Continue if KEYIN
FEA3:F0 39   FEDE   520           BEQ   IOPRT1
FEA5:80 1B   FEC2   521           BRA   OPRT0         ;=>do PR#0
FEA7:09 C0          522 NOTPRT0   ORA   #<IOADR
FEA9:A0 00          523           LDY   #$00

22 AUTOST2        Apple //c F8 monitor firmware      20-OCT-86  06:41 PAGE 88

FEAB:94 00          524 IOPRT2    STY   LOC0,X
FEAD:95 01          525           STA   LOC1,X
FEAF:60             526           RTS
FEB0:               527 *
FEB0:4C 00 E0       528 XBASIC    JMP   BASIC         ;TO BASIC, COLD START
FEB3:               529 *
FEB3:4C 03 E0       530 BASCONT   JMP   BASIC2        ;TO BASIC, WARM START
FEB6:               531 *
FEB6:20 75 FE       532 GO        JSR   A1PC          ;ADDR TO PC IF SPECIFIED
FEB9:20 3F FF       533           JSR   RESTORE       ;RESTORE FAKE REGISTERS
FEBC:6C 3A 00       534           JMP   (PCL)         ; AND GO!
FEBF:               535 *
FEBF:4C D7 FA       536 REGZ      JMP   REGDSP        ;GO DISPLAY REGISTERS
FEC2:               537 *
FEC2:3A             538 OPRT0     DEC   A             ;Need $FF
FEC3:8D FB 07       539           STA   CURSOR        ;set checkerboard cursor
FEC6:A9 F7          540           LDA   #$FF-M.CTL    ;reset mode
FEC8:80 04   FECE   541           BRA   DOPR0
FECA:               542 *
FECA:4C F8 03       543 USR       JMP   USRADR        ;JUMP TO CONTROL-Y VECTOR IN RAM
FECD:               544 *
FECD:60             545 WRITE     RTS                 ;Tape write not needed
FECE:               546 *
FECE:8D 7B 06       547 DOPR0     STA   VFACTV        ;say video firmware inactive
FED1:8D 0E C0       548           STA   CLRALTCHAR    ;switch in normal char set
FED4:0C FB 04       549           TSB   VMODE         ;don't change M.CTL
FED7:DA             550           PHX                 ;save X and Y
FED8:5A             551           PHY                 ;for rest of PR#0
FED9:20 CD CD       552           JSR   CHK80         ;convert to 40 if needed
FEDC:7A             553           PLY
FEDD:FA             554           PLX
FEDE:A9 FD          555 IOPRT1    LDA   #<COUT1       ;set I/O page
FEE0:80 C9   FEAB   556           BRA   IOPRT2        ;=>go set output hook
FEE2:               557 *
FEE2:               558 * DECCH decrements the current cursor
FEE2:               559 * CLRCH sets all cursors to 0
FEE2:               560 * SETCUR sets cursors to value in Acc.
FEE2:               561 * See explanatory note with GETCUR
FEE2:               562 *
FEE2:5A             563 DECCH     PHY                 ;(from $FC10)
FEE3:20 9D CC       564           JSR   GETCUR        ;get current CH
FEE6:88             565           DEY                 ;decrement it
FEE7:80 05   FEEE   566           BRA   SETCUR1       ;go update cursors
FEE9:               567 *
FEE9:A9 01          568 CLRCH     LDA   #1            ;set all cursors to 0
FEEB:3A             569 WDTHCH    DEC   A             ;dec window width (from $FC17)
FEEC:5A             570 SETCUR    PHY                 ;save Y
FEED:A8             571           TAY                 ;need value in Y
FEEE:20 AD CC       572 SETCUR1   JSR   GETCUR2       ;save new CH
FEF1:7A             573           PLY                 ;restore Y
FEF2:AD 7B 05       574           LDA   OURCH         ;and get new CH into acc
FEF5:60             575           RTS                 ;(Need LDA to set flags)
FEF6:               576 *
FEF6:20 00 FE       577 CRMON     JSR   BL1           ;HANDLE CR AS BLANK
FEF9:68             578           PLA                 ; THEN POP STACK
FEFA:68             579           PLA                 ; AND RETURN TO MON
FEFB:D0 6C   FF69   580           BNE   MONZ          ;(ALWAYS)
FEFD:               581 *
```

```
FEFD:60          582 READ     RTS              ;Tape read not needed
FEFE:            583 *
FEFE:            584 * OPTBL is a table containing the new opcodes that
FEFE:            585 * wouldn't fit into the existing lookup table.
FEFE:            586 *
FEFE:12          587 OPTBL    DFB  $12         ;ORA (ZPAG)
FEFF:14          588          DFB  $14         ;TRB ZPAG
FF00:1A          589          DFB  $1A         ;INC A
FF01:1C          590          DFB  $1C         ;TRB ABS
FF02:32          591          DFB  $32         ;AND (ZPAG)
FF03:34          592          DFB  $34         ;BIT ZPAG,X
FF04:3A          593          DFB  $3A         ;DEC A
FF05:3C          594          DFB  $3C         ;BIT ABS,X
FF06:52          595          DFB  $52         ;EOR (ZPAG)
FF07:5A          596          DFB  $5A         ;PHY
FF08:64          597          DFB  $64         ;STZ ZPAG
FF09:72          598          DFB  $72         ;ADC (ZPAG)
FF0A:74          599          DFB  $74         ;STZ ZPAG,X
FF0B:7A          600          DFB  $7A         ;PLY
FF0C:7C          601          DFB  $7C         ;JMP (ABS,X)
FF0D:89          602          DFB  $89         ;BIT IMM
FF0E:92          603          DFB  $92         ;STA (ZPAG)
FF0F:9C          604          DFB  $9C         ;STZ ABS
FF10:9E          605          DFB  $9E         ;STZ ABS,X
FF11:B2          606          DFB  $B2         ;LDA (ZPAG)
FF12:D2          607          DFB  $D2         ;CMP (ZPAG)
FF13:F2          608          DFB  $F2         ;SBC (ZPAG)
FF14:FC          609          DFB  $FC         ;??? (the unknown opcode)
FF15:      0016  610 NUMOPS   EQU  *-OPTBL-1   ;number of bytes to check
FF15:            611 *
FF15:            612 * INDX contains pointers to the mnemonics for each of
FF15:            613 * the opcodes in OPTBL.  Pointers with BIT 7
FF15:            614 * set indicate extensions to MNEML or MNEMR.
FF15:            615 *
FF15:38          616 INDX     DFB  $38
FF16:FB          617          DFB  $FB
FF17:37          618          DFB  $37
FF18:FB          619          DFB  $FB
FF19:39          620          DFB  $39
FF1A:21          621          DFB  $21
FF1B:36          622          DFB  $36
FF1C:21          623          DFB  $21
FF1D:3A          624          DFB  $3A
FF1E:F8          625          DFB  $F8
FF1F:FA          626          DFB  $FA
FF20:3B          627          DFB  $3B
FF21:FA          628          DFB  $FA
FF22:F9          629          DFB  $F9
FF23:22          630          DFB  $22
FF24:21          631          DFB  $21
FF25:3C          632          DFB  $3C
FF26:FA          633          DFB  $FA
FF27:FA          634          DFB  $FA
FF28:3D          635          DFB  $3D
FF29:3E          636          DFB  $3E
FF2A:3F          637          DFB  $3F
FF2B:FC          638          DFB  $FC         ;???
FF2C:00          639          BRK
```



```
FF2D:            640 *
FF2D:A9 C5       641 PRERR    LDA  #$C5           ;PRINT 'ERR', THEN FALL INTO
FF2F:20 ED FD    642          JSR  COUT           ; FWEEPER.
FF32:A9 D2       643          LDA  #$D2
FF34:20 ED FD    644          JSR  COUT
FF37:20 ED FD    645          JSR  COUT
FF3A:            646 *
FF3A:A9 87       647 BELL     LDA  #$87           ;MAKE A JOYFUL NOISE, THEN RETURN.
FF3C:4C ED FD    648          JMP  COUT
FF3F:            649 *
FF3F:A5 48       650 RESTORE  LDA  STATUS         ;RESTORE 6502 REGISTER CONTENTS
FF41:48          651          PHA                 ; USED BY DEBUG SOFTWARE
FF42:A5 45       652          LDA  A5H
FF44:A6 46       653 RESTR1   LDX  XREG
FF46:A4 47       654          LDY  YREG
FF48:28          655          PLP
FF49:60          656          RTS
FF4A:            657 *
FF4A:85 45       658 SAVE     STA  A5H            ;SAVE 6502 REGISTER CONTENTS
FF4C:86 46       659 SAV1     STX  XREG           ; FOR DEBUG SOFTWARE
FF4E:84 47       660          STY  YREG
FF50:08          661          PHP
FF51:68          662          PLA
FF52:85 48       663          STA  STATUS
FF54:BA          664          TSX
FF55:86 49       665          STX  SPNT
FF57:D8          666          CLD
FF58:60          667          RTS
FF59:            668 *
FF59:20 84 FE    669 OLDRST   JSR  SETNORM        ;SET SCREEN MODE
FF5C:20 2F FB    670          JSR  INIT           ; AND INIT KBD/SCREEN
FF5F:20 93 FE    671          JSR  SETVID         ; AS I/O DEVS.
FF62:20 89 FE    672          JSR  SETKBD
FF65:            673 *
FF65:D8          674 MON      CLD                 ;MUST SET HEX MODE!
FF66:20 3A FF    675          JSR  BELL           ;FWEEPER.
FF69:A9 AA       676 MONZ     LDA  #$AA           ;'*' PROMPT FOR MONITOR
FF6B:85 33       677          STA  PROMPT
FF6D:20 67 FD    678          JSR  GETLNZ         ;READ A LINE OF INPUT
FF70:20 C7 FF    679          JSR  ZMODE          ;CLEAR MONITOR MODE, SCAN IDX
FF73:20 A7 FF    680 NXTITM   JSR  GETNUM         ;GET ITEM, NON-HEX
FF76:84 34       681          STY  YSAV           ; CHAR IN A-REG.
FF78:A0 17       682          LDY  #SUBTBL-CHRTBL ; X-REG=0 IF NO HEX INPUT
FF7A:88          683 CHRSRCH  DEY
FF7B:30 E8   FF65 684          BMI  MON            ;COMMAND NOT FOUND, BEEP & TRY AGAIN.
FF7D:D9 CC FF    685          CMP  CHRTBL,Y       ;FIND COMMAND CHAR IN TABLE
FF80:D0 F8   FF7A 686          BNE  CHRSRCH        ;NOT THIS TIME
FF82:20 BE FF    687          JSR  TOSUB          ;GOT IT! CALL CORRESPONDING SUBROUTINE
FF85:A4 34       688          LDY  YSAV           ;PROCESS NEXT ENTRY ON HIS LINE
FF87:4C 73 FF    689          JMP  NXTITM
FF8A:            690 *
FF8A:A2 03       691 DIG      LDX  #$03
FF8C:0A          692          ASL  A
FF8D:0A          693          ASL  A              ;GOT HEX DIGIT,
FF8E:0A          694          ASL  A              ; SHIFT INTO A2
FF8F:0A          695          ASL  A
FF90:0A          696 NXTBIT   ASL  A
FF91:26 3E       697          ROL  A2L
```

```
22 AUTOST2        Apple //c F8 monitor firmware      20-OCT-86  06:41 PAGE 91

FF93:26 3F          698          ROL   A2H
FF95:CA             699          DEX                ;LEAVE X=$FF IF DIG
FF96:10 F8   FF90   700          BPL   NXTBIT
FF98:A5 31          701 NXTBAS   LDA   MODE
FF9A:D0 06   FFA2   702          BNE   NXTBS2       ;IF MODE IS ZERO,
FF9C:B5 3F          703          LDA   A2H,X        ; THEN COPY A2 TO A1 AND A3
FF9E:95 3D          704          STA   A1H,X
FFA0:95 41          705          STA   A3H,X
FFA2:E8             706 NXTBS2   INX
FFA3:F0 F3   FF98   707          BEQ   NXTBAS
FFA5:D0 06   FFAD   708          BNE   NXTCHR
FFA7:A2 00          709 GETNUM   LDX   #$00         ;CLEAR A2
FFA9:86 3E          710          STX   A2L
FFAB:86 3F          711          STX   A2H
FFAD:20 B4 C5       712 NXTCHR   JSR   GETUP        ;Get char, iny, upshift
FFB0:49 B0          713          EOR   #$B0
FFB2:C9 0A          714          CMP   #$0A
FFB4:90 D4   FF8A   715          BCC   DIG          ;it's a digit
FFB6:69 88          716          ADC   #$88
FFB8:C9 FA          717          CMP   #$FA
FFBA:4C CB CF       718          JMP   LOOKASC      ;+ Check for quote
FFBD:00             719          BRK
FFBE:               720 *
FFBE:A9 FE          721 TOSUB    LDA   #<GO         ;DISPATCH TO SUBROUTINE, BY
FFC0:48             722          PHA                ; PUSHING THE HI-ORDER SUBR ADDR,
FFC1:B9 E3 FF       723          LDA   SUBTBL,Y     ; THEN THE LO-ORDER SUBR ADDR
FFC4:48             724          PHA                ; ONTO THE STACK,
FFC5:A5 31          725          LDA   MODE         ; (CLEARING THE MODE, SAVE THE OLD
FFC7:A0 00          726 ZMODE    LDY   #$00         ; MODE IN A-REG),
FFC9:84 31          727          STY   MODE
FFCB:60             728          RTS                ; AND 'RTS' TO THE SUBROUTINE!
FFCC:               729 *
FFCC:BC             730 CHRTBL   DFB   $BC          ;^C  (BASIC WARM START)
FFCD:B2             731          DFB   $B2          ;^Y  (USER VECTOR)
FFCE:BE             732          DFB   $BE          ;^E  (OPEN AND DISPLAY REGISTERS)
FFCF:9A             733          DFB   $9A          ;+!  (Mini assembler)
FFD0:EF             734          DFB   $EF          ;V   (MEMORY VERIFY)
FFD1:C4             735          DFB   $C4          ;^K  (IN#SLOT)
FFD2:A9             736          DFB   $A9          ;^P  (PR#SLOT)
FFD3:BB             737          DFB   $BB          ;^B  (BASIC COLD START)
FFD4:A6             738          DFB   $A6          ;'-' (SUBTRACTION)
FFD5:A4             739          DFB   $A4          ;'+' (ADDITION)
FFD6:06             740          DFB   $06          ;M   (MEMORY MOVE)
FFD7:95             741          DFB   $95          ;'<' (DELIMITER FOR MOVE, VFY)
FFD8:07             742          DFB   $07          ;N   (SET NORMAL VIDEO)
FFD9:02             743          DFB   $02          ;I   (SET INVERSE VIDEO)
FFDA:05             744          DFB   $05          ;L   (DISASSEMBLE 20 INSTRS)
FFDB:00             745          DFB   $00          ;G   (EXECUTE PROGRAM)
FFDC:93             746          DFB   $93          ;':' (MEMORY FILL)
FFDD:A7             747          DFB   $A7          ;'.' (ADDRESS DELIMITER)
FFDE:C6             748          DFB   $C6          ;'CR' (END OF INPUT)
FFDF:99             749          DFB   $99          ;BLANK
FFE0:EC             750          DFB   $EC          ;+S  (Step)
FFE1:ED             751          DFB   $ED          ;+T  (Trace)
FFE2:EA             752          NOP                ;+
FFE3:               753 *
FFE3:               754 * Table of low order monitor routine
FFE3:               755 * dispatch addresses.
```

```
22 AUTOST2        Apple //c F8 monitor firmware      20-OCT-86  06:41 PAGE 92

FFE3:               756 *
FFE3:B2             757 SUBTBL   DFB   >BASCONT-1
FFE4:C9             758          DFB   >USR-1
FFE5:BE             759          DFB   >REGZ-1
FFE6:6B             760          DFB   >MINI-1      ;+
FFE7:35             761          DFB   >VERIFY-1
FFE8:8C             762          DFB   >INPRT-1
FFE9:96             763          DFB   >OUTPRT-1
FFEA:AF             764          DFB   >XBASIC-1
FFEB:17             765          DFB   >SETMODE-1
FFEC:17             766          DFB   >SETMODE-1
FFED:2B             767          DFB   >MOVE-1
FFEE:1F             768          DFB   >LT-1
FFEF:83             769          DFB   >SETNORM-1
FFF0:7F             770          DFB   >SETINV-1
FFF1:5D             771          DFB   >LIST-1
FFF2:B5             772          DFB   >GO-1
FFF3:17             773          DFB   >SETMODE-1
FFF4:17             774          DFB   >SETMODE-1
FFF5:F5             775          DFB   >CRMON-1
FFF6:03             776          DFB   >BLANK-1
FFF7:70             777          DFB   >STEPZ-1      ;+
FFF8:6E             778          DFB   >TRACE-1      ;+
FFF9:               779 *
FFF9:       0001    780          ds    $FFFA-*,0
FFFA:               781 *
FFFA:FB 03          782          DW    NMI           ;NON-MASKABLE INTERRUPT VECTOR
FFFC:62 FA          783          DW    RESET         ;RESET VECTOR
FFFE:03 C8          784 IRQVECT  DW    NEWIRQ        ;INTERRUPT REQUEST VECTOR
0000:                62          include bank2
```

```
23 BANK2          Apple //c F8 monitor firmware      20-OCT-86  06:41 PAGE 93

0000:               2 ****************************************
0000:               3 *
0000:               4 * Bank 2 of the roms
0000:               5 *
0000:               6 ****************************************
----- NEXT OBJECT FILE NAME IS FIRM.2
C000:       C000    7           org   $C000
C000:              63           include mint        ;Mouse & acia interrupt handler
C000:       0100    1           ds    $C100-*,0
```

```
24 MINT           Mouse & serial interrupt stuff     20-OCT-86  06:41 PAGE 94

C100:               4 ****************************************
C100:               5 *
C100:               6 * Mouse interrupt handler
C100:               7 *
C100:               8 * MOUSEINT - Monitor's interrupt handler
C100:               9 *
C100:              10 * Returns C = 0 if interrupt handled
C100:              11 * If not mouse interrupt, Goes to aciaint
C100:              12 * New in this rom:
C100:              13 * If D7 of moumode = 1, mouse X and Y interrupts are not processed
C100:              14 * and are passed on to the user.
C100:              15 *
C100:              16 ****************************************
C100:       C100   17 mouseint  equ   *             ;Entry point if X & Y set up
C100:A9 0E         18           lda   #$0E          ;Clear status bits
C102:1C 7F 07      19           trb   moustat


C105:38            21           sec                  ;Assume interrupt not handled
C106:              22 * Check for vertical blanking interrupt
C106:AD 19 C0      23           lda   vblint         ;VBL interrupt?
C109:10 2B   C136  24           bpl   chkmou
C10B:8D 79 C0      25           sta   iouenbl        ;Enable iou access & clear VBL interrupt
C10E:A9 0C         26           lda   #vblmode       ;Should we leave vbl active?
C110:2C FF 07      27           bit   moumode
C113:D0 03   C118  28           bne   cvnovbl
C115:8D 5A C0      29           sta   iou+2          ;Disable VBL
C118:09 02         30 cvnovbl   ora   #movmode
C11A:8D 78 C0      31           sta   ioudsbl
C11D:2C 7F 06      32           bit   mouarm         ;VBL bit in arm isn't used
C120:D0 02   C124  33           bne   cvmoved
C122:A9 0C         34           lda   #vblmode       ;Didn't move
C124:2C 63 C0      35 cvmoved   bit   moubut         ;Button pressed?
C127:10 02   C12B  36           bpl   cvbut
C129:49 04         37           eor   #butmode       ;Clear the button bit
C12B:2D FF 07      38 cvbut     and   moumode        ;Which bits were set in the mode
C12E:0C 7F 07      39           tsb   moustat
C131:1C 7F 06      40           trb   mouarm
C134:69 FE         41           adc   #$FE           ;C=1 if int passes to user
C136:              42 * Check & update mouse movement
C136:       C136   43 chkmou    equ   *
C136:AD FF 07      44           lda   moumode        ;If D7 = 1, user better handle it
C139:30 72   C1AD  45           bmi   xmdone
C13B:AD 15 C0      46           lda   mouxint        ;Mouse interrupt?
C13E:0D 17 C0      47           ora   mouyint
C141:10 6A   C1AD  48           bpl   xmdone         ;If not return with C from vbl
C143:8A            49           txa                   ;Get X1 in A
C144:A2 00         50           ldx   #0
C146:2C 15 C0      51           bit   mouxint        ;X movement?
C149:30 0A   C155  52           bmi   cmxmov
C14B:98            53 cmloop    tya                   ;Get Y1 into A
C14C:49 80         54           eor   #$80           ;Complement direction
C14E:A2 80         55           ldx   #$80
C150:2C 17 C0      56           bit   mouyint
C153:10 39   C18E  57           bpl   cmnoy
C155:0A            58 cmxmov    asl   A
C156:BD 7F 04      59           lda   mouxl,x        ;A = current low byte
```

```
C159:B0 1A   C175    60           bcs   cmrght      ;Which way?
C15B:DD 7D 04        61           cmp   minxl,x     ;Move left
C15E:D0 08   C168    62           bne   cmlok
C160:BD 7F 05        63           lda   mouxh,x
C163:DD 7D 05        64           cmp   minxh,x
C166:F0 22   C18A    65           beq   cmnoint
C168:BD 7F 04        66 cmlok     lda   mouxl,x
C16B:D0 03   C170    67           bne   cmnt0       ;Borrow from high byte?
C16D:DE 7F 05        68           dec   mouxh,x
C170:DE 7F 04        69 cmnt0     dec   mouxl,x
C173:80 15   C18A    70           bra   cmnoint
C175:DD 7D 06        71 cmrght    cmp   maxxl,x     ;At high bound?
C178:D0 08   C182    72           bne   cmrok
C17A:BD 7F 05        73           lda   mouxh,x
C17D:DD 7D 07        74           cmp   maxxh,x
C180:F0 08   C18A    75           beq   cmnoint
C182:FE 7F 04        76 cmrok     inc   mouxl,x     ;Move right
C185:D0 03   C18A    77           bne   cmnoint
C187:FE 7F 05        78           inc   mouxh,x
C18A:E0 00           79 cmnoint   cpx   #0
C18C:F0 BD   C14B    80           beq   cmloop
C18E:8D 48 C0        81 cmnoy     sta   mouclr
C191:A9 02           82           lda   #movmode    ;Should we enable VBL?
C193:2D FF 07        83           and   moumode
C196:F0 09   C1A1    84           beq   cmnovbl     ;Branch if not
C198:8D 79 C0        85           sta   iouenbl
C19B:8D 5B C0        86           sta   iou+3       ;Enable VBL int
C19E:8D 78 C0        87           sta   ioudsbl
C1A1:09 20           88 cmnovbl   ora   #movarm     ;Mark that we moved
C1A3:0C 7F 06        89           tsb   mouarm
C1A6:A9 0E           90           lda   #$0E
C1A8:2D 7F 07        91           and   moustat
C1AB:69 FE           92           adc   #$FE        ;C=1 iff any bits were 1
C1AD:B0 05   C1B4    93 xmdone    bcs   aciaint     ;If not handled, try acia
C1AF:4C 84 C7        94           jmp   swrts2      ;Back we go
```



```
C1B2:                 96 *   This routine will determine if the source of
C1B2:                 97 * is either of the built in ACIAs.  If neither port
C1B2:                 98 * generated the interrupt, or the interrupt was due
C1B2:                 99 * to a transmit buffer empty, protocol converter, or
C1B2:                100 * 'unbuffered' receiver full, the carry is set indi-
C1B2:                101 * cating an externally serviced interrupt.
C1B2:                102 *   If the interrupt source was keyboard, 'buffered'
C1B2:                103 * serial input, or the DCD, the interrupt is serviced
C1B2:                104 * and the carry is cleared indicating interrupt was
C1B2:                105 * serviced. (DCD handshake replaces CTS.)
C1B2:                106 *   Location "ACIABUF" specifies which (if either) re-
C1B2:                107 * ceiver data is buffered.  For port 1 it must contain
C1B2:                108 * $C1, for port 2 a $C2.  Any other values are cause
C1B2:                109 * interrupts to pass to external (RAM based) routines.
C1B2:                110 *   Location "TYPHED" specifies whether Keyboard in-
C1B2:                111 * put should be buffered, ignored, or processed by
C1B2:                112 * RAM based routines.  If bit 7=1 and bit 6=0, key-
C1B2:                113 * board data is placed in the type-ahead buffer. If
C1B2:                114 * bit 6 is set the interrupt is cleared, but must
C1B2:                115 * be recognized and serviced by a RAM routine.  If
C1B2:                116 * both bits = 0, the interrupt is serviced, but the
C1B2:                117 * keyboard data is ignored.
C1B2:                118 *   While using type-ahead, Open-Apple CTRL-X will
C1B2:                119 * flush the buffer.  No other code is recognized.
C1B2:                120 *   If the source was an ACIA that has the transmit
C1B2:                121 * interrupt enabled, the original value of the ACIAs
C1B2:                122 * status registers is preserved. Automatic serial input
C1B2:                123 * buffering is not serviced from a port so configured.
C1B2:                124 * Interrupts originating from the protocol converter or
C1B2:                125 * keyboard (RAM serviced) do not inhibit serial buffering
C1B2:                126 * and are passed thru.  The RAM service routine can rec-
C1B2:                127 * ognize the interrupt source by a 1 state in bit 6 of
C1B2:                128 * the ACIAs status register.  The RAM service routine must
C1B2:                129 * cause the clearing of DSR (bit 6) AND make a second ac-
C1B2:                130 * cess to the status register before returning.
C1B2:                131 *
C1B2:                132 *
C1B2:38              133 notacia  sec                ;Not acia int
C1B3:60              134 acdone   rts
C1B4:        C1B4    135 aciaint  equ   *
C1B4:20 BA C1        136          jsr   aciaint2     ;Extra jsr since rest needs RTS
C1B7:4C 84 C7        137          jmp   swrts2
C1BA:        C1BA    138 aciaint2 equ   *
C1BA:A2 C2           139          ldx   #<comslot    ;Test port 2 first
C1BC:20 C2 C1        140          jsr   aciatst      ;Check for interrupt
C1BF:90 F2   C1B3    141          bcc   acdone       ;Return if interrupt done
C1C1:CA              142          dex                ;Try port 1
C1C2:BC 42 C1        143 aciatst  ldy   devno2,x     ;Get index for acia
C1C5:A9 04           144          lda   #$4          ;If xmit ints enabled pass to user
C1C7:59 FA BF        145          eor   scomd,y      ;Check if D<3>, D<2> = 01
C1CA:29 0C           146          and   #$0C         ;
C1CC:F0 E4   C1B2    147          beq   notacia      ;User better take it!
C1CE:B9 F9 BF        148          lda   sstat,y      ;Get status
C1D1:9D 38 04        149          sta   astat,x      ;Save it away
C1D4:10 DC   C1B2    150          bpl   notacia      ;No interrupt
C1D6:E0 C2           151 aitst2   cpx   #<comslot    ;C=1 if com port. Called from serout3
C1D8:B0 02   C1DC    152          bcs   aiport2      ;Invert DSR if port1
C1DA:49 40           153          eor   #$40
```

```
C1DC:3C 38 05      154 aiport2  bit   extint,x     ;Is DSR enabled?
C1DF:70 29   C20A  155          bvs   aipass       ;Yes, user wants it
C1E1:10 25   C208  156          bpl   aieatit      ;No, eat it
C1E3:90 23   C208  157          bcc   aieatit      ;Yes but I don't want it for port 1
C1E5:89 40         158          bit   #$40         ;Is DSR 1?
C1E7:F0 21   C20A  159          beq   aipass       ;If not, skip it
C1E9:              160 * It's a keyboard interrupt
C1E9:AD 00 C0      161          lda   kbd          ;Get the key
C1EC:A0 80         162          ldy   #$80
C1EE:20 28 C2      163          jsr   putbuf       ;Put it in the buffer
C1F1:C9 98         164          cmp   #$98         ;Is it a ^x?
C1F3:D0 0B   C200  165          bne   ainoflsh
C1F5:AD 62 C0      166          lda   butn1        ;And the closed apple?
C1F8:10 06   C200  167          bpl   ainoflsh
C1FA:8E FC 05      168          stx   twkey        ;Flush the type ahead buffer
C1FD:8E FF 06      169          stx   trkey
C200:AD 10 C0      170 ainoflsh lda   kbdstrb      ;Clear the keyboard
C203:              171 * $A0 $B0 table needed by serial firmware
C203:        C142  172 devno2   equ   *-sltdmy
C203:A0 B0         173          ldy   #$B0         ;Restore y
C205:B9 F9 BF      174          lda   sstat,y      ;Read status to clear int
C208:29 BF         175 aieatit  and   #$BF         ;Clear the DSR bit
C20A:0A            176 aipass   asl   A            ;Shift DSR into C
C20B:0A            177          asl   A
C20C:29 20         178          and   #$20         ;Is the receiver full?
C20E:F0 3E   C24E  179          beq   aciadone     ;If not, we're done
C210:B9 FA BF      180          lda   scomd,y      ;Are receive interrupts enabled?
C213:49 01         181          eor   #1           ;Check for D<1>,D<0> = 01
C215:29 03         182          and   #3
C217:D0 35   C24E  183          bne   aciadone     ;If not, were done
C219:8A            184          txa                ;Is this acia buffered?
C21A:4D FC 04      185          eor   aciabuf
C21D:D0 93   C1B2  186          bne   notacia      ;The user better handle it!
C21F:08            187          php                ;Save DSR status
C220:20 22 C3      188          jsr   getdata      ;Get char & check xon, etc
C223:90 28   C24D  189          bcc   aieat        ;Don't put in buffer if eaten
C225:A0 00         190          ldy   #0
C227:D0            191          dfb   $D0          ;BNE opcode to skip PHP
C228:        C228  192 putbuf   equ   *
C228:08            193          php
C229:DA            194          phx
C22A:48            195          pha
C22B:B9 7C 05      196          lda   twser,y      ;Get buffer pointer
C22E:AA            197          tax                ;Save it for later
C22F:1A            198          inc   A            ;Bump it to next free byte
C230:89 7F         199          bit   #$7F         ;Overflow?
C232:D0 01   C235  200          bne   pbok
C234:98            201          tya                ;Wrap pointer
C235:D9 7C 06      202 pbok     cmp   trser,y      ;Buffer full?
C238:F0 03   C23D  203          beq   pbfull
C23A:99 7C 05      204          sta   twser,y      ;Save the new pointer
C23D:68            205 pbfull   pla                ;Get the data
C23E:2C 14 C0      206          bit   rdramwrt
C241:8D 05 C0      207          sta   wrcardram    ;It goes to aux ram
C244:9D 00 08      208          sta   thbuf,x
C247:30 03   C24C  209          bmi   aiaux        ;Branch if we want aux
C249:8D 04 C0      210          sta   wrmainram
C24C:FA            211 aiaux    plx
```



```
C24D:28            212 aieat    plp                ;Get DSR status back
C24E:60            213 aciadone rts
```

```
24 MINT          Mouse & serial interrupt stuff      20-OCT-86  06:41 PAGE 99

C24F:                215 ****************************************
C24F:                216 *
C24F:                217 * SEROUT3 - Outputs a character to a acia
C24F:                218 * Inputs: A = char, X = Cn
C24F:                219 *
C24F:                220 ****************************************
C24F:          C24F  221 serout3   equ   *
C24F:20 55 C2        222           jsr   serout4
C252:4C 84 C7        223           jmp   swrts2
C255:          C255  224 serout4   equ   *             ;Entry point with rts
C255:48              225           pha                  ;Save the char
C256:2C AB C2        226           bit   sorts          ;Control char?
C259:F0 03   C25E    227           beq   sordy          ;Don't inc column if so
C25B:FE 38 07        228           inc   col,x
C25E:20 B2 C2        229 sordy     jsr   getstat2       ;Get acia status
C261:29 30           230           and   #$30           ;Y set by getstat
C263:C9 10           231           cmp   #$10
C265:D0 F7   C25E    232           bne   sordy
C267:BD B8 06        233           lda   flags,x        ;Is XON/XOFF enabled?
C26A:89 20           234           bit   #$20
C26C:F0 1F   C28D    235           beq   sook           ;Branch if not
C26E:EC FC 04        236           cpx   aciabuf        ;Is port interrupt driven?
C271:F0 13   C286    237           beq   sotst
C273:20 E9 C2        238           jsr   xrdnobuf       ;Get a char from the acia
C276:90 0E   C286    239           bcc   sotst          ;Branch if no char
C278:BC 34 C2        240           ldy   charptr,x      ;Get pointer to charbuf
C27B:99 FE 05        241           sta   charbuf,y      ;Save the character
C27E:BD B8 06        242           lda   flags,x        ;Set bit for char in buffer
C281:09 04           243           ora   #$04
C283:9D B8 06        244           sta   flags,x
C286:BD B8 06        245 sotst     lda   flags,x        ;Check if in xoff
C289:29 02           246           and   #$02
C28B:D0 D1   C25E    247           bne   sordy          ;Loop if not ready
C28D:BC 42 C1        248 sook      ldy   devno2,x
C290:68              249           pla
C291:48              250           pha                  ;Get char to XMIT
C292:99 F8 BF        251           sta   sdata,y        ;Out it goes
C295:3C B8 06        252           bit   flags,x        ;V=1 if LF after CR
C298:49 0D           253           eor   #$0D           ;check for CR.
C29A:0A              254           asl   A              ;preserve bit 7
C29B:D0 0D   C2AA    255           bne   sodone         ;branch if not CR.
C29D:50 06   C2A5    256           bvc   clrcol         ;branch if no LF after CR
C29F:A9 14           257           lda   #$14           ;Get LF*2
C2A1:6A              258           ror   A              ;no shift in high bit
C2A2:20 55 C2        259           jsr   serout4        ;Output the LF but don't echo it
C2A5:64 24           260 clrcol    stz   ch             ;0 position & column
C2A7:9E 38 07        261           stz   col,x
C2AA:68              262 sodone    pla                  ;Get the char back
C2AB:60              263 sorts     rts



C2AC:                265 ****************************************
C2AC:                266 *
C2AC:                267 * GETSTAT - Gets the status from a acia
C2AC:                268 * GETSTAT2 - Call from this side
C2AC:                269 * If interrupt, aciatst is called
C2AC:                270 * note: external interrupts are lost
```

```
24 MINT          Mouse & serial interrupt stuff      20-OCT-86  06:41 PAGE 100

C2AC:                271 * inputs: X = Cn
C2AC:                272 * outputs: A = status, X = Cn, Y = devno
C2AC:                273 *
C2AC:                274 ****************************************
C2AC:          C2AC  275 getstat   equ   *
C2AC:20 B2 C2        276           jsr   getstat2
C2AF:4C 84 C7        277           jmp   swrts2         ;Return to other side
C2B2:          C2B2  278 getstat2  equ   *
C2B2:08              279           php                  ;Save interrupt status
C2B3:78              280           sei
C2B4:BC 42 C1        281 gsttst    ldy   devno2,x       ;Get index into hardware
C2B7:B9 F9 BF        282           lda   sstat,y        ;Get the status
C2BA:10 05   C2C1    283           bpl   gstnoint       ;D7 = 1 if interrupt
C2BC:20 D6 C1        284           jsr   aitst2         ;Go service the interrupt
C2BF:80 F3   C2B4    285           bra   gsttst         ;Interrupt may have changed status
C2C1:28              286 gstnoint  plp                  ;Restore interrupt status
C2C2:60              287           rts
```

```
C2C3:                  289 ****************************************
C2C3:                  290 *   This is the serial input routine.  Carry
C2C3:                  291 * flag set indicates that returned data is
C2C3:                  292 * valid.
C2C3:                  293 *
C2C3:                  294 ****************************************
C2C3:          C2C3    295 xrdser    equ   *
C2C3:20 C9 C2          296           jsr   xrdser2
C2C6:4C 84 C7          297           jmp   swrts
C2C9:          C2C9    298 xrdser2   equ   *
C2C9:EC FC 04          299           cpx   aciabuf       ;is serial input buffered?
C2CC:D0 07   C2D5      300           bne   xnosbuf       ;(in english "NO SERIAL BUFFER")
C2CE:A0 00             301           ldy   #0            ;Y=0 for serial buffer
C2D0:20 FD C2          302           jsr   getbuf2       ;Any data in buffer?
C2D3:B0 1F   C2F4      303           bcs   xrddone
C2D5:                  304 *
C2D5:BD B8 06          305 xnosbuf   lda   flags,x       ;Is there a char in the onr byte buffer?
C2D8:89 04             306           bit   #$04
C2DA:F0 0D   C2E9      307           beq   xrdnobuf      ;Branch if not
C2DC:29 FB             308           and   #$FB          ;Clear the bit
C2DE:9D B8 06          309           sta   flags,x
C2E1:BC 34 C2          310           ldy   charptr,x
C2E4:B9 FE 05          311           lda   charbuf,y
C2E7:38                312           sec
C2E8:60                313           rts
C2E9:                  314 *
C2E9:20 B2 C2          315 xrdnobuf  jsr   getstat2      ;Get ACIA status
C2EC:29 08             316           and   #$8
C2EE:18                317           clc                 ;indicate no data
C2EF:F0 03   C2F4      318           beq   xrddone       ;Branch if no data!
C2F1:20 22 C3          319           jsr   getdata       ;Get data and check xon, etc
C2F4:60                320 xrddone   rts

C2F5:          C234    322 charptr   equ   *-$C1         ;Pointer to character buffers
C2F5:00 80             323           dfb   $0,$80



C2F7:                  325 ****************************************
C2F7:                  326 *
C2F7:                  327 * GETBUF - Gets a byte from the input buffer
C2F7:                  328 * Inputs: Y=0 for Serial buffer 80 for Keyboard buffer
C2F7:                  329 * C = 0 if no data C = 1 if data valid A = Data
C2F7:                  330 *
C2F7:                  331 ****************************************
C2F7:          C2F7    332 getbuf    equ   *
C2F7:20 FD C2          333           jsr   getbuf2
C2FA:4C 84 C7          334           jmp   swrts
C2FD:          C2FD    335 getbuf2   equ   *
C2FD:B9 7C 06          336           lda   trser,Y       ;Test for data in buffer
C300:D9 7C 05          337           cmp   twser,Y       ;If = then no data
C303:18                338           clc
C304:F0 1B   C321      339           beq   gbdone        ;Branch if empty
C306:48                340           pha                 ;Save current value
C307:1A                341           inc   A             ;Update the pointer
C308:89 7F             342           bit   #$7F          ;Overflow
C30A:D0 01   C30D      343           bne   gbnoovr
C30C:98                344           tya
```



```
C30D:99 7C 06          345 gbnoovr   sta   trser,y       ;Store the updated pointer
C310:7A                346           ply                 ;Get the old value of the pointer
C311:AD 13 C0          347           lda   rdramrd       ;Are we in main ram
C314:0A                348           asl   A             ;C=1 for Aux ram
C315:8D 03 C0          349           sta   rdcardram     ;Force Aux ram
C318:B9 00 08          350           lda   thbuf,Y       ;Get byte from buffer
C31B:B0 04   C321      351           bcs   gbdone        ;Branch if we were in aux bank
C31D:8D 02 C0          352           sta   rdmainram     ;Set back to main
C320:38                353           sec                 ;Mark data there
C321:60                354 gbdone    rts



C322:                  356 ****************************************
C322:                  357 *
C322:                  358 * GETDATA - Gets data from serial port
C322:                  359 * and checks for LF, XON, XOFF
C322:                  360 * inputs: Y = index to acia
C322:                  361 * outputs: A = data, Y dest, C = 1 if data ok = 0 if eaten
C322:                  362 *
C322:                  363 ****************************************
C322:          C322    364 getdata   equ   *
C322:B9 F8 BF          365           lda   sdata,y
C325:48                366           pha                 ;Save the data
C326:09 80             367           ora   #$80          ;Set D7 for compares
C328:A8                368           tay
C329:BD B8 06          369           lda   flags,x       ;Get options byte
C32C:89 08             370           bit   #$08          ;Eat linefeeds?
C32E:D0 04   C334      371           bne   gdnolf
C330:C0 8A             372           cpy   #lfeed        ;Is it a LF?
C332:F0 12   C346      373           beq   gdeat         ;Eat it if it is
C334:89 20             374 gdnolf    bit   #$20          ;Xon/XOFF enabled?
C336:F0 10   C348      375           beq   gdok
C338:C0 91             376           cpy   #xon          ;Is it an XON?
C33A:D0 04   C340      377           bne   gdnxon
C33C:29 FD             378           and   #$FD          ;Clear xoff bit
C33E:80 06   C346      379           bra   gdeat         ;And eat it
C340:C0 93             380 gdnxon    cpy   #xoff
C342:D0 04   C348      381           bne   gdok
C344:09 02             382           ora   #$02          ;Set xoff bit
C346:18                383 gdeat     clc
C347:B0                384           dfb   $B0           ;BCS opcode
C348:38                385 gdok      sec
C349:9D B8 06          386           sta   flags,x
C34C:68                387           pla
C34D:60                388           rts
C34E:                   64           include auxstuff    ;Auxillary move stuff
```

```
C34E:                4 ****************************************
C34E:                5 * NAME    : MOVEAUX
C34E:                6 * FUNCTION: PERFORM CROSSBANK MEMORY MOVE
C34E:                7 * INPUT   : A1=SOURCE ADDRESS
C34E:                8 *         : A2=SOURCE END
C34E:                9 *         : A4=DESTINATION START
C34E:               10 *         : CARRY SET=MAIN-->CARD
C34E:               11 *                CLR=CARD-->MAIN
C34E:               12 * OUTPUT  : NONE
C34E:               13 * VOLATILE: NOTHING
C34E:               14 * CALLS   : NOTHING
C34E:               15 ****************************************
C34E:       C34E    16 MOVEAUX  EQU  *
C34E:48             17          PHA                 ;SAVE AC
C34F:AD 13 C0       18          LDA  RDRAMRD        ;SAVE STATE OF
C352:48             19          PHA                 ; MEMORY FLAGS
C353:AD 14 C0       20          LDA  RDRAMWRT
C356:48             21          PHA
C357:               22 *
C357:               23 * SET FLAGS FOR CROSSBANK MOVE:
C357:               24 *
C357:90 08   C361   25          BCC  MOVEC2M        ;=>CARD-->MAIN
C359:8D 02 C0       26          STA  RDMAINRAM      ;SET FOR MAIN
C35C:8D 05 C0       27          STA  WRCARDRAM      ; TO CARD
C35F:B0 06   C367   28          BCS  MOVESTRT       ;=>(ALWAYS TAKEN)
C361:               29 *
C361:       C361    30 MOVEC2M  EQU  *
C361:8D 04 C0       31          STA  WRMAINRAM      ;SET FOR CARD
C364:8D 03 C0       32          STA  RDCARDRAM      ; TO MAIN
C367:               33 *
C367:       C367    34 MOVESTRT EQU  *
C367:B2 3C          35 MOVELOOP LDA  (A1L)          ;get a byte
C369:92 42          36          STA  (A4L)          ;move it
C36B:E6 42          37          INC  A4L
C36D:D0 02   C371   38          BNE  NEXTA1
C36F:E6 43          39          INC  A4H
C371:A5 3C          40 NEXTA1   LDA  A1L
C373:C5 3E          41          CMP  A2L
C375:A5 3D          42          LDA  A1H
C377:E5 3F          43          SBC  A2H
C379:E6 3C          44          INC  A1L
C37B:D0 02   C37F    45          BNE  CO1
C37D:E6 3D          46          INC  A1H
C37F:90 E6   C367    47 CO1      BCC  MOVELOOP       ;=>more to move
C381:               48 *
C381:8D 04 C0       49          STA  WRMAINRAM      ;CLEAR FLAG2
C384:68             50          PLA                 ;GET ORIGINAL STATE
C385:10 03   C38A   51          BPL  C03            ;=>IT WAS OFF
C387:8D 05 C0       52          STA  WRCARDRAM
C38A:       C38A    53 C03      EQU  *
C38A:8D 02 C0       54          STA  RDMAINRAM      ;CLEAR FLAG1
C38D:68             55          PLA                 ;GET ORIGINAL STATE
C38E:10 03   C393   56          BPL  MOVERET        ;=>IT WAS OFF
C390:8D 03 C0       57          STA  RDCARDRAM
C393:       C393    58 MOVERET  EQU  *
C393:68             59          PLA                 ;Restore AC
C394:4C 84 C7       60          JMP  SWRTS2
```



```
C397:               62 ****************************************
C397:               63 * NAME    : XFER
C397:               64 * FUNCTION: TRANSFER CONTROL CROSSBANK
C397:               65 * INPUT   : $03ED=TRANSFER ADDR
C397:               66 *         : CARRY SET=XFER TO CARD
C397:               67 *                CLR=XFER TO MAIN
C397:               68 *         : VFLAG CLR=USE STD ZP/STK
C397:               69 *                SET=USE ALT ZP/STK
C397:               70 * OUTPUT  : NONE
C397:               71 * VOLATILE: $03ED/03EE IN DEST BANK
C397:               72 * CALLS   : NOTHING
C397:               73 * NOTE    : ENTERED VIA JMP, NOT JSR
C397:               74 ****************************************
C397:               75 *
C397:       C397    76 XFER     EQU  *
C397:48             77          PHA                 ;SAVE AC ON CURRENT STACK
C398:               78 *
C398:               79 * COPY DESTINATION ADDRESS TO THE
C398:               80 *  OTHER BANK SO THAT WE HAVE IT
C398:               81 *  IN CASE WE DO A SWAP:
C398:               82 *
C398:AD ED 03       83          LDA  $03ED          ;GET XFERADDR LO
C39B:48             84          PHA                 ;SAVE ON CURRENT STACK
C39C:AD EE 03       85          LDA  $03EE          ;GET XFERADDR HI
C39F:48             86          PHA                 ;SAVE IT TOO
C3A0:               87 *
C3A0:               88 * SWITCH TO APPROPRIATE BANK:
C3A0:               89 *
C3A0:90 08   C3AA   90          BCC  XFERC2M        ;=>CARD-->MAIN
C3A2:8D 03 C0       91          STA  RDCARDRAM      ;SET FOR RUNNING
C3A5:8D 05 C0       92          STA  WRCARDRAM      ; IN CARD RAM
C3A8:B0 06   C3B0   93          BCS  XFERZP         ;=> always taken
C3AA:       C3AA    94 XFERC2M  EQU  *
C3AA:8D 02 C0       95          STA  RDMAINRAM      ;SET FOR RUNNING
C3AD:8D 04 C0       96          STA  WRMAINRAM      ; IN MAIN RAM
C3B0:               97 *
C3B0:       C3B0    98 XFERZP   EQU  *              ;SWITCH TO ALT ZP/STK
C3B0:68             99          PLA                 ;STUFF XFERADDR
C3B1:8D EE 03      100          STA  $03EE          ; HI AND
C3B4:68            101          PLA
C3B5:8D ED 03      102          STA  $03ED          ;  LO
C3B8:68            103          PLA                 ;RESTORE AC
C3B9:70 05   C3C0  104          BVS  XFERAZP        ;=>switch in alternate zp
C3BB:8D 08 C0      105          STA  SETSTDZP       ;else force standard zp
C3BE:50 03   C3C3  106          BVC  JMPDEST        ;=>always perform transfer
C3C0:8D 09 C0      107 XFERAZP  STA  SETALTZP       ;switch in alternate zp
C3C3:4C EB C7      108 JMPDEST  JMP  SWXFGO2        ;Back we go
C3C6:              109 ****************************************
C3C6:               65          include banger2     ;Diagnostic routines
```

```
C3C6:                 3 ****************************************
C3C6:                 4 *
C3C6:                 5 * Here is the rest of the diagnostic stuff
C3C6:                 6 * the first part has been moved into the $D000 space
C3C6:                 7 * to make desperately needed room
C3C6:                 8 *
C3C6:                 9 ****************************************
C3C6:       C3C6     10 TSTMEM    equ   *
C3C6:86 01            11           stx   $01
C3C8:86 02            12           stx   $02
C3CA:86 03            13           stx   $03
C3CC:A2 04            14           ldx   #4            ;do RAM $100-$FFFF five times
C3CE:86 04            15           stx   $04
C3D0:85 05            16 MEM1      STA   $05           ;keep acc in a safe place
C3D2:A2 04            17           ldx   #4
C3D4:64 01            18           stz   $01
C3D6:E6 01            19           inc   1             ;point to page 1 first
C3D8:A8               20 mem2      tay                 ;save ACC in Y for now
C3D9:8D 83 C0         21           sta   lcbank2       ;anticipate not $C000 range...
C3DC:8D 83 C0         22           sta   lcbank2
C3DF:A5 01            23           lda   $01           ;get page address
C3E1:29 F0            24           and   #$F0          ;test for $C0-$CF range
C3E3:C9 C0            25           cmp   #$C0
C3E5:D0 0C   C3F3     26           bne   mem3          ;branch if not...
C3E7:AD 8B C0         27           lda   lcbank1
C3EA:AD 8B C0         28           lda   lcbank1       ;select primary $D000 space
C3ED:A5 01            29           lda   $01
C3EF:69 0F            30           adc   #$F           ;Plus carry =+$10
C3F1:D0 02   C3F5     31           bne   mem4          ;branch always taken
C3F3:A5 01            32 mem3      lda   $01
C3F5:85 03            33 mem4      sta   $03
C3F7:98               34           tya                 ;restore pattern to ACC
C3F8:A0 00            35           ldy   #$00          ;fill this page with the pattern
C3FA:18               36 mem5      clc
C3FB:7D 2A C8         37           adc   ntbl,x
C3FE:91 02            38           sta   ($02),y
C400:CA               39           dex                 ;keep x in the range 0-4
C401:10 02   C405     40           bpl   mem6
C403:A2 04            41           ldx   #4
C405:C8               42 mem6      iny                 ;all 256 filled yet?
C406:D0 F2   C3FA     43           bne   mem5          ;branch if not
C408:E6 01            44           inc   1             ;bump page #
C40A:D0 CC   C3D8     45           bne   mem2          ;loop through $0100 to $FF00

C40C:E6 01            47           inc   $01           ;point to page 1 again
C40E:A2 04            48           LDX   #4
C410:A5 05            49           LDA   $05
C412:A8               50 mem7      tay                 ;save ACC in Y for now
C413:AD 83 C0         51           lda   lcbank2       ;anticipate not $C000 range...
C416:AD 83 C0         52           lda   lcbank2
C419:A5 01            53           lda   $01           ;get page address
C41B:29 F0            54           and   #$F0          ;test for $C0-$CF range
C41D:C9 C0            55           cmp   #$C0
C41F:D0 09   C42A     56           bne   mem8          ;branch if not...
C421:AD 8B C0         57           lda   lcbank1       ;select primary $D000 space
C424:A5 01            58           lda   $01
C426:69 0F            59           adc   #$F           ;Plus carry =+$10
C428:D0 02   C42C     60           bne   mem9          ;branch always taken
```



```
C42A:A5 01            61 mem8      lda   $01
C42C:85 03            62 mem9      sta   $03
C42E:98               63           tya                 ;restore pattern to ACC
C42F:A0 00            64           ldy   #$00          ;fill this page with the pattern
C431:18               65 memA      clc
C432:7D 2A C8         66           adc   ntbl,x
C435:51 02            67           eor   ($02),y
C437:D0 39   C472     68           bne   MEMERROR      ;if any bits are different, give up!!!
C439:B1 02            69           lda   ($02),y       ;restore correct pattern
C43B:CA               70           dex                 ;keep x in the range 0-4
C43C:10 02   C440     71           bpl   memB
C43E:A2 04            72           ldx   #4
C440:C8               73 memB      iny                 ;all 256 filled yet?
C441:D0 EE   C431     74           bne   memA          ;branch if not
C443:E6 01            75           inc   1             ;bump page #
C445:D0 CB   C412     76           bne   mem7          ;loop through $0100 to $FF00
C447:6A               77           ror   a             ;change ACC for next pass
C448:2C 19 C0         78           bit   rdvblbar      ; use RDVBL for a little randomness...
C44B:10 02   C44F     79           bpl   memC
C44D:49 A5            80           eor   #$A5
C44F:C6 04            81 memC      dec   $04           ;have 5 passes been done yet?
C451:30 03   C456     82           bmi   memD          ;skip if yes
C453:4C D0 C3         83           jmp   mem1          ;start next pass


C456:AA               85 memD      TAX                 ;save acc
C457:2C 13 C0         86           BIT   rdramrd       ;main or aux ram ?
C45A:30 10   C46C     87           BMI   MEMF          ;skip if aux ram
C45C:8A               88           txa
C45D:8D 05 C0         89           STA   wrcardram     ;enable aux mem write
C460:8D 03 C0         90           STA   rdcardram     ;enable aux mem read
C463:8D 09 C0         91           STA   setaltzp      ;swap in alt zero page
C466:8D 81 C0         92           STA   ROMIN         ;Force rom enable
C469:4C 97 D4         93           jmp   TSTZPG        ; and test it!

C46C:8D 08 C0         95 MEMF      STA   setstdzp      ;swap in main zero page
C46F:4C EF C4         96           JMP   SWCHTST
```

```
C472:38          98 MEMERROR  sec             ;indicate main ram failure
C473:AA          99 BADBITS   tax             ;save bit pattern in x for now
C474:AD 13 C0   100           lda  rdramrd    ;main or aux mem?
C477:B8         101           clv             ;with V-FLG
C478:10 03  C47D 102           bpl  bbits1     ;branch if primary bank
C47A:2C 2A C8   103           bit  setv
C47D:A9 A0      104 bbits1    lda  #$A0       ;try to clear video screen
C47F:A0 06      105           ldy  #6
C481:99 FE BF   106 clrsts    sta  ioadr-2,y
C484:99 06 C0   107           sta  ioadr+6,y
C487:88         108           dey
C488:88         109           dey
C489:D0 F6  C481 110          bne  clrsts
C48B:8D 51 C0   111           sta  txtset
C48E:8D 54 C0   112           sta  txtpage1
C491:99 00 04   113 clrs      sta  $400,y
C494:99 00 05   114           sta  $500,y
C497:99 00 06   115           sta  $600,y
C49A:99 00 07   116           sta  $700,y
C49D:C8         117           iny
C49E:D0 F1  C491 118          bne  clrs
C4A0:8A         119           txa             ;test for switch test failure
C4A1:F0 27  C4CA 120          beq  BADSWTCH   ;branch if it was a switch
C4A3:A0 03      121           ldy  #3
C4A5:B0 02  C4A9 122          bcs  badmain    ;branch if ZP ok
C4A7:A0 05      123           ldy  #5
C4A9:A9 AA      124 badmain   lda  #$AA       ;mark aux report with an asterisks
C4AB:50 03  C4B0 125          bvc  badprim
C4AD:8D B0 05   126           sta  screen-8
C4B0:B9 66 C8   127 badprim   lda  rmess,y
C4B3:99 B1 05   128           sta  screen-7,y
C4B6:88         129           dey
C4B7:10 F7  C4B0 130          bpl  badprim    ;message is either "RAM" or "RAM ZP"
C4B9:A0 10      131           ldy  #$10       ;print bits
C4BB:8A         132 bbits2    txa
C4BC:4A         133           lsr  a
C4BD:AA         134           tax
C4BE:A9 58      135           lda  #$58       ;bits are printed as ascii 0 or 1
C4C0:2A         136           rol  a
C4C1:99 B6 05   137           sta  screen-2,y
C4C4:88         138           dey
C4C5:88         139           dey
C4C6:D0 F3  C4BB 140          bne  bbits2
C4C8:F0 FE  C4C8 141 hangx    beq  hangx      ;hang forever and ever
```



```
C4CA:A2 02      143 BADSWTCH  ldx  #2
C4CC:7A         144           ply
C4CD:08         145           php
C4CE:BD 6C C8   146 bswtch1   lda  smess,x    ;anticipate MMU error
C4D1:28         147           plp
C4D2:08         148           php
C4D3:90 03  C4D8 149          bcc  bswtch2    ;branch if not IOU error
C4D5:BD 6F C8   150           lda  smess+3,x  ;anticipate IOU error
C4D8:C0 06      151 bswtch2   cpy  #6         ;compare with where we left off
C4DA:90 0B  C4E7 152          bcc  bswtch3    ;skip if MMU
C4DC:C0 08      153           cpy  #8
C4DE:90 04  C4E4 154          bcc  bswtch2a   ;skip if GLU (ioudis or dhires failure)
C4E0:C0 11      155           cpy  #$11
C4E2:90 03  C4E7 156          bcc  bswtch3    ;skip if IOU
C4E4:BD 72 C8   157 bswtch2a  lda  smess+6,x  ;GLU error (ioudis failure)
C4E7:9D B8 05   158 bswtch3   sta  screen,x
C4EA:CA         159           dex
C4EB:10 E1  C4CE 160          bpl  bswtch1    ;print "MMU", "IOU" or "GLU"
C4ED:30 FE  C4ED 161 hangy    bmi  hangy      ;branch forever


C4EF:A0 01      163 SWCHTST   ldy  #MMUIDX
C4F1:A9 7F      164 swtst1    lda  #$7F
C4F3:6A         165 swtst2    ror  a          ;set IOU/MMU switches to match A
C4F4:BE 2F C8   166           ldx  SWTBL0,y
C4F7:F0 0F  C508 167          beq  swtst4     ;branch if done setting switches
C4F9:90 03  C4FE 168          bcc  swtst3     ;branch if setting switch to 0-state
C4FB:BE 41 C8   169           ldx  SWTBL1,y   ;else get index to set switch to 1
C4FE:9D FF BF   170 swtst3    sta  ioadr-1,x  ;set switch
C501:C8         171           iny
C502:D0 EF  C4F3 172          bne  swtst2     ;branch always taken...
C504:           173 *
C504:AE 30 C0   174 click     ldx  spkr
C507:2A         175           rol  a
C508:88         176 swtst4    dey
C509:BE 53 C8   177           ldx  RSWTBL,y   ;now verify the settings just made
C50C:F0 13  C521 178          beq  swtst6     ;branch if done this pass
C50E:30 F4  C504 179          bmi  click      ;branch if this switch no to be verified.
C510:2A         180           rol  a
C511:90 07  C51A 181          bcc  swtst5
C513:1E 00 C0   182           asl  ioadr,x
C516:90 1F  C537 183          bcc  swerr
C518:B0 EE  C508 184          bcs  swtst4     ;branch always
C51A:1E 00 C0   185 swtst5    asl  ioadr,x
C51D:B0 18  C537 186          bcs  swerr
C51F:90 E7  C508 187          bcc  swtst4     ;branch always
C521:           188 *
C521:2A         189 swtst6    rol  a          ;restore original value
C522:C8         190           iny             ; and IOU/MMU index
C523:38         191           sec
C524:E9 01      192           sbc  #1         ;try next pattern
C526:B0 CB  C4F3 193          bcs  swtst2
C528:88         194           dey             ;was MMU just tested?
C529:F0 08  C533 195          beq  swtst7     ;yes, go test IOU
C52B:C0 08      196           cpy  #IOUIDX-1  ;was IOU just tested?
C52D:D0 10  C53F 197          bne  BIGLOOP    ;no, go loop again
C52F:A0 11      198           ldy  #GLUIDX    ;yes, go test IOUDIS switch
C531:D0 BE  C4F1 199          bne  swtst1     ;branch always
```

```
C533:A0 09        200 swtst7   ldy  #IOUIDX
C535:D0 BA  C4F1  201          bne  swtst1       ;branch always
C537:             202 *
C537:5A           203 swerr    phy               ;save y to distinguish from MMU or GLU
C538:             204 ;                           failure
C538:A2 00        205          ldx  #0           ;indicate switch error
C53A:C0 0A        206          cpy  #IOUIDX+1    ;set carry if IOU was cause
C53C:4C 7D C4     207          jmp  bbits1
```



```
C53F:46 80        209 BIGLOOP  lsr  $80
C541:D0 AC  C4EF  210          bne  SWCHTST
C543:A9 A0        211 blp2     lda  #$A0
C545:A0 00        212          ldy  #0
C547:99 00 04     213 blp3     sta  $400,y       ;clear screen for success message
C54A:99 00 05     214          sta  $500,y
C54D:99 00 06     215          sta  $600,y
C550:99 00 07     216          sta  $700,y
C553:C8           217          iny
C554:D0 F1  C547  218          bne  blp3
C556:AD 61 C0     219 blp4     LDA  butn0        ;test for both Open and Closed Apple
C559:2D 62 C0     220          AND  butn1        ; pressed
C55C:0A           221          asl  a            ;put result in carry
C55D:E6 FF        222          INC  $FF
C55F:A5 FF        223          LDA  $FF
C561:90 03  C566  224          bcc  dquit
C563:4C 8E D4     225          jmp  DIAGS
C566:             226 *
C566:AD 51 C0     227 dquit    lda  txtset       ;put success message on the screen
C569:A0 08        228          ldy  #8
C56B:B9 75 C8     229 suc2     lda  success,y
C56E:99 B8 05     230          sta  SCREEN,y
C571:88           231          dey
C572:10 F7  C56B  232          bpl  suc2
C574:30 E0  C556  233          bmi  blp4         ;loop forever

C576:       000A  235          ds   $c580-*,$00
C580:              66          include rw.slinky
```

```
C580:                    2 ************************************************
C580:                    3 * PREAD - Reads bytes from card into the Apple
C580:                    4 * D7 of the address = 1 if aux ram
C580:                    5 ************************************************

C580:DA                  7 sl.pread  phx               ;save x
C581:AE 78 06            8           ldx   sl.lcstate  ;get language card state
C584:FE 00 C0            9           inc   $C000,x     ;restore it, the rom is ayway
C587:FA                 10           plx               ;restore x

C588:A5 49              12           lda   paddr       ;Move the address
C58A:9D F8 BF           13           sta   addrl,x
C58D:A5 4A              14           lda   paddr+1
C58F:9D F9 BF           15           sta   addrm,x
C592:A5 4B              16           lda   paddr+2
C594:29 7F              17           and   #$7F        ;Mask off high bit
C596:D9 B8 03           18           cmp   numbanks,y  ;Valid address
C599:B0 53   C5EE       19           bge   prbad
C59B:9D FA BF           20           sta   addrh,x
C59E:2C 14 C0           21           bit   rdramwrt    ;Save current bank
C5A1:08                 22           php
C5A2:8D 04 C0           23           sta   wrmainram   ;Assume main
C5A5:24 4B              24           bit   paddr+2     ;If D7 = 1 then aux
C5A7:10 03   C5AC       25           bpl   prmain
C5A9:8D 05 C0           26           sta   wrcardram   ;Its the card ram
C5AC:A0 00              27 prmain    ldy   #0
C5AE:A5 48              28           lda   pcount+1    ;More than a page to move?
C5B0:8D F8 05           29           sta   yval
C5B3:F0 14   C5C9       30           beq   prlast
C5B5:BD FB BF           31 prloop    lda   data,x      ;Get a byte
C5B8:91 45              32           sta   (pbuff),y
C5BA:C8                 33           iny
C5BB:BD FB BF           34           lda   data,x
C5BE:91 45              35           sta   (pbuff),y
C5C0:C8                 36           iny
C5C1:D0 F2   C5B5       37           bne   prloop
C5C3:E6 46              38           inc   pbuff+1     ;Bump buffer pointer to next page
C5C5:C6 48              39           dec   pcount+1    ;Dec page count
C5C7:D0 EC   C5B5       40           bne   prloop
C5C9:A5 47              41 prlast    lda   pcount      ;Any bytes left to do?
C5CB:F0 16   C5E3       42           beq   prdone
C5CD:8D 78 05           43           sta   xval        ;Save bytes moved
C5D0:4A                 44           lsr   A           ;C = 1 if odd # of bytes
C5D1:B0 06   C5D9       45           bcs   prodd
C5D3:BD FB BF           46 prloop2   lda   data,x
C5D6:91 45              47           sta   (pbuff),y
C5D8:C8                 48           iny
C5D9:BD FB BF           49 prodd     lda   data,x
C5DC:91 45              50           sta   (pbuff),y
C5DE:C8                 51           iny
C5DF:C4 47              52           cpy   pcount
C5E1:D0 F0   C5D3       53           bne   prloop2
C5E3:8D 04 C0           54 prdone    sta   wrmainram   ;Fix main / aux ram
C5E6:28                 55           plp
C5E7:10 03   C5EC       56           bpl   prmain2
C5E9:8D 05 C0           57           sta   wrcardram
C5EC:80 05   C5F3       58 prmain2   bra   prbadz
```



```
C5EE:A9 2D              60 prbad     lda   #badblk     ;Invalid address
C5F0:8D F8 04           61           sta   error
C5F3:8D 81 C0           62 prbadz    sta   romin       ;put the rom back in
C5F6:60                 63           rts
```

```
C5F7:                 65 ************************************************
C5F7:                 66 * PWRITE  - Writes bytes from Apple to card
C5F7:                 67 * D7 of the address = 1 if aux ram
C5F7:                 68 ************************************************

C5F7:DA               70 sl.pwrite phx              ;save x
C5F8:AE 78 06         71           ldx  sl.lcstate  ;get language card state
C5FB:FE 00 C0         72           inc  $C000,x     ;restore it, the rom is ayway
C5FE:FA               73           plx              ;restore x

C5FF:A5 49            75           lda  paddr       ;Move the address
C601:9D F8 BF         76           sta  addrl,x
C604:A5 4A            77           lda  paddr+1
C606:9D F9 BF         78           sta  addrm,x
C609:A5 4B            79           lda  paddr+2
C60B:29 7F            80           and  #$7F        ;Mask off high bit
C60D:D9 B8 03         81           cmp  numbanks,y  ;Valid address
C610:B0 DC   C5EE     82           bge  prbad
C612:9D FA BF         83           sta  addrh,x
C615:2C 13 C0         84           bit  rdramrd     ;Save current bank
C618:08               85           php
C619:8D 02 C0         86           sta  rdmainram   ;Assume main
C61C:24 4B            87           bit  paddr+2     ;If D7 = 1 then aux
C61E:10 03   C623     88           bpl  pwmain
C620:8D 03 C0         89           sta  rdcardram   ;Its the card ram
C623:A0 00            90 pwmain    ldy  #0
C625:A5 48            91           lda  pcount+1    ;More than a page to move?
C627:8D F8 05         92           sta  yval
C62A:F0 14   C640     93           beq  pwlast
C62C:B1 45            94 pwloop    lda  (pbuff),y   ;Get a byte
C62E:9D FB BF         95           sta  data,x
C631:C8               96           iny
C632:B1 45            97           lda  (pbuff),y
C634:9D FB BF         98           sta  data,x
C637:C8               99           iny
C638:D0 F2   C62C    100           bne  pwloop
C63A:E6 46           101           inc  pbuff+1     ;Bump buffer pointer to next page
C63C:C6 48           102           dec  pcount+1    ;Dec page count
C63E:D0 EC   C62C    103           bne  pwloop
C640:A5 47           104 pwlast    lda  pcount      ;Any bytes left to do?
C642:8D 78 05        105           sta  xval
C645:F0 13   C65A    106           beq  pwdone
C647:4A              107           lsr  A           ;C = 1 if odd # of bytes
C648:B0 06   C650    108           bcs  pwodd
C64A:B1 45           109 pwloop2   lda  (pbuff),y
C64C:9D FB BF        110           sta  data,x
C64F:C8              111           iny
C650:B1 45           112 pwodd     lda  (pbuff),y
C652:9D FB BF        113           sta  data,x
C655:C8              114           iny
C656:C4 47           115           cpy  pcount
C658:D0 F0   C64A    116           bne  pwloop2
C65A:8D 02 C0        117 pwdone    sta  rdmainram   ;Fix main / aux ram
C65D:28              118           plp
C65E:10 03   C663    119           bpl  pwmain2
C660:8D 03 C0        120           sta  rdcardram
C663:8D 81 C0        121 pwmain2   sta  romin        ;put the rom back in
C666:60              122           rts
```



```
C667:                124 ************************************
C667:                125 * Various data tables
C667:                126 ************************************

C667:                128 * Status info table
C667:        C667    129 stattbl  equ  *
C667:F8              130          dfb  $F8            ;Status byte
C668:00 00 00        131          dfb  $00,$00,$00    ;Size
C66B:07              132          dfb  7              ;Name length
C66C:                133          MSB  OFF
C66C:52 41 4D 43     134          asc  'RAMCARD'
C673:20 20 20 20     135          asc  '              '
C67C:                136          MSB  ON
C67C:00 00           137          dw   0              ;Type subtype
C67E:01 01           138          dw   revnum         ;Version

C680:03 03 03        140 parmtbl  dfb  03,03,03       ;Table of parameters
C683:03 03 03        141          dfb  03,03,03
C686:01 01 03        142          dfb  01,01,03
C689:03 01 01        143          dfb  03,01,01
C68C:01 01 01        144          dfb  01,01,01
C68F:01 04 04        145          dfb  01,04,04
C692:04 04 FF        146          dfb  04,04,$FF
C695:FF FF FF        147          dfb  $FF,$FF,$FF
C698:FF FF           148          dfb  $FF,$FF

C69A:        C69A    150 cmdtbl   equ  *              ;Table of command addresses
C69A:52              151          dfb  >pstat0-1      ;Status unit 0
C69B:73              152          dfb  >sl.pstatus-1  ;Status
C69C:2F              153          dfb  >pzcmd-1       ;Read block unit 0
C69D:B8              154          dfb  >prdblk-1      ;Read block
C69E:2F              155          dfb  >pzcmd-1       ;Write block unit 0
C69F:BC              156          dfb  >pwrblk-1      ;Write block
C6A0:2F              157          dfb  >pzcmd-1       ;Format unit 0
C6A1:38              158          dfb  >iorts-1       ;Format
C6A2:69              159          dfb  >pcntl-1       ;Control unit 0
C6A3:69              160          dfb  >pcntl-1       ;Control
C6A4:38              161          dfb  >iorts-1       ;Init unit 0
C6A5:38              162          dfb  >iorts-1       ;Init
C6A6:2F              163          dfb  >pzcmd-1       ;Open unit 0
C6A7:2F              164          dfb  >pzcmd-1       ;Open
C6A8:2F              165          dfb  >pzcmd-1       ;Close unit 0
C6A9:2F              166          dfb  >pzcmd-1       ;Close
C6AA:2F              167          dfb  >pzcmd-1       ;Read unit 0
C6AB:39              168          dfb  >pread2.z-1    ;Read
C6AC:2F              169          dfb  >pzcmd-1       ;Write unit 0
C6AD:3C              170          dfb  >pwrite2-1     ;Write
C6AE:45              171          dfb  >xstatus-1     ;ProDOS status call
C6AF:9D              172          dfb  >xread-1       ;ProDOS read call
C6B0:A1              173          dfb  >xwrite-1      ;ProDOS write call
C6B1:38              174          dfb  >iorts-1       ;ProDOS format call
C6B2:3F              175          dfb  >dosconv2-1    ;Dos Command
C6B3:42              176          dfb  >xdiag-1       ;Diagnostics !
```

```
C6B4:DA           178 swsl.bt   phx               ;save x
C6B5:20 16 C8     179           jsr   getlc       ;get language state
C6B8:5A           180           phy               ;save it
C6B9:8C 78 06     181           sty   sl.lcstate  ;save it here too
C6BC:20 EF D8     182           jsr   boot.sl     ;do the boot
C6BF:4C 0E C8     183           jmp   fixlc       ;restore language card state and return

C6C2:DA           185 sw.setmou phx               ;save x
C6C3:20 16 C8     186           jsr   getlc       ;get language card state
C6C6:5A           187           phy               ;save it
C6C7:20 21 D6     188           jsr   x.setmou    ;set the mouse mode to a
C6CA:4C 0E C8     189           jmp   fixlc       ;restore language card state and return

C6CD:DA           191 sw.mtstint phx              ;save x
C6CE:20 16 C8     192           jsr   getlc       ;get language card state
C6D1:5A           193           phy               ;save it
C6D2:20 C2 D6     194           jsr   x.mtstint   ;check mouse status bits
C6D5:4C 0E C8     195           jmp   fixlc       ;restore language card state and return

C6D8:DA           197 sw.mread  phx               ;save x
C6D9:20 16 C8     198           jsr   getlc       ;get language card state
C6DC:5A           199           phy               ;save it
C6DD:20 79 D6     200           jsr   x.mread     ;updates the mouse screen holes
C6E0:4C 0E C8     201           jmp   fixlc       ;restore language card state and return

C6E3:DA           203 sw.mclear phx               ;save x
C6E4:20 16 C8     204           jsr   getlc       ;get language card state
C6E7:5A           205           phy               ;save it
C6E8:20 68 D6     206           jsr   x.mclear    ;sets the mouse to 0,0
C6EB:4C 0E C8     207           jmp   fixlc       ;restore language card state and return

C6EE:DA           209 sw.mclamp phx               ;save x
C6EF:20 16 C8     210           jsr   getlc       ;get language card state
C6F2:5A           211           phy               ;save it
C6F3:20 A3 D6     212           jsr   x.mclamp    ;store new mouse bounds
C6F6:4C 0E C8     213           jmp   fixlc       ;restore language card state and return

C6F9:DA           215 sw.mhome  phx               ;save x
C6FA:20 16 C8     216           jsr   getlc       ;get language card state
C6FD:5A           217           phy               ;save it
C6FE:20 51 D6     218           jsr   x.mhome     ;clear mouse position and status
C701:4C 0E C8     219           jmp   fixlc       ;restore language card state and return

C704:DA           221 sw.initmouse phx            ;save x
C705:20 16 C8     222           jsr   getlc       ;get language card state
C708:5A           223           phy               ;save it
C709:20 00 D6     224           jsr   i.nitmouse  ;reset the mouse
C70C:4C 0E C8     225           jmp   fixlc       ;restore language card state and return

C70F:       000D  227           ds    $C71C-*,00
C71C:              67           include mcode.x.aux
```



```
C71C:              2 ************************************************
C71C:              3 * the following code had better start at $C71C or else
C71C:              4 ************************************************

C71C:8D 28 C0      6           sta   rombank
C71F:4C C2 C6      7           jmp   sw.setmou     ;do the real thing

C722:8D 28 C0      9           sta   rombank
C725:4C CD C6     10           jmp   sw.mtstint    ;do the real thing

C728:8D 28 C0     12           sta   rombank
C72B:4C D8 C6     13           jmp   sw.mread      ;do the real thing

C72E:8D 28 C0     15           sta   rombank
C731:4C E3 C6     16           jmp   sw.mclear     ;do the real thing

C734:8D 28 C0     18           sta   rombank
C737:4C EE C6     19           jmp   sw.mclamp     ;do the real thing

C73A:8D 28 C0     21           sta   rombank
C73D:4C F9 C6     22           jmp   sw.mhome      ;do the real thing

C740:8D 28 C0     24           sta   rombank
C743:4C 04 C7     25           jmp   sw.initmouse  ;do the real thing

C746:8D 28 C0     27 m..oveirq sta   rombank
C749:4C 9A CF     28           jmp   m.oveirq

C74C:8D 28 C0     30           sta   rombank
C74F:4C B4 C6     31           jmp   swsl.bt

C752:8D 28 C0     33           sta   rombank
C755:DA           34           phx                 ;save x
C756:20 16 C8     35           jsr   getlc         ;get language card state
C759:5A           36           phy                 ;save it
C75A:8C 78 06     37           sty   sl.lcstate    ;save it here too
C75D:20 00 D8     38           jsr   execute       ;do something with slinky
C760:4C 0E C8     39           jmp   fixlc         ;restore language card state and return

C763:       001D  41           ds    $c780-*,$00
C780:             68           include switcher2   ;Bank switch stuff @ 2:C780
```

```
C780:        0000    2          ds    $C780-*,$00
C780:                3 ***************************************
C780:                4 *
C780:                5 * SWITCHING ROUTINES
C780:                6 *
C780:                7 ***************************************
C780:8D 28 C0        8 swrti2    sta   rombank
C783:40              9           rti
C784:8D 28 C0       10 swrts2    sta   rombank
C787:60             11           rts
C788:8D 28 C0       12 swreset2  sta   rombank
C78B:4C 62 FA       13           jmp   reset
C78E:8D 28 C0       14 swirq2    sta   rombank       ;Irq entry
C791:2C 87 C7       15           bit   swrtsop
C794:4C 04 C8       16           jmp   irqent
C797:8D 28 C0       17 swsthk2   sta   rombank
C79A:4C 80 C8       18           jmp   pcnv
C79D:8D 28 C0       19 swzzqt2   sta   rombank       ;Mouse basic routines
C7A0:4C 00 D4       20           jmp   basicin
C7A3:8D 28 C0       21           sta   rombank       ;Set terminal mode
C7A6:4C F1 C7       22           jmp   swsttm3
C7A9:8D 28 C0       23           sta   rombank       ;Jump to command routine
C7AC:4C 06 C8       24           jmp   swcmd3
C7AF:8D 28 C0       25           sta   rombank       ;Aux move
C7B2:4C 4E C3       26           jmp   moveaux
C7B5:8D 28 C0       27           sta   rombank       ;XFER
C7B8:4C 97 C3       28           jmp   xfer
C7BB:8D 28 C0       29           sta   rombank       ;Mouse interrupt handler
C7BE:4C 00 C1       30           jmp   mouseint
C7C1:8D 28 C0       31           sta   rombank       ;Diagnostics
C7C4:4C 8E D4       32           jmp   diags
C7C7:8D 28 C0       33           sta   rombank       ;Appletalk
C7CA:4C 80 C5       34           jmp   atalk
C7CD:8D 28 C0       35           sta   rombank       ;Serial output
C7D0:4C 4F C2       36           jmp   serout3
C7D3:8D 28 C0       37           sta   rombank       ;Get status
C7D6:4C AC C2       38           jmp   getstat
C7D9:8D 28 C0       39           sta   rombank       ;Read from serial port
C7DC:4C C3 C2       40           jmp   xrdser
C7DF:8D 28 C0       41           sta   rombank       ;Get char from buffer
C7E2:4C F7 C2       42           jmp   getbuf
C7E5:8D 28 C0       43           sta   rombank
C7E8:4C C5 D4       44           jmp   zznm
C7EB:8D 28 C0       45 swxfgo2   sta   rombank       ;Go to users xfer dest
C7EE:6C ED 03       46           jmp   ($3ED)
C7F1:DA             47 swsttm3   phx                  ;Save X
C7F2:20 16 C8       48           jsr   getlc
C7F5:5A             49           phy
C7F6:20 A0 D1       50           jsr   setterm
C7F9:80 13   C80E   51           bra   fixlc         ;Fix Language card and return

C7FB:        0008   53           ds    $C803-*,0     ;$C803 interrupt entry point
C803:4C 8E C7       54           jmp   swirq2

C806:DA             56 swcmd3    phx                  ;Go to the command routine
C807:20 16 C8       57           jsr   getlc         ;Get language card state
C80A:5A             58           phy                  ;Save it
C80B:20 00 D0       59           jsr   command
```



```
C80E:FA             60 fixlc     plx
C80F:FE 00 C0       61           inc   $C000,x       ;Restore LC
C812:FA             62           plx                  ;Restore real X
C813:4C 84 C7       63           jmp   swrts2

C816:               65 ***************************************
C816:               66 * GETLC - Gets language card state in Y
C816:               67 ***************************************
C816:        C816   68 getlc     equ   *
C816:A0 81          69           ldy   #$81
C818:2C 12 C0       70           bit   rdlcram       ;Language card enabled?
C81B:10 0C   C829   71           bpl   glcdone
C81D:A0 8B          72           ldy   #$8B
C81F:2C 11 C0       73           bit   rdlcbnk2      ;Bank 2?
C822:10 02   C826   74           bpl   glcbnk1
C824:A0 83          75           ldy   #$83          ;Bank 1!
C826:8D 81 C0       76 glcbnk1   sta   romin
C829:60             77 glcdone   rts



C82A:               79 * Diagnostic routine tables
C82A:        C82A   80 setv      equ   *
C82A:53 43 2B 29    81 ntbl      dfb   83,67,43,41,7
C82F:00 89 03 05    82 swtbl0    dfb   $00,$89,$03,$05,$09,$01,$7F,$5F
C837:00 83 51 53    83           dfb   $00,$83,$51,$53,$55,$57,$0F,$0D,$00,$80
C841:00 81 04 06    84 swtbl1    dfb   $00,$81,$04,$06,$0A,$02,$7F,$60
C849:00 84 52 54    85           dfb   $00,$84,$52,$54,$56,$58,$10,$0E,$00,$7F
C853:00 11 13 14    86 rswtbl    dfb   $00,$11,$13,$14,$16,$18,$FF,$7F
C85B:00 12 1A 1B    87           dfb   $00,$12,$1A,$1B,$1C,$1D,$1E,$1F,$00,$7E,$00
C866:               88           MSB   ON
C866:D2 C1 CD A0    89 rmess     asc   "RAM          ZP"
C86C:CD CD D5 C9    90 smess     asc   "MMUIOUGLU"


C875:D3 F9 F3 F4    92 success   asc   "System       OK"
C87E:        0002   69           ds    $C880-*,0     ;Protocol converter
C880:        0780   70           ds    $D000-*,0
D000:               71           include command     ;Serial port command processor
```

```
30 COMMAND        Command processor for serial & comm 20-OCT-86  06:41 PAGE 119

D000:                2 ************************************************
D000:                3 * The command routine now supports 5 new 2-character commands.  These
D000:                4 * commands enable or disable a feature of the serial port and are
D000:                5 * derived from their equivalent in the super serial card for the //.
D000:                6 *
D000:                7 * The new commands are as follows:
D000:                8 *      L - send LF out after CR
D000:                9 *      X - detect XOFF, and wait for XON
D000:               10 *      F - accept keyboard input
D000:               11 *      M - ignore LF in after CR
D000:               12 *      C - auto CR when column count > printer width
D000:               13 *
D000:               14 * Usage of location $779 (port 1) and $77A (port 2) are as follows:
D000:               15 *      bit 7 - echo output to screen if on
D000:               16 *      bit 6 - generate LF after CR if on
D000:               17 *      bit 5 - accept XOFF if on
D000:               18 *      bit 4 - ignore keyboard input if on
D000:               19 *      bit 3 - accept LF in after CR if on
D000:               20 *      bit 2 - a character was received through the ACIA and is in
D000:               21 *              location $5FE (port 1) or $67E (port 2) if on
D000:               22 *      bit 1 - XOFF is accepted, awaiting XON if on
D000:               23 *      bit 0 - signifies comm port if on, printer port if off
D000:               24 ************************************************

D000:       000D    26 charCR    equ   $0D
D000:       0000    27 ucspace   equ   $00          ;need an "upper case" space character

D000:48             29 command   pha                 ;shove character on stack
D001:3C B8 03       30           bit   sermode,x     ;Already in command?
D004:30 1C  D022    31           bmi   incmd         ;If so, go do it
D006:BC 38 06       32           ldy   eschar,x      ;If eschar = 0 ignore commands
D009:F0 14  D01F    33           beq   nocmd
D00B:5D 38 06       34           eor   eschar,x      ;Is it the command char?
D00E:0A             35           asl   A             ;Ignore high bit
D00F:D0 0E  D01F    36           bne   nocmd         ;char not command char
D011:AC FB 07       37 command1  ldy   cursor        ;Save the cursor
D014:8C 79 06       38           sty   oldcur
D017:A0 BF          39           ldy   #cmdcur       ;Set command cursor
D019:8C FB 07       40           sty   cursor
D01C:4C B5 D0       41           jmp   cominit1      ;initiate command mode

D01F:38             43 nocmd     sec                 ;Mark char not handled
D020:68             44 nocmd2    pla                 ;Restore original char
D021:60             45           rts

D022:       D022    47 incmd     equ   *             ;Command mode
D022:BC 42 C1       48           ldy   devno2,x      ;Get index for ACIA
D025:29 5F          49           and   #$5F          ;no hi-bit and upshift lower case
D027:48             50           pha                 ;save character
D028:BD B8 03       51           lda   sermode,x     ;need to see if in 2-chr command
D02B:89 08          52           bit   #$08          ;bit 3 set if so
D02D:D0 03  D032    53           bne   incmd2        ;branch if so
D02F:68             54           pla                 ;pull char back, not in 2-chr cmd
D030:80 52  D084    55           bra   incmd1        ;go on with regular command mode

D032:       D032    57 incmd2    equ   *             ;handle 2nd chr of 2-chr commands
D032:68             58           pla                 ;pull char off stack
D033:48             59           pha                 ; & reshove it to keep stack neat
```

```
30 COMMAND        Command processor for serial & comm 20-OCT-86  06:41 PAGE 120

D034:C9 00          60           cmp   #ucspace      ;is it a space? (uppercased)
D036:D0 04  D03C    61           bne   incmd3        ;no, go on with 2-chr cmd handling
D038:18             62           clc                 ;yes, ignore spaces between characters
D039:               63 ;                              of 2-chr commands
D039:68             64           pla                 ;pull uppercased char off stack
D03A:80 E4  D020    65           bra   nocmd2        ;ie mark them "handled" and don't
D03C:               66 ;                              do anything else

D03C:BD B8 03       68 incmd3    lda   sermode,x     ;get sermode back
D03F:48             69           pha                 ;save sermode for a minit
D040:29 07          70           and   #7            ;throw out all but bits 0-2
D042:8D F8 06       71           sta   temp          ;save - this is index of which cmd it is
D045:68             72           pla                 ;get sermode back
D046:29 F0          73           and   #$F0          ;now clear bits 0-3
D048:9D B8 03       74           sta   sermode,x     ;since we're done with them now
D04B:68             75           pla                 ;get character back
D04C:DA             76           phx                 ;shove x (Cn) on stack
D04D:AE F8 06       77           ldx   temp          ;get index to command's 1st chr
D050:C9 45          78           cmp   #$45          ;is it an E?
D052:F0 71  D0C5    79           beq   enable        ;yes
D054:C9 44          80           cmp   #$44          ;no, is it a D?
D056:F0 6F  D0C7    81           beq   disable       ;yes
D058:FA             82           plx                 ;retrieve X=Cn (old X still in temp)
D059:DA             83           phx                 ;push it back to keep stack neat
D05A:DD 38 06       84           cmp   eschar,x      ;compare to the command character
D05D:08             85           php                 ;save result of comparison for a bit
D05E:AE F8 06       86           ldx   temp          ;reload X= index to cmd's first chr
D061:28             87           plp                 ;retrieve result of comparison of char
D062:               88 ;                              to command char
D062:F0 13  D077    89           beq   flagit        ;yes tis 1-chr cmd followd by nother cmd
D064:C9 0D          90           cmp   #charCR       ;is it a (guess what) CR?
D066:F0 17  D07F    91           beq   oneletter     ;yes - a 1-chr command

D068:               93 ;come here for unimplemented but legal 2-char commands

D068:       D068    95 cmd2null  equ   *
D068:FA             96           plx                 ;pull x (Cn) off stack
D069:AD 79 06       97           lda   oldcur        ;restore non-cmd-mode cursor
D06C:8D FB 07       98           sta   cursor
D06F:1E B8 03       99           asl   sermode,x     ;clear cmd-mode bit (bit 7 of sermode)
D072:5E B8 03      100           lsr   sermode,x     ;by shifting out bit 7 & shifting in a 0
D075:80 A8  D01F   101           bra   nocmd         ;return marking character not handled

D077:       D077   103 flagit    equ   *             ;come here if get eschar after LXFM or T
D077:FA            104           plx                 ;need X=Cn to set bit 0 of sermode
D078:DA            105           phx                 ;but leave Cn on stack too
D079:FE B8 03      106           inc   sermode,x     ;bit 0 was 0, but is now 1
D07C:              107 ;                              1 means new command mode
D07C:AE F8 06      108           ldx   temp          ;X= index to cmd's first chr
D07F:       D07F   109 oneletter equ   *             ;come here if 2-chr cmd turns out 1 chr
D07F:BD 25 D2      110           lda   cmd2list,x    ;get command chr
D082:80 0B  D08F   111           bra   backto1       ;treat it as if we just got it

D084:       D084   113 incmd1    equ   *             ;in command mode, not 2-chrs tho
D084:DA            114           phx                 ;Save slot
D085:A2 04         115           ldx   #4            ;check 5 possible 2-chr cmds
D087:DD 25 D2      116 cmd2loop cmp   cmd2list,x    ;is it there?
D08A:F0 71  D0FD   117           beq   cmd2found     ;yes, need to flag it for next time
```

```
D08C:CA            118           dex               ;nope
D08D:10 F8   D087  119           bpl   cmd2loop    ;try next if there is one
D08F:        D08F  120 backto1    equ   *           ;come here to check for 1-chr cmds
D08F:A2 0C         121           ldx   #12         ;Check 13 commands
D091:DD 18 D2      122 cmdloop   cmp   cmdlist,x
D094:F0 74   D10A  123           beq   cmfound     ;Right char?
D096:CA            124           dex
D097:10 F8   D091  125           bpl   cmdloop
D099:FA            126           plx               ;We didn't find it
D09A:68            127           pla
D09B:48            128           pha
D09C:29 7F         129           and   #$7F        ;if char is cntl char
D09E:C9 20         130           cmp   #$20        ;it can be the new comd char
D0A0:B0 03   D0A5  131           bcs   ckdig       ;branch if not cntl character
D0A2:9D 38 06      132 cmdz2     sta   eschar,x    ;save command char and
D0A5:              133 ;                           drop thru ckdig to cdone
D0A5:49 30         134 ckdig     eor   #$30        ;zap it down to 0n if char was a digit
D0A7:C9 0A         135           cmp   #$0A        ;is it a digit?
D0A9:B0 33   D0DE  136           bcs   cdone       ;skip if no, an unexpected intruder
D0AB:A0 0A         137           ldy   #10         ;A = A + 10 * current number
D0AD:6D 7E 07      138 digloop   adc   number      ;C=0 on first entry
D0B0:88            139           dey
D0B1:D0 FA   D0AD  140           bne   digloop
D0B3:80 0A   D0BF  141           bra   cominit     ;not starting new cmd mode, just save #

D0B5:        D0B5  143 cominit1   equ   *           ;start new cmd mode here
D0B5:BD B8 03      144           lda   sermode,x   ;get sermode
D0B8:29 C0         145           and   #$C0        ;clear bits 0-5 (starting a new cmd seq --
D0BA:9D B8 03      146           sta   sermode,x   ;they are used for misc during cmd mode)
D0BD:A9 00         147           lda   #0          ;load a 0 to stuff in NUMBER
D0BF:8D 7E 07      148 cominit   sta   number
D0C2:38            149           sec               ;Mark in command mode
D0C3:80 25   D0EA  150           bra   cmset

D0C5:        D0C5  152 enable     equ   *           ;got a 2-chr command aE
D0C5:38            153           sec               ;set carry
D0C6:90            154           dfb   $90         ;bcc to skip next byte (the CLC)
D0C7:        D0C7  155 disable    equ   *           ;got a 2-chr command aD
D0C7:18            156           clc               ;clear carry
D0C8:08            157           php               ;push P to save carry
D0C9:E0 00         158           cpx   #0          ;if X=0 then command is LE or LD
D0CB:F0 27   D0F4  159           beq   cmd21       ;so just make it act like L or K
D0CD:E0 04         160           cpx   #4          ;if X=4 then command is CE or CD
D0CF:F0 41   D112  161           beq   cmd.c       ;skip if so

D0D1:              163 ************************************************
D0D1:              164 * for other 2-chr cmds, their FLAGS masks' indexes are 2X+3
D0D1:              165 * for an E or 2X+4 for a D
D0D1:              166 ************************************************

D0D1:8A            168           txa               ;copy x to acc for arithmetic
D0D2:18            169           clc               ;clear carry for arithmetic
D0D3:0A            170           asl   A           ;multiply index by 2
D0D4:69 03         171           adc   #3          ;add 3 to get mask index
D0D6:AA            172           tax               ;put mask index in X
D0D7:28            173           plp               ;get carry back
D0D8:B0 01   D0DB  174           bcs   xready      ;carry set = Enable so X is ready
D0DA:E8            175           inx               ;cmd was Disable so inc X to next mask
```



```
D0DB:4C 39 D1      176 xready    jmp   cmdi        ;go do mask stuff to FLAGS

D0DE:              178 ;sermode bit 0 tells whether to set or clear command mode

D0DE:        D0DE  180 cdone      equ   *
D0DE:BD B8 03      181           lda   sermode,x   ;so get it
D0E1:4A            182           lsr   A           ;shift bit 0 to carry
D0E2:B0 D1   D0B5  183           bcs   cominit1    ;if set, start new cmd mode
D0E4:AD 79 06      184           lda   oldcur      ;Restore the cursor
D0E7:8D FB 07      185           sta   cursor      ;& fall through to cmset with carry clear
D0EA:08            186 cmset     php
D0EB:1E B8 03      187           asl   sermode,x   ;set command mode according to carry
D0EE:28            188           plp
D0EF:7E B8 03      189           ror   sermode,x   ;leaves carry clear
D0F2:68            190           pla               ;character handled
D0F3:60            191           rts               ;because carry clear...

D0F4:        D0F4  193 cmd21      equ   *           ;come here to handle LE & LD
D0F4:A9 4C         194           lda   #$4C        ;make LE look like L
D0F6:28            195           plp               ;get P back with carry indicating E or D
D0F7:B0 96   D08F  196           bcs   backto1     ;carry set means it was an E
D0F9:A9 4B         197           lda   #$4B        ;make LD look like K
D0FB:80 92   D08F  198           bra   backto1

D0FD:8A            200 cmd2found txa               ;copy index of cmd to acc
D0FE:FA            201           plx               ;restore X to Cn
D0FF:1D B8 03      202           ora   sermode,x   ;copy top 2 bits of sermode
D102:09 08         203           ora   #$08        ;& set bit 3 - 2-chr-command-mode flag
D104:9D B8 03      204           sta   sermode,x   ;sermode = index to 2-chr cmds issued
D107:38            205           sec               ;set carry so we stay in command mode
D108:80 E0   D0EA  206           bra   cmset       ;for next time

D10A:A9 D1         208 cmfound   lda   #<cmdcr     ;get hi byte of where to go
D10C:48            209           pha               ;save it on stack
D10D:BD F5 D1      210           lda   cmdtable,x  ;get lo byte of where to go
D110:48            211           pha               ;save it on stack
D111:60            212           rts               ;go there by RTSing

D112:28            214 cmd.c     plp               ;restore status to check carry bit
D113:FA            215           plx               ;restore slot number in x
D114:B0 05   D11B  216           bcs   cmd.c1      ;skip if enable
D116:9E B8 04      217           stz   pwdth,x     ;CD is same as PWDTH=0, no CR
D119:80 C3   D0DE  218           bra   cdone       ;we're done here

D11B:BC 86 D1      220 cmd.c1    ldy   defidx2-$C1,x ;get y index into aux screenholes
D11E:20 2A D2      221           jsr   r.getalt    ;go get it from aux
D121:9D B8 04      222           sta   pwdth,x     ;restore default PWDTH
D124:80 B8   D0DE  223           bra   cdone       ;we're done here


D126:FA            225 cmdz      plx               ;Zero escape character
D127:9E B8 04      226           stz   pwdth,x     ;And the width
D12A:A9 00         227           lda   #0
D12C:4C A2 D0      228           jmp   cmdz2


D12F:        D12F  230 cmdcr     equ   *
D12F:        D12F  231 cmdn      equ   *
```

```
D12F:7A            232          ply
D130:AD 7E 07      233          lda   number      ;Get number inputted
D133:F0 05   D13A  234          beq   cmdi2       ;skip if 0
D135:99 B8 04      235          sta   pwdth,y     ;Update printer width
D138:F0            236          dfb   $F0         ;BEQ opcode to skip next byte (the PLY)
D139:        D139  237 cmdi     equ   *
D139:        D139  238 cmdk     equ   *
D139:        D139  239 cmdl     equ   *
D139:7A            240          ply
D13A:B9 B8 06      241 cmdi2    lda   flags,y
D13D:3D 02 D2      242          and   mask1,x     ;Mask off bit we'll change
D140:1D 0D D2      243          ora   mask2,x     ;Change it
D143:99 B8 06      244          sta   flags,y     ;Back it goes
D146:98            245          tya               ;Put slot back in x
D147:AA            246          tax               ;(via acc)
D148:4C DE D0      247 cdone2   jmp   cdone       ;Good bye


D14B:88            249 cmdp     dey               ;Make y point to command reg
D14C:A9 1F         250 cmdd     lda   #$1F        ;Mask off high three bits
D14E:38            251          sec               ;C=1 means high 3 bits
D14F:90            252          dfb   $90         ;BCC opcode to skip next byte
D150:A9 F0         253 cmdb     lda   #$F0        ;Mask off lower 4 bits F0 = BNE
D152:18            254          clc               ;F0 will skip this if cmdp or cmdd
D153:39 FB BF      255          and   scntl,y     ;Mask off bits being changed
D156:8D F8 06      256          sta   temp        ;Save it
D159:FA            257          plx
D15A:AD 7E 07      258          lda   number      ;Get inputed number
D15D:29 0F         259          and   #$0F        ;Only lower nibble valid
D15F:90 05   D166  260          bcc   noshift     ;If C=1 shift to upper 3 bits
D161:0A            261          asl   A
D162:0A            262          asl   A
D163:0A            263          asl   A
D164:0A            264          asl   A
D165:0A            265          asl   A
D166:0D F8 06      266 noshift  ora   temp        ;Get the rest of the bits
D169:C8            267          iny               ;Put them in the ACIA
D16A:80 17   D183  268          bra   cmdp2       ;increment puts em away where they go.


D16C:B9 FA BF      270 cmds     lda   scomd,y     ;Transmit a break
D16F:48            271          pha               ;Save current ACIA state
D170:09 0C         272          ora   #$0C        ;Do the break
D172:99 FA BF      273          sta   scomd,y
D175:A9 E9         274          lda   #233        ;For 233 ms
D177:A2 53         275 mswait   ldx   #83         ;Wait 1 ms
D179:48            276 msloop   pha               ;((12*82)+11)+2+3=1000us
D17A:68            277          pla
D17B:CA            278          dex
D17C:D0 FB   D179  279          bne   msloop
D17E:3A            280          dec   a
D17F:D0 F6   D177  281          bne   mswait
D181:68            282          pla
D182:FA            283          plx
D183:        D183  284 cmdp2    equ   *
D183:99 FA BF      285          sta   scomd,y
D186:80 C0   D148  286          bra   cdone2
```



```
D188:        D188  288 cmdr     equ   *
D188:99 F9 BF      289          sta   sstat,y     ;Reset the ACIA
D18B:AD 7B 06      290          lda   vfactv      ;Check if video firmware active
D18E:0A            291          asl   A           ;Save it in C
D18F:20 97 C7      292          jsr   swsthk2     ;assume video firmware active
D192:90 03   D197  293          bcc   cmdq        ;branch if good guesser...
D194:20 9D C7      294          jsr   swzzqt2     ;Reset the hooks
D197:18            295 cmdq     clc               ;Quit terminal mode
D198:B0            296          dfb   $B0         ;BCS to skip next byte
D199:38            297 cmdt     sec               ;Into terminal mode
D19A:FA            298          plx               ;Recover X
D19B:20 A0 D1      299          jsr   setterm
D19E:80 A8   D148  300          bra   cdone2


D1A0:        D1A0  302 setterm  equ   *           ;set/clear terminal mode
D1A0:BD B8 03      303          lda   sermode,x   ;Get terminal mode status
D1A3:89 40         304          bit   #$40        ;Z=1 if not in terminal mode
D1A5:90 12   D1B9  305          bcc   stclr       ;Branch if clearing terminal mode
D1A7:D0 20   D1C9  306          bne   stwasok     ;Was already set
D1A9:E4 39         307          cpx   kswh        ;Are we in the input hooks
D1AB:D0 47   D1F4  308          bne   strts       ;Leaves C=1 if =
D1AD:09 40         309          ora   #$40        ;Set term mode bit
D1AF:AC 79 06      310          ldy   oldcur      ;Save what was in oldcur
D1B2:8C 7A 06      311          sty   oldcur2
D1B5:A0 DF         312          ldy   #termcur    ;Get new cursor value
D1B7:80 07   D1C0  313          bra   stset
D1B9:F0 0E   D1C9  314 stclr    beq   stwasok     ;Branch if already clear
D1BB:29 BF         315          and   #$BF        ;Clear the bit
D1BD:AC 7A 06      316          ldy   oldcur2     ;Restore the cursor
D1C0:9D B8 03      317 stset    sta   sermode,x
D1C3:8C 79 06      318          sty   oldcur      ;Save cursor to be restored after command
D1C6:8C FB 07      319          sty   cursor
D1C9:BC 42 C1      320 stwasok  ldy   devno2,x
D1CC:58            321          cli               ;want to leave with interrupts active
D1CD:08            322          php
D1CE:78            323          sei               ;but off while we twittle bits
D1CF:B9 FA BF      324          lda   scomd,y
D1D2:09 02         325          ora   #$2         ;disable receiver interrupts if
D1D4:90 02   D1D8  326          bcc   cmdt2       ; not in terminal mode
D1D6:29 FD         327          and   #$FD        ;enable when in terminal mode
D1D8:        D1D8  328 cmdt2    equ   *
D1D8:99 FA BF      329          sta   scomd,y
D1DB:A9 00         330          lda   #0
D1DD:6A            331          ror   a           ;set kbd interrupts according to t-mode
D1DE:8D FA 05      332          sta   typhed
D1E1:10 07   D1EA  333          bpl   cmdt3       ;branch if leaving terminal mode
D1E3:9C 7C 05      334          stz   twser       ; and ser buf...
D1E6:9C 7C 06      335          stz   trser
D1E9:8A            336          txa               ;use x to enable serial buffering
D1EA:8D FC 04      337 cmdt3    sta   aciabuf
D1ED:28            338          plp               ;restore carry, enable interrupts.
D1EE:8E FC 05      339 flush    stx   twkey       ;Flush the type ahead buffer
D1F1:8E FF 06      340          stx   trkey
D1F4:60            341 strts    rts


D1F5:              343          MSB   OFF
```

```
D1F5:       D1F5  344 cmdtable  equ   *           ;command routines' lo bytes
D1F5:38           345           dfb   >cmdi-1
D1F6:38           346           dfb   >cmdk-1
D1F7:38           347           dfb   >cmdl-1
D1F8:2E           348           dfb   >cmdn-1
D1F9:2E           349           dfb   >cmdcr-1
D1FA:4F           350           dfb   >cmdb-1
D1FB:4B           351           dfb   >cmdd-1
D1FC:4A           352           dfb   >cmdp-1
D1FD:96           353           dfb   >cmdq-1
D1FE:87           354           dfb   >cmdr-1
D1FF:6B           355           dfb   >cmds-1
D200:98           356           dfb   >cmdt-1
D201:25           357           dfb   >cmdz-1

D202:             359 * masks for:    I   K   L   N   CR  XE  XD  FE  FD  ME  MD
D202:7F BF BF 7F  360 mask1     dfb   $7F,$BF,$BF,$7F,$FF,$DF,$DF,$EF,$EF,$F7,$F7
D20D:80 00 40 00  361 mask2     dfb   $80,$00,$40,$00,$00,$20,$00,$00,$10,$00,$08

D218:       D218  363 cmdlist   equ   *
D218:49 4B 4C 4E  364           asc   "IKLN"
D21C:0D           365           dfb   $0D           ;cr (part of cmdlist)
D21D:42 44 50 51  366           asc   "BDPQRSTZ"
D225:       D225  367 cmd2list  equ   *
D225:4C 58 46 4D  368           asc   "LXFMC"       ;2-chr commands' first chrs

D22A:             370 ************************************************
D22A:             371 * R.GETALT is the same as GETALT in main rom.  Only the
D22A:             372 * location is different.
D22A:             373 ************************************************

D22A:AD 13 C0     375 r.getalt  lda   rdramrd       ;save state of aux memory
D22D:0A           376           asl
D22E:AD 18 C0     377           lda   rd80col       ;and the 80STORE switch
D231:08           378           php
D232:8D 00 C0     379           sta   clr80col      ;no 80STORE to get page 1
D235:8D 03 C0     380           sta   rdcardram     ;pop in the other half of RAM
D238:B9 78 04     381           lda   $478,y        ;read the desired byte
D23B:28           382           plp                 ;and restore memory
D23C:B0 03   D241 383           bcs   r.getalt1
D23E:8D 02 C0     384           sta   rdmainram
D241:10 03   D246 385 r.getalt1 bpl   r.getalt2
D243:8D 01 C0     386           sta   set80col
D246:60           387 r.getalt2 rts

D247:03 07        389 defidx2   dfb   3,7            ;same as DEFIDX in main rom.
D249:              72           include mbasic       ;Mouse BASIC routines @ 2:C100
D249:       01B7    1           ds    $D400-*,0
```



```
D400:               3 ************************************************
D400:               4 * BASICIN - input from basic
D400:               5 *
D400:               6 * creates  +XXXXX,+YYYYY,+SS
D400:               7 * XXXXX = x position, YYYYY = y position, SS = status
D400:               8 *       - = key pressed
D400:               9 *       1 = button pressed
D400:              10 *       2 = button just pressed
D400:              11 *       3 = button just released
D400:              12 *       4 = button not pressed
D400:              13 ************************************************

D400:91 28         15 basicin   sta   (basl),y     ;fix flashing char
D402:A9 05         16           lda   #>inent      ;fix input entry
D404:85 38         17           sta   kswl
D406:AD 00 C0      18           lda   kbd          ;test the keyboard
D409:0A            19           asl   A
D40A:08            20           php                ;save kbd and int stat for later
D40B:78            21           sei                ;no interrupts while getting position
D40C:20 79 D6      22           jsr   x.mread
D40F:A0 05         23           ldy   #5           ;move x position into the buffer
D411:AE 7F 05      24           ldx   mouxh
D414:AD 7F 04      25           lda   mouxl
D417:20 41 D4      26           jsr   hextodec     ;convert it
D41A:A0 0C         27           ldy   #12
D41C:AE FF 05      28           ldx   mouyh
D41F:AD FF 04      29           lda   mouyl
D422:20 41 D4      30           jsr   hextodec
D425:AD 7F 07      31           lda   moustat
D428:2A            32           rol   A
D429:2A            33           rol   A
D42A:2A            34           rol   A
D42B:29 03         35           and   #3
D42D:49 03         36           eor   #3
D42F:1A            37           inc   A
D430:28            38           plp                ;restore int & kbd status
D431:A0 10         39           ldy   #16
D433:20 52 D4      40           jsr   hexdec2      ;x=0 from last div10
D436:7A            41           ply
D437:A2 11         42           ldx   #17          ;x = eol
D439:A9 8D         43           lda   #$8D         ;cr
D43B:9D 00 02      44 putinbuf  sta   inbuf,x
D43E:4C 84 C7      45           jmp   swrts2       ;goback

D441:              47 ************************************************
D441:              48 * HEXTODEC - puts +0000, into the input buffer
D441:              49 * inputs: a = low byte of number
D441:              50 *         x = high byte of number
D441:              51 *         y = position of ones digit
D441:              52 ************************************************

D441:E0 80         54 hextodec  cpx   #$80         ;is it a negative number?
D443:90 0D   D452  55           bcc   hexdec2
D445:49 FF         56           eor   #$FF         ;form two's complement
D447:69 00         57           adc   #0           ;c = 1 from compare
D449:48            58           pha                ;save it
D44A:8A            59           txa
D44B:49 FF         60           eor   #$FF
```

```
D44D:69 00          61           adc   #0
D44F:AA             62           tax
D450:68             63           pla
D451:38             64           sec
D452:8D 14 02       65 hexdec2   sta   binl          ;store the number to convert
D455:8E 15 02       66           stx   binh
D458:A9 2B          67           lda   #'+'          ;store the sign in the buffer
D45A:90 02   D45E   68           bcc   hdpos2
D45C:A9 2D          69           lda   #'-'
D45E:48             70 hdpos2    pha                 ;save the sign
D45F:A9 2C          71           lda   #','          ;store a comma after the number
D461:99 01 02       72           sta   inbuf+1,y
D464:        D464   73 hdloop    equ   *             ;divide by 10

D464:               75 * divide BINH,L by 10 and leave remainder in a

D464:A2 11          77           ldx   #16+1         ;16 bits and first time do nothing
D466:A9 00          78           lda   #0
D468:18             79           clc                 ;c=0 so first ROL leaves a=0
D469:2A             80 dv10loop  rol   A
D46A:C9 0A          81           cmp   #10           ;a >= 10?
D46C:90 02   D470   82           bcc   dv10lt        ;branch if <
D46E:E9 0A          83           sbc   #10           ;c = 1 from compare and is left set
D470:2E 14 02       84 dv10lt    rol   binl
D473:2E 15 02       85           rol   binh
D476:CA             86           dex
D477:D0 F0   D469   87           bne   dv10loop
D479:09 30          88           ora   #'0'          ;make a ascii char
D47B:99 00 02       89           sta   inbuf,y
D47E:88             90           dey
D47F:F0 08   D489   91           beq   hddone        ;stop on 0,6,12
D481:C0 07          92           cpy   #7
D483:F0 04   D489   93           beq   hddone
D485:C0 0E          94           cpy   #14
D487:D0 DB   D464   95           bne   hdloop
D489:68             96 hddone    pla                 ;get the sign
D48A:99 00 02       97           sta   inbuf,y
D48D:60             98           rts
D48E:               73           include banger
```



```
D48E:                3 * These routines test all 128K ram.  All combinations of soft
D48E:                4 * switches applicable to the //c are tested and verified.
D48E:                5 *
D48E:                6 * In the event of any failure, the diagnostic is halted.  A message
D48E:                7 * is written to screen memory indicating the source of the failure.
D48E:                8 * When RAM fails the message is composed of "RAM ZP" (indicating
D48E:                9 * failure detected in the first page of RAM) or "RAM" (meaning the
D48E:               10 * other 63.75K), followed by a binary representation of the failing
D48E:               11 * bits set to "1".  For example, "RAM  0 1 1 0 0 0 0 0" indicates
D48E:               12 * that bits 5 and 6 were detected as failing.  To represent
D48E:               13 * auxillary memory, a "*" symbol is printed preceeding the message.
D48E:               14 *
D48E:               15 * When the MMU or IOU fail, the message is simply "MMU" or "IOU".
D48E:               16 * If the IOUDIS or DHIRES switch fails, the message is "GLU".
D48E:               17 *
D48E:               18 * The test will run continuously for as long as the Open and Closed
D48E:               19 * Apple keys remain depressed (or no keyboard is connected) and no
D48E:               20 * failures are encountered.  The message "System OK" will appear in
D48E:               21 * the middle of the screen when a successful cycle has been run and
D48E:               22 * either of the Apple keys are no longer depressed.  Another cycle
D48E:               23 * may be initiated by pressing both Apple keys again while this
D48E:               24 * message is on the screen.  To exit diagnostics, Control-Reset
D48E:               25 * must be pressed without the Apple keys depressed.
D48E:               26 *
D48E:               27 *
D48E:        0011   28 GLUIDX    EQU   $11
D48E:        0009   29 IOUIDX    EQU   $09
D48E:        0001   30 MMUIDX    EQU   $01
D48E:        05B8   31 SCREEN    EQU   $5B8
D48E:               32 *
D48E:8D 50 C0       33 DIAGS     sta   txtclr        ;text mode off
D491:8D 78 C0       34           sta   ioudsbl       ;Disable IOU
D494:8D 5F C0       35           sta   setan3        ;Double hires off

D497:               37 * Test Zero-Page, then all of memory.  Report errors when
D497:               38 * encountered.  Accumulator can be anything on entry.  All
D497:               39 * registers used, but no stack.  Addresses between $C000
D497:               40 * and $CFFF are mapped to main $D000 bank.  Addresses
D497:               41 * between $C000 and $CFFF are mapped to main $D000 bank.

D497:A0 04          43 TSTZPG    ldy   #$4
D499:A2 00          44           ldx   #0
D49B:18             45 zp1       clc                 ;fill zero page with a pattern
D49C:79 2A C8       46           adc   ntbl,y
D49F:95 00          47           sta   $00,x
D4A1:E8             48           inx
D4A2:D0 F7   D49B   49           bne   zp1           ;after all bytes filled,
D4A4:18             50 zp2       clc                 ; ACC has original value again.
D4A5:79 2A C8       51           adc   ntbl,y        ;so values can be tested
D4A8:D5 00          52           cmp   $00,x
D4AA:D0 10   D4BC   53           bne   ZPERROR       ;branch if memory failed
D4AC:E8             54           inx
D4AD:D0 F5   D4A4   55           bne   zp2           ;loop until all 256 bytes tested
D4AF:6A             56           ror   a             ;change ACC so location $FF will change
D4B0:2C 19 C0       57           bit   rdvblbar      ; use RDVBL for a little randomness...
D4B3:10 02   D4B7   58           bpl   zp3
D4B5:49 A5          59           eor   #$A5
D4B7:88             60 zp3       dey                 ;use a different pattern now
```

```
D4B8:10 E1   D49B  61          bpl   zp1          ;branch to retest with other value
D4BA:30 06   D4C2  62          bmi   TSTMEM2      ;branch always


D4BC:55 00         64 ZPERROR  eor   $00,x        ;which bits are bad?
D4BE:18            65          clc                ;indicate zero page failure
D4BF:4C 73 C4      66          jmp   BADBITS
D4C2:4C C6 C3      67 TSTMEM2  JMP   TSTMEM       ;Off to the rest of it



D4C5:       D4C5   69 zznm     equ   *
D4C5:20 9D C7      70          jsr   swzzqt2      ;Get out of the hooks
D4C8:68            71          pla                ;Get junk off of stack
D4C9:7A            72          ply
D4CA:68            73          pla
D4CB:A9 FF         74          lda   #$FF
D4CD:AA            75          tax
D4CE:E8            76 zzloop   inx
D4CF:5D DA D4      77          eor   qtbl,x
D4D2:9D 00 02      78          sta   inbuf,x
D4D5:10 F7   D4CE  79          bpl   zzloop
D4D7:4C 84 C7      80          jmp   swrts2

D4DA:AD 3B 0A 0B   82 qtbl     dfb   $AD,$3B,$0A,$0B,$48,$77,$3E,$05
D4E2:00 05 08 0C   83          dfb   $00,$05,$08,$0C,$1E,$53,$65,$37
D4EA:1C 07 0C 45   84          dfb   $1C,$07,$0C,$45,$62,$27,$00,$17
D4F2:1C 07 07 05   85          dfb   $1C,$07,$07,$05,$4B,$6D,$24,$02
D4FA:0E 45 61 32   86          dfb   $0E,$45,$61,$32,$18,$02,$07,$1D
D502:53 6A 2B 0C   87          dfb   $53,$6A,$2B,$0C,$08,$16,$53,$68
D50A:3D 06 07 1B   88          dfb   $3D,$06,$07,$1B,$01,$E3
D510:              74          include mousein7.x
```



```
D510:       00F0    2          ds    $D600-*,$00
D600:                3 ************************************************
D600:                4 * Initmouse - resets the mouse
D600:                5 *  Also clears all of the mouse holes
D600:                6 * note that iou access fires pdlstrb & makes mouse happy
D600:                7 ************************************************

D600:9C 7F 07        9 i.nitmouse stz  moustat      ;Clear status
D603:A2 80          10          ldx   #$80
D605:A0 01          11          ldy   #1
D607:9E 7D 04       12 xrloop   stz   minxl,x      ;Minimum = $0000
D60A:9E 7D 05       13          stz   minxh,x
D60D:A9 FF          14          lda   #$FF         ;Maximum = $03FF
D60F:9D 7D 06       15          sta   maxxl,x
D612:A9 03          16          lda   #03
D614:9D 7D 07       17          sta   maxxh,x
D617:A2 00          18          ldx   #0
D619:88             19          dey
D61A:10 EB   D607   20          bpl   xrloop
D61C:20 51 D6       21          jsr   x.mhome      ;Clear the mouse holes
D61F:A9 00          22          lda   #0           ;Fall into SETMOU

D621:               24 ************************************************
D621:               25 * XSETMOU - Sets the mouse mode to A
D621:               26 ************************************************

D621:AA             28 x.setmou tax
D622:20 46 C7       29          jsr   m..oveirq    ;Make sure interrupt vector is right
D625:8A             30          txa                ;Only x preserved by moveirq
D626:8D 78 04       31          sta   moutemp
D629:4A             32          lsr   A            ;D0 = 1 if mouse active
D62A:0D 78 04       33          ora   moutemp      ;D2 = 1 if vbl active
D62D:C9 10          34          cmp   #$10         ;If >=$10 then invalid mode
D62F:B0 1F   D650   35          bcs   sminvalid
D631:29 05          36          and   #5           ;Extract VBL & Mouse
D633:F0 01   D636   37          beq   xsoff        ;Turning it off?
D635:58             38          cli                ;If not, ints active
D636:69 55          39 xsoff    adc   #$55         ;Make iou byte C=0

D638:               41 ************************************************
D638:               42 *
D638:               43 * SETIOU - Sets the IOU interrupt modes to A
D638:               44 *  Inputs: A = Bits to change
D638:               45 *  D7 = Y int on falling edge
D638:               46 *  D6 = Y int on rising edge
D638:               47 *  D5 = X int on falling edge
D638:               48 *  D4 = X int on rising edge
D638:               49 *  D3 = Enable VBL int
D638:               50 *  D2 = Disable VBL int
D638:               51 *  D1 = Enable mouse int
D638:               52 *  D0 = Disable mouse int
D638:               53 *
D638:               54 ************************************************

D638:08             56 setiou   php
D639:78             57          sei                ;Don't allow ints while iou enabled
D63A:8E FF 07       58          stx   moumode
D63D:8D 79 C0       59          sta   iouenbl      ;Enable iou access
```

```
D640:A2 08          60          ldx  #8
D642:CA             61 siloop   dex
D643:0A             62          asl  A          ;Get a bit to check
D644:90 03   D649   63          bcc  sinoch     ;No change if C=0
D646:9D 58 C0       64          sta  iou,x      ;Set it
D649:D0 F7   D642   65 sinoch   bne  siloop     ;Any bits left in A?
D64B:8D 78 C0       66          sta  ioudsbl    ;Turn off iou access
D64E:28             67          plp
D64F:18             68          clc
D650:60             69 sminvalid rts

D651:               71 ************************************************
D651:               72 * XMHOME- Clears mouse position & status
D651:               73 ************************************************

D651:A2 80          75 x.mhome  ldx  #$80       ;Point mouse to upper left
D653:80 02   D657   76          bra  xmh2
D655:A2 00          77 xmhloop  ldx  #0
D657:BD 7D 04       78 xmh2     lda  minxl,x
D65A:9D 7F 04       79          sta  mouxl,x
D65D:BD 7D 05       80          lda  minxh,x
D660:9D 7F 05       81          sta  mouxh,x
D663:CA             82          dex
D664:10 EF   D655   83          bpl  xmhloop
D666:80 0C   D674   84          bra  xmcdone

D668:               86 ************************************************
D668:               87 * XMCLEAR - Sets the mouse to 0,0
D668:               88 ************************************************

D668:9C 7F 04       90 x.mclear stz  mouxl
D66B:9C 7F 05       91          stz  mouxh
D66E:9C FF 04       92          stz  mouyl
D671:9C FF 05       93          stz  mouyh
D674:9C 7F 06       94 xmcdone  stz  mouarm
D677:18             95          clc
D678:60             96          rts

D679:               98 ************************************************
D679:               99 * XMREAD - Updates the screen holes
D679:              100 ************************************************

D679:A9 20         102 x.mread  lda  #movarm    ;Has mouse moved?
D67B:1C 7F 07      103          trb  moustat    ;Clear moved bit in stat
D67E:2D 7F 06      104          and  mouarm
D681:1C 7F 06      105          trb  mouarm     ;Clear arm bit
D684:2C FF 07      106          bit  moumode    ;If D7 = 1 leave buttons alone
D687:30 13   D69C  107          bmi  xmrd2
D689:2C 63 C0      108          bit  moubut     ;Button pressed?
D68C:30 02   D690  109          bmi  xrbut
D68E:09 80         110          ora  #$80
D690:2C 7F 07      111 xrbut    bit  moustat    ;Pressed last time?
D693:10 02   D697  112          bpl  xrbut2
D695:09 40         113          ora  #$40
D697:8D 7F 07      114 xrbut2   sta  moustat
D69A:18            115          clc
D69B:60            116          rts
D69C:       D69C   117 xmrd2    equ  *          ;Leave button bits alone
```



```
D69C:0D 7F 07      118          ora  moustat
D69F:29 E0         119          and  #$E0       ;Button bits
D6A1:80 F4   D697  120          bra  xrbut2

D6A3:              122 ************************************************
D6A3:              123 * XMCLAMP - Store new bounds
D6A3:              124 *  Inputs A = 1 for Y, 0 for X axis
D6A3:              125 *    minl, minh, maxl, maxh = new bounds
D6A3:              126 ************************************************

D6A3:6A            128 x.mclamp ror  A          ;1 -> 80
D6A4:6A            129          ror  A
D6A5:29 80         130          and  #$80
D6A7:AA            131          tax
D6A8:AD 78 04      132          lda  minl
D6AB:9D 7D 04      133          sta  minxl,x
D6AE:AD 78 05      134          lda  minh
D6B1:9D 7D 05      135          sta  minxh,x
D6B4:AD F8 04      136          lda  maxl
D6B7:9D 7D 06      137          sta  maxxl,x
D6BA:AD F8 05      138          lda  maxh
D6BD:9D 7D 07      139          sta  maxxh,x
D6C0:18            140          clc             ;No error
D6C1:60            141          rts

D6C2:              143 ************************************************
D6C2:              144 * XMTSTINT - Checks mouse status bits
D6C2:              145 * Used for user mouse interrupt
D6C2:              146 ************************************************

D6C2:48            148 x.mtstint pha
D6C3:18            149          clc
D6C4:A9 0E         150          lda  #$0E
D6C6:2D 7F 07      151          and  moustat
D6C9:D0 01   D6CC  152          bne  nostat2
D6CB:38            153          sec
D6CC:68            154 nostat2  pla
D6CD:60            155          rts

D6CE:       0032   157          ds   $D700-*,$00
D700:               75          include s.execute
D700:       0100    1          ds   $D800-*,$00
```

```
D800:                    3 ************************************************
D800:                    4 * execution routines.  these routines must begin in the same page
D800:                    5 ************************************************

D800:                    7 ************************************************
D800:                    8 * EXECUTE - does command in command
D800:                    9 *           inputs: a = command
D800:                   10 ************************************************

D800:85 42               12 execute   sta   cmmand
D802:A0 C4               13           ldy   #4+$C0
D804:8C F8 07            14           sty   sl.mslot
D807:A2 C8               15           ldx   #4*$10+$88
D809:8E 78 07            16           stx   sl.devno      ;save command and hardware index
D80C:A9 00               17           lda   #0            ;clear error flag
D80E:8D F8 04            18           sta   error
D811:20 D3 D9            19           jsr   sl.format     ;do we need to format?
D814:A4 42               20           ldy   cmmand        ;get command
D816:B9 80 C6            21           lda   parmtbl,y     ;check parameter count
D819:30 04   D81F        22           bmi   exec2         ;if negative, no parm check
D81B:C5 43               23           cmp   pparm
D81D:D0 15   D834        24           bne   pzcnt
D81F:A9 D8               25 exec2     lda   #<xstatus     ;all entry points on same page
D821:80 01   D824        26           bra   exec3         ;skip around the basic patch

D823:        0001        28           ds    $d824-*,$00   ;break handler will correct for this

D824:48                  30 exec3     pha
D825:B9 9A C6            31           lda   cmdtbl,y
D828:48                  32           pha
D829:AC F8 07            33           ldy   sl.mslot
D82C:AE 78 07            34           ldx   sl.devno
D82F:60                  35           rts

D830:A9 01               37 pzcmd     lda   #badcmd       ;invalid command
D832:D0 02   D836        38           bne   pzcnt2
D834:A9 04               39 pzcnt     lda   #badpcnt
D836:8D F8 04            40 pzcnt2    sta   error
D839:60                  41 iorts     rts

D83A:4C 80 C5            43 pread2.z  jmp   sl.pread       ;entry point in this page
D83D:4C F7 C5            44 pwrite2   jmp   sl.pwrite      ;entry point in this page
D840:4C 4F D9            45 dosconv2  jmp   dosconv        ;entry point in this page
D843:4C 30 DB            46 xdiag     jmp   xdiagz         ;entry point in this page

D846:                    48 ************************************************
D846:                    49 * XSTATUS - ProDOS status call
D846:                    50 ************************************************

D846:B9 B8 03            52 xstatus   lda   numbanks,y     ;size = # 64K banks / 2
D849:4A                  53           lsr   A
D84A:8D F8 05            54           sta   yval
D84D:A9 00               55           lda   #0
D84F:8D 78 05            56           sta   xval
D852:60                  57           rts

D853:                    59 ************************************************
D853:                    60 * PSTAT0 - Status call for device 0
```



```
D853:                    61 *          returns status block for call 0
D853:                    62 *          1 0 0 0 0 0 0 0
D853:                    63 ************************************************

D853:A5 47               65 pstat0    lda   pstat         ;must be call 0
D855:D0 17   D86E        66           bne   stbad         ;branch if bad
D857:8D F8 05            67           sta   yval          ;set bytes read count
D85A:A0 08               68           ldy   #8
D85C:8C 78 05            69           sty   xval
D85F:88                  70           dey
D860:91 45               71 st0lp     sta   (pbuff),y     ;save out the 0s
D862:88                  72           dey
D863:D0 FB   D860        73           bne   st0lp
D865:A9 01               74           lda   #1
D867:91 45               75           sta   (pbuff),y
D869:60                  76           rts

D86A:                    78 ************************************************
D86A:                    79 * PCNTL - control call
D86A:                    80 *         call 0 (reset) is implemented for both devices
D86A:                    81 ************************************************

D86A:A5 47               83 pcntl     lda   pstat         ;call 0?
D86C:F0 05   D873        84           beq   pcntok
D86E:A9 21               85 stbad     lda   #badctl       ;oops! bad status/control number
D870:8D F8 04            86           sta   error
D873:60                  87 pcntok    rts

D874:                    89 ************************************************
D874:                    90 * PSTATUS - status call for device 1
D874:                    91 *           call 0,3 supported
D874:                    92 ************************************************

D874:A9 04               94 sl.pstatus lda  #4            ;number bytes for call 0
D876:A6 47               95           ldx   pstat
D878:F0 06   D880        96           beq   pst0
D87A:E0 03               97           cpx   #3            ;is it #3?
D87C:D0 F0   D86E        98           bne   stbad         ;branch if bad call
D87E:A9 19               99           lda   #25           ;# bytes for call 3
D880:8D 78 05           100 pst0      sta   xval
D883:A2 00              101           ldx   #0
D885:8E F8 05           102           stx   yval
D888:A8                 103           tay
D889:88                 104           dey
D88A:B9 67 C6           105 pstmov    lda   stattbl,y     ;move the status info
D88D:91 45              106           sta   (pbuff),y
D88F:88                 107           dey
D890:10 F8   D88A       108           bpl   pstmov
D892:AC F8 07           109           ldy   sl.mslot      ;get the size
D895:B9 B8 03           110           lda   numbanks,y
D898:4A                 111           lsr   A
D899:A0 02              112           ldy   #2
D89B:91 45              113           sta   (pbuff),y
D89D:60                 114           rts

D89E:                   116 ************************************************
D89E:                   117 * XREAD - read a block
D89E:                   118 * XWRITE - write a block
```

```
34 S.EXECUTE      slinky execution routines          20-OCT-86  06:41 PAGE 135

D89E:             119 *
D89E:             120 * ProDOS read & write are changed into Protocol converter read block
D89E:             121 * and write block which are then changed into read & write
D89E:             122 ************************************************

D89E:2C 39 D8     124 xread    bit   iorts        ;V = 1 for read
D8A1:50           125          dfb   $50          ;BVC never taken
D8A2:B8           126 xwrite   clv                 ;V = 0 for write

D8A3:A5 47        128 xrwcmn   lda   block+1      ;move block & buffer pointer
D8A5:85 48        129          sta   pblock+1
D8A7:A5 46        130          lda   block        ;be careful not to step
D8A9:85 47        131          sta   pblock       ;on our own toes
D8AB:A5 45        132          lda   buffer+1
D8AD:85 46        133          sta   pbuff+1
D8AF:A5 44        134          lda   buffer
D8B1:85 45        135          sta   pbuff
D8B3:A9 00        136          lda   #0
D8B5:85 49        137          sta   pblock+2
D8B7:F0 05  D8BE  138          beq   xread2       ;skip past other sev & clv

D8B9:             140 ************************************************
D8B9:             141 * PRDBLK - Protocol converter block read
D8B9:             142 * PWRBLK - Protocol converter block write
D8B9:             143 ************************************************

D8B9:2C 39 D8     145 prdblk   bit   iorts        ;V = 1 for read
D8BC:50           146          dfb   $50          ;BVC never taken
D8BD:B8           147 pwrblk   clv

D8BE:A5 47        149 xread2   lda   pblock       ;convert block into 512 bytes
D8C0:0A           150          asl   A
D8C1:85 4A        151          sta   paddr+1
D8C3:A5 48        152          lda   pblock+1
D8C5:2A           153          rol   A
D8C6:85 4B        154          sta   paddr+2
D8C8:B0 1F  D8E9  155          bcs   prbad2       ;if C=1 then bad address
D8CA:A5 49        156          lda   pblock+2     ;third byte must be 0
D8CC:D0 1B  D8E9  157          bne   prbad2
D8CE:85 49        158          sta   paddr        ;low byte of address is 0
D8D0:85 47        159          sta   pcount       ;count = $200
D8D2:A9 02        160          lda   #2
D8D4:85 48        161          sta   pcount+1
D8D6:AD 14 C0     162          lda   rdramwrt     ;fix aux bit in address
D8D9:70 03  D8DE  163          bvs   prdread
D8DB:AD 13 C0     164          lda   rdramrd
D8DE:29 80        165 prdread  and   #$80         ;D7 = 1 if aux
D8E0:05 4B        166          ora   paddr+2
D8E2:85 4B        167          sta   paddr+2
D8E4:70 06  D8EC  168          bvs   prbad3       ;go do read
D8E6:4C F7 C5     169          jmp   sl.pwrite
D8E9:4C EE C5     170 prbad2   jmp   prbad

D8EC:4C 80 C5     172 prbad3   jmp   sl.pread
```

```
34 S.EXECUTE      slinky execution routines          20-OCT-86  06:41 PAGE 136

D8EF:             174 ************************************************
D8EF:             175 * here is the rest of the boot code
D8EF:             176 * input: a = kswh, output: v = 1 if boot fails
D8EF:             177 * jumps to DOS patch if IN# from DOS
D8EF:             178 ************************************************

D8EF:A0 C4        180 boot.sl  ldy   #4+$C0
D8F1:8C F8 07     181          sty   sl.mslot
D8F4:A2 C8        182          ldx   #4*$10+$88
D8F6:8E 78 07     183          stx   sl.devno
D8F9:CD F8 07     184          cmp   sl.mslot     ;is it a IN#
D8FC:D0 09  D907  185          bne   btnodos
D8FE:AD 00 BF     186          lda   proflag      ;are we in DOS?
D901:F0 04  D907  187          beq   btnodos      ;0 = Pascal
D903:C9 4C        188          cmp   #$4C         ;JMP = ProDOS
D905:D0 20  D927  189          bne   dospatch     ;go patch DOS

D907:9C 01 08     191 btnodos  stz   bootbuf+1    ;assume fail
D90A:             192 ;lda power2,y ;if power up bytes not set, don't boot
D90A:             193 ;eor #$A5
D90A:             194 ;cmp powerup,y
D90A:B9 B8 06     195          lda   powerup,y    ;get power up byte
D90D:C9 A5        196          cmp   #$A5         ;was it set?
D90F:D0 11  D922  197          bne   btfail       ;skip if no
D911:A0 03        198          ldy   #3
D913:B9 23 D9     199 btmv     lda   btcmd,y
D916:99 44 00     200          sta   buffer,y
D919:88           201          dey
D91A:10 F7  D913  202          bpl   btmv
D91C:AC F8 07     203          ldy   sl.mslot
D91F:20 9E D8     204          jsr   xread        ;go read the block and return
D922:60           205 btfail   rts

D923:00 08        207 btcmd    dw    $800
D925:00 00        208          dw    0            ;read in block 0 @ $800

D927:             210 ************************************************
D927:             211 * DOSPATCH - patches rwts to jump to us
D927:             212 ************************************************

D927:A9 4C        214 dospatch lda   #$4C         ;JMP opcode
D929:8D 00 BD     215          sta   rwts
D92C:A9 D1        216          lda   #>dosent4
D92E:8D 01 BD     217          sta   rwts+1       ;make page 3 vector point to us
D931:8C 02 BD     218          sty   rwts+2       ;Y = Cn
D934:A9 C3        219          lda   #>dossyn     ;patch out init command
D936:8D 1E 9D     220          sta   dosinit
D939:A9 A6        221          lda   #<dossyn
D93B:8D 1F 9D     222          sta   dosinit+1
D93E:68           223          pla                ;pop off return address
D93F:68           224          pla

D940:FA           226          plx
D941:FE 00 C0     227          inc   $C000,x      ;restore laguage card
D944:FA           228          plx                ;restore real x
D945:68           229          pla
D946:68           230          pla
```

```
D947:A2 00        232          ldx   #0
D949:A9 98        233          lda   #$98          ;return a control-X
D94B:4C 84 C7     234          jmp   swrts2        ;switch rom bank and return

D94E:60           236 dcrts    rts
```



```
D94F:             238 ************************************************
D94F:             239 * DOSCONV - changes DOS command into ours
D94F:             240 * output: command table in zp Y = command
D94F:             241 ************************************************

D94F:A0 02        243 dosconv  ldy   #ibdrvn       ;get drive 1 or 2
D951:B1 48        244          lda   (iobpl),y
D953:C9 01        245          cmp   #1            ;only 1 valid
D955:F0 03   D95A 246          beq   dcl
D957:4C 30 D8     247 dcerr    jmp   pzcmd         ;bad drive number
D95A:A0 04        248 dcl      ldy   #ibtrk        ;get track & sector
D95C:B1 48        249          lda   (iobpl),y     ;addr = 00000TTT TTTSSSSS 00000000
D95E:4A           250          lsr   A
D95F:66 4A        251          ror   paddr+1
D961:4A           252          lsr   A
D962:66 4A        253          ror   paddr+1
D964:4A           254          lsr   A
D965:85 4B        255          sta   paddr+2
D967:A5 4A        256          lda   paddr+1
D969:6A           257          ror   A
D96A:29 E0        258          and   #$E0
D96C:C8           259          iny
D96D:11 48        260          ora   (iobpl),y     ;or in sector
D96F:85 4A        261          sta   paddr+1
D971:A0 08        262          ldy   #ibbufp       ;get pointer to user's buffer
D973:B1 48        263          lda   (iobpl),y
D975:85 45        264          sta   pbuff
D977:C8           265          iny
D978:B1 48        266          lda   (iobpl),y
D97A:85 46        267          sta   pbuff+1
D97C:A0 0C        268          ldy   #ibcmd        ;get command
D97E:B1 48        269          lda   (iobpl),y
D980:F0 CC   D94E 270          beq   dcrts         ;0 = null = do nothing
D982:29 03        271          and   #3
D984:F0 D1   D957 272          beq   dcerr         ;4 = format is an error
D986:09 11        273          ora   #17           ;1 -> 17 , 2 -> 19
D988:A8           274          tay                  ;Y = command
D989:A2 00        275          ldx   #0
D98B:86 47        276          stx   pcount        ;count = $100 bytes
D98D:86 49        277          stx   paddr
D98F:E8           278          inx
D990:86 48        279          stx   pcount+1
D992:4C 1F D8     280          jmp   exec2
D995:              76          include s.makecat
```

```
D995:                 2 ************************************************
D995:                 3 * TESTSIZE - determines ramdisk size nondestructively
D995:                 4 * Inputs: Y = mslot X = devno
D995:                 5 ************************************************
D995:       D995      6 testsize  equ   *
D995:A9 00            7           lda   #0            ;Set address reg
D997:9D F8 BF         8           sta   addrl,x
D99A:9D F9 BF         9           sta   addrm,x
D99D:A9 10           10           lda   #$10          ;Start at 1 meg and go down
D99F:38              11           sec
D9A0:E9 01           12 tsloop    sbc   #1            ;Move down a bank
D9A2:9D FA BF        13           sta   addrh,x
D9A5:BD FB BF        14           lda   data,x        ;Save the data that is there
D9A8:48              15           pha
D9A9:DE F8 BF        16           dec   addrl,x       ;Fix address
D9AC:A9 A5           17           lda   #$A5          ;Store something there
D9AE:9D FB BF        18           sta   data,x
D9B1:DE F8 BF        19           dec   addrl,x
D9B4:5D FB BF        20           eor   data,x        ;0 if the data is there
D9B7:DE F8 BF        21           dec   addrl,x
D9BA:C9 01           22           cmp   #1            ;C = 0 if data ok
D9BC:68              23           pla
D9BD:9D FB BF        24           sta   data,x        ;Restore what was there
D9C0:BD FA BF        25           lda   addrh,x
D9C3:29 0F           26           and   #$0F          ;Only lower 4 bits valid
D9C5:F0 04   D9CB    27           beq   tsnoram       ;No ram somehow!!!
D9C7:B0 D7   D9A0    28           bcs   tsloop        ;Loop until we find a bank
D9C9:69 01           29           adc   #1            ;C = 0 from compare
D9CB:99 B8 03        30 tsnoram   sta   numbanks,y
D9CE:4A              31           lsr   A             ;Sizetemp = < block count
D9CF:8D 78 04        32           sta   sizetemp
D9D2:60              33           rts

D9D3:                35 ************************************************
D9D3:                36 * Routines for making a directory on the disk
D9D3:                37 ************************************************

D9D3:                39 ************************************************
D9D3:                40 * Format - Tests for powerup and puts catalog on the disk if needed
D9D3:                41 * Inputs: Y = mslot X = devno
D9D3:                42 ************************************************
D9D3:       D9D3     43 sl.format equ   *
D9D3:AC F8 07        44           ldy   sl.mslot
D9D6:                45 ;lda power2,y ;Is power2 eor powerup = A5
D9D6:                46 ;eor #$A5
D9D6:                47 ;cmp powerup,y
D9D6:                48 ;beq fmtdone
D9D6:                49 ;sta powerup,y
D9D6:A9 A5           50           lda   #$A5          ;get power up value
D9D8:D9 B8 06        51           cmp   powerup,y     ;is it = to the slinky power up byte?
D9DB:F0 2B   DA08    52           beq   fmtdone       ;skip if yes
D9DD:99 B8 06        53           sta   powerup,y     ;it's not, but it's set now

D9E0:                55 * If all screen holes spaces, someone might have cleared
D9E0:                56 * the screen wrong so we won't reformat
D9E0:C9 05           57           cmp   #$05          ;Space eor $A5?
D9E2:       D9E2     58 fmnosp    equ   *             ;Z = 1 if all spaces
D9E2:08              59           php
```



```
D9E3:20 95 D9        60           jsr   testsize
D9E6:28              61           plp
D9E7:F0 1F   DA08    62           beq   fmtdone
D9E9:AD 00 BF        63           lda   proflag       ;What type of catalog?
D9EC:F0 1B   DA09    64           beq   fmpas
D9EE:C9 4C           65           cmp   #$4C          ;JMP if ProDOS
D9F0:D0 1D   DA0F    66           bne   fmdos
D9F2:A0 FF           67           ldy   #procat       ;Do a ProDOS catalog
D9F4:20 39 DA        68           jsr   makecat       ;Put in all but bit map
D9F7:A9 01           69           lda   #01           ;Blocks 0-6 busy
D9F9:A0 20           70 fmpmap1   ldy   #32           ;32 FFs for each $100 blocks
D9FB:9D FB BF        71 fmpmap2   sta   data,x
D9FE:09 FF           72           ora   #$FF          ;Rest are FFs
DA00:88              73           dey
DA01:D0 F8   D9FB    74           bne   fmpmap2
DA03:CE 78 04        75           dec   sizetemp
DA06:D0 F1   D9F9    76           bne   fmpmap1
DA08:60              77 fmtdone   rts

DA09:                79 * Do a Pascal catalog
DA09:       DA09     80 fmpas     equ   *
DA09:A0 78           81           ldy   #pascat
DA0B:20 39 DA        82           jsr   makecat
DA0E:60              83           rts

DA0F:                85 * Do a DOS catalog
DA0F:       DA0F     86 fmdos     equ   *
DA0F:A0 2C           87           ldy   #doscat
DA11:20 39 DA        88           jsr   makecat
DA14:A9 44           89           lda   #$44          ;Point to track 3 bitmap
DA16:9D F8 BF        90           sta   addrl,x       ;Addrl = 0 from makecat
DA19:AD 78 04        91           lda   sizetemp      ;Check if at least 512K
DA1C:A0 72           92           ldy   #114          ;Assume 256K
DA1E:C9 04           93           cmp   #4            ;At least $400 blocks
DA20:90 02   DA24    94           blt   fmdbmap
DA22:A0 BA           95           ldy   #186          ;Make 400K volume
DA24:BD F8 BF        96 fmdbmap   lda   addrl,x       ;Don't free catalog
DA27:C9 7C           97           cmp   #$7C          ;Track $11?
DA29:D0 05   DA30    98           bne   fmdok
DA2B:A9 7E           99           lda   #$7E          ;Skip first 16 sectors
DA2D:9D F8 BF       100           sta   addrl,x       ;7C -> 7E so no false carry
DA30:A9 FF          101 fmdok     lda   #$FF
DA32:9D FB BF       102           sta   data,x
DA35:88             103           dey
DA36:D0 EC   DA24   104           bne   fmdbmap
DA38:60             105           rts
```

```
DA39:                 107 ************************************************
DA39:                 108 * MAKECAT - Creates a catalog
DA39:                 109 *  Inputs: X = index into catalog tables
DA39:                 110 *
DA39:                 111 ************************************************
DA39:          DA39   112 makecat   equ   *
DA39:A9 00            113          lda   #0            ;First bank is 0
DA3B:9D F8 BF         114          sta   addrl,x       ;Start at 0
DA3E:9D F9 BF         115          sta   addrm,x
DA41:9D FA BF         116          sta   addrh,x
DA44:9D FB BF         117 mcboot   sta   data,x        ;Zero out first 8 blocks
DA47:BD F9 BF         118          lda   addrm,x       ;Loop until $400
DA4A:29 F0            119          and   #$F0          ;A = 0 if < $1000
DA4C:F0 F6    DA44    120          beq   mcboot        ;Leaves us pointing at $1000 (block 8)
DA4E:A9 04            121          lda   #4            ;Point to $400 (block 2)
DA50:9D F9 BF         122          sta   addrm,x
DA53:C8               123 mcbyte   iny
DA54:B9 A7 DA         124          lda   cattbl,y      ;Get next byte from the table
DA57:C9 FD            125          cmp   #zers          ;Zeros flag?
DA59:F0 1C    DA77    126          beq   mc0
DA5B:C9 FE            127          cmp   #skpfe        ;$FE zeros?
DA5D:F0 1E    DA7D    128          beq   mcfe
DA5F:C9 FC            129          cmp   #sizeflg      ;Block size?
DA61:D0 05    DA68    130          bne   mcntsz
DA63:AD 78 04         131          lda   sizetemp      ;Get <# blocks
DA66:D0 09    DA71    132          bne   mcntnm        ;Better not be 0
DA68:C9 AA            133 mcntsz   cmp   #nameflg      ;Slot # for name?
DA6A:D0 05    DA71    134          bne   mcntnm
DA6C:AD F8 07         135          lda   sl.mslot      ;Get $Cn
DA6F:49 F0            136          eor   #$F0          ;$Cn -> $3n
DA71:9D FB BF         137 mcntnm   sta   data,x        ;Stick byte in catalog
DA74:4C 53 DA         138          jmp   mcbyte        ;Go to next byte
DA77:C8               139 mc0      iny
DA78:B9 A7 DA         140          lda   cattbl,y      ;Get # zeros
DA7B:F0 11    DA8E    141          beq   mcadd         ;If 0, it's an address
DA7D:48               142 mcfe     pha                  ;Save count
DA7E:A9 00            143          lda   #0
DA80:9D FB BF         144          sta   data,x
DA83:68               145          pla
DA84:38               146          sec
DA85:E9 01            147          sbc   #1
DA87:D0 F4    DA7D    148          bne   mcfe
DA89:F0 C8    DA53    149          beq   mcbyte        ;Always taken
DA8B:9D FB BF         150 mcadd2   sta   data,x        ;Store a 0
DA8E:DD F8 BF         151 mcadd    cmp   addrl,x       ;Finish off current page
DA91:D0 F8    DA8B    152          bne   mcadd2
DA93:C8               153          iny
DA94:B9 A7 DA         154          lda   cattbl,y      ;Get new address
DA97:F0 0D    DAA6    155          beq   mcdone        ;If 0, all done
DA99:9D F9 BF         156          sta   addrm,x
DA9C:C8               157          iny
DA9D:B9 A7 DA         158          lda   cattbl,y
DAA0:9D FA BF         159          sta   addrh,x
DAA3:4C 53 DA         160          jmp   mcbyte
DAA6:60               161 mcdone   rts
```



```
DAA7:                 163 ************************************************
DAA7:                 164 * The catalog tables
DAA7:                 165 * These tables contain the catalog info compacted
DAA7:                 166 * in a scheme so clever it probably doesn't save
DAA7:                 167 * me anything
DAA7:                 168 * For each bank:
DAA7:                 169 *  The first byte is the bank #. 0 = done
DAA7:                 170 *  zers,n = skip n 0s
DAA7:                 171 *  zers,0 = fill rest of page with 0s and get new address
DAA7:                 172 *  zers,0,0 = All done
DAA7:                 173 *  skpfe = fill $FE 0s
DAA7:                 174 *  nameflg = replace with ascii slot #
DAA7:                 175 *  sizeflg = replace with < block size
DAA7:                 176 ************************************************
DAA7:          DAA7   177 cattbl    equ   *

DAA7:          FFFF   179 procat    equ   *-cattbl-1
DAA7:00 00            180          dw    0             ;Prevoius pointer
DAA9:03 00            181          dw    3             ;Next block
DAAB:F4               182          dfb   $F4           ;Storage type
DAAC:52 41 4D         183          asc   'RAM'
DAAF:AA               184          dfb   nameflg
DAB0:FD 19            185          dfb   zers,$19
DAB2:C3 27 0D         186          dfb   $C3,$27,$0D
DAB5:00 00            187          dw    0
DAB7:06 00            188          dw    6             ;Bitmap pointer
DAB9:00 FC            189          dfb   0,sizeflg     ;Number of blocks
DABB:FD D7            190          dfb   zers,$D7
DABD:FE               191          dfb   skpfe
DABE:02 00            192          dw    2             ;Block 3 $600
DAC0:04 00            193          dw    4
DAC2:FE FE            194          dfb   skpfe,skpfe
DAC4:03 00            195          dw    3             ;Block 4 $800
DAC6:05 00            196          dw    5
DAC8:FE FE            197          dfb   skpfe,skpfe
DACA:04 00            198          dw    4             ;Block 5 $A00
DACC:00 FE            199          dfb   0,skpfe       ;Get into second page
DACE:FD 00            200          dw    zers,0,0      ;All done left at block 6

DAD4:          002C   202 doscat    equ   *-cattbl-1
DAD4:FD 00 20 02      203          dfb   zers,0,$20,$02 ;Trk $11 Sec 0 = $022000
DAD8:02               204          dfb   2             ;Sec 0 = VTOC
DAD9:11 0F            205          dfb   $11,$0F       ;Pointer to catalog
DADB:04               206          dfb   $4            ;Dos release
DADC:00 00            207          dfb   0,0           ;Unused
DADE:FB               208          dfb   $FB           ;Volume number
DADF:FD 20            209          dfb   zers,$20
DAE1:7A               210          dfb   $7A           ;TS pairs in TS list
DAE2:FD 08            211          dfb   zers,8
DAE4:FF FF FF FF      212          dfb   $FF,$FF,$FF,$FF ;Allocation mask
DAE8:32               213          dfb   $32           ;# Tracks
DAE9:20               214          dfb   $20           ;# Sectors
DAEA:00 01            215          dw    $100          ;# Bytes per sector
DAEC:FD CB            216          dfb   zers,$CB
DAEE:FE               217          dfb   skpfe         ;Sector 1 all 0s
DAEF:11 01 FE         218          dfb   $11,$01,skpfe ;Next cat sector pointer
DAF2:11 02 FE         219          dfb   $11,$02,skpfe
DAF5:11 03 FE         220          dfb   $11,$03,skpfe
```

```
35 S.MAKECAT      slinky miscellaneous routines      20-OCT-86  06:41 PAGE 143

DAF8:11 04 FE      221          dfb   $11,$04,skpfe
DAFB:11 05 FE      222          dfb   $11,$05,skpfe
DAFE:11 06 FE      223          dfb   $11,$06,skpfe
DB01:11 07 FE      224          dfb   $11,$07,skpfe
DB04:11 08 FE      225          dfb   $11,$08,skpfe
DB07:11 09 FE      226          dfb   $11,$09,skpfe
DB0A:11 0A FE      227          dfb   $11,$0A,skpfe
DB0D:11 0B FE      228          dfb   $11,$0B,skpfe
DB10:11 0C FE      229          dfb   $11,$0C,skpfe
DB13:11 0D FE      230          dfb   $11,$0D,skpfe
DB16:11 0E FE      231          dfb   $11,$0E,skpfe
DB19:FD 00 20 02   232          dfb   zers,0,$20,$02 ;Leave pointing at VTOC
DB1D:FD 00 00      233          dfb   zers,0,0       ;All done

DB20:       0078   235 pascat    equ   *-cattbl-1
DB20:00 00         236          dfb   0,0
DB22:06            237          dfb   6
DB23:FD 03         238          dfb   zers,3
DB25:04            239          dfb   4
DB26:52 41 4D      240          asc   'RAM'
DB29:AA            241          dfb   nameflg
DB2A:FD 04         242          dfb   zers,4
DB2C:FC            243          dfb   sizeflg
DB2D:FD 00 00      244          dfb   zers,0,0
DB30:               77          include s.diag0.src
```

```
36 S.DIAG0.SRC    slinky miscellaneous routines      20-OCT-86  06:41 PAGE 144

DB30:               2 ************************************************
DB30:               3 * XDIAGZ - Moves the slinky diagnostic code from $DC00 - $DFFF to
DB30:               4 * main ram starting at $2000 and jumps to $2000
DB30:               5 ************************************************
DB30:       DB30    6 xdiagz    equ   *
DB30:A2 00          7           ldx   #0
DB32:BD 00 DC       8 xdloop    lda   diagcode,x    ;move the code
DB35:9D 00 20       9           sta   diagdest,x
DB38:BD 00 DD      10           lda   diagcode+$100,x
DB3B:9D 00 21      11           sta   diagdest+$100,x
DB3E:BD 00 DE      12           lda   diagcode+$200,x
DB41:9D 00 22      13           sta   diagdest+$200,x
DB44:BD 00 DF      14           lda   diagcode+$300,x
DB47:9D 00 23      15           sta   diagdest+$300,x
DB4A:E8            16           inx
DB4B:D0 E5   DB32  17           bne   xdloop
DB4D:AE 78 07      18           ldx   sl.devno      ;get device number
DB50:A9 1F         19           lda   #<diagstart   ;put start address on stack
DB52:48            20           pha
DB53:A9 FF         21           lda   #>diagstart
DB55:48            22           pha

DB56:              24 ************************************************
DB56:              25 * switch the main rom back in and go to the diagnostics via return
DB56:              26 ************************************************

DB56:4C 84 C7      28           jmp   swrts2

DB59:       00A7   30           ds    $DC00-*,$00
DC00:              78           include vectors2

DC00:               2 ****************************************
DC00:               3 * VECTORS
DC00:               4 ****************************************
DC00:       23FA    5           ds    $FFFA-*,$00
FFFA:88 C7          6           dw    swreset2      ;NMI
FFFC:88 C7          7           dw    swreset2      ;RESET
FFFE:8E C7          8           dw    swirq2        ;INT
```

```
   3D A1H          3C A1L          FE78 A1PCLP     FE7F A1PCRTS
 FE75 A1PC         3F A2H            3E A2L          41 A3H
   40 A3L          43 A4H            42 A4L          45 A5H
   44 A5L          45 ACC          C1B3 ACDONE     04FC ACIABUF
 C24E ACIADONE   C1B4 ACIAINT      C1BA ACIAINT2     C1C2 ACIATST
 FD84 ADDINP     FDD1 ADD          BFFA ADDRH      BFF8 ADDRL
 BFF9 ADDRM      FBF8 ADV2       ?FBF4 ADVANCE     C24C AIAUX
 C24D AIEAT      C208 AIEATIT      C200 AINOFLSH     C20A AIPASS
 C1DC AIPORT2    C1D6 AITST2       C01E ALTCHARSET   C91D AMOD1
 C93A AMOD2      C93C AMOD3        C93B AMOD4        C94A AMOD5
 C94F AMOD6      CA29 AMOD7        CA38 AMOD8      ?03F5 AMPERV
 C5BB APPLE2C    FB60 APPLEII      0438 ASTAT        C580 ATALK
?FABA AUTOSCAN   D08F BACKTO1      C473 BADBITS        2D BADBLK
   01 BADCMD       21 BADCTL       C4A9 BADMAIN        04 BADPCNT
 C4B0 BADPRIM    C6A2 BADRD1       C6D3 BADREAD      C4CA BADSWTCH
   11 BADUNIT    C7C1 BANGER         2B BAS2H          2A BAS2L
 FBC1 BASCALC    FBD0 BASCLC2      FEB3 BASCONT        29 BASH
 E003 BASIC2     C317 BASICINIT    E000 BASIC        C324 BASICENT
 D400 BASICIN      28 BASL         C47D BBITS1       C4BB BBITS2
 FD71 BCKSPC     FA85 BEEPSKIP   ?FBDD BELL1         FBE4 BELL2
 FF3A BELL       C53F BIGLOOP      0215 BINH         0214 BINL
 C329 BINPUT     FE00 BL1          FE04 BLANK        FCD0 BLAST
   46 BLOCK      ?C543 BLP2        C547 BLP3         C556 BLP4
 D8EF BOOT.SL    C40E BOOT4      ?  00 BOOTBLK       0800 BOOTBUF
   4F BOOTDEV    C5F5 BOOTFAIL   ?  02 BOOTJMP         3C BOOTTMP
?C326 BPRINT     CAF1 BRANCH     ?FA4C BREAK       03F0 BRKV
 C4CE BSWTCH1    C4D8 BSWTCH2      C4E4 BSWTCH2A     C4E7 BSWTCH3
?FC10 BS         D923 BTCMD        D922 BTFAIL       D913 BTMV
 D907 BTNODOS    C439 BTOK4.1      C448 BTOK4.3      C428 BTOK4
 C43D BTOK4.2      44 BUFFER         04 BUTMODE      C061 BUTN0
 C062 BUTN1      C38A C03          C307 C3COUT1    ?C300 C3ENTRY
 C305 C3KEYIN    FD62 CANCEL       DAA7 CATTBL       D0DE CDONE
 D148 CDONE2     ?CD7D CGO         F9BA CHAR1        F9B4 CHAR2
 05FE CHARBUF    C234 CHARPTR      C136 CHKMOU         24 CH
   0D CHARCR     CDCD CHK80        FBD9 CHKBELL      CB4E CHKRT
 C130 CHOK       FF7A CHRSRCH      FFCC CHRTBL       D0A5 CKDIG
 FC9E CLEOLZ     FC46 CLEOP1       C504 CLICK        CBEE CLR0
 CBFC CLR1       CBF1 CLR2         CC02 CLR3         CBC7 CLR40
 C000 CLR80COL   C00C CLR80VID     CBDA CLR80        C00E CLRALTCHAR
?C058 CLRAN0     ?C05A CLRAN1    ?C05C CLRAN2      ?C05E CLRAN3
 FEE9 CLRCH      C2A5 CLRCOL       FC9C CLREOL       FC5D CLREOP1
 FC44 CLREOP2    FC42 CLREOP       CBCF CLRHALF      CD9B CLRIT
 CC99 CLRKBD     CFC3 CLRKBD2      FCA0 CLRLIN       CC04 CLRPORT
?CFFF CLRROM     F838 CLRSC2     ?F832 CLRSCR        C481 CLRSTS
 C491 CLRS       F83C CLRSC3       F836 CLRTOP       D112 CMD.C
 D11B CMD.C1     D0FD CMD2FOUND    D225 CMD2LIST     D087 CMD2LOOP
 D0F4 CMD2L      ?D068 CMD2NULL    D150 CMDB         D12F CMDCR
   BF CMDCUR     D14C CMDD         D139 CMDI         D13A CMDI2
 D139 CMDK       D218 CMDLIST      D091 CMDLOOP      D139 CMDL
 D12F CMDN       D14B CMDP         D183 CMDP2        D197 CMDQ
 D188 CMDR       D16C CMDS         D199 CMDT         D1D8 CMDT2
 D1EA CMDT3      D1F5 CMDTABLE     C69A CMDTBL       D0A2 CMDZ2
 D126 CMDZ       D10A CMFOUND      C168 CMLOK        C14B CMLOOP
   42 CMMAND     C18A CMNOINT      C1A1 CMNOVBL      C18E CMNOY
 C170 CMNT0      C175 CMRGHT       C182 CMROK        D0EA CMSET
 C155 CMXMOV     C37F CO1          0738 COL          FCCA COLDSTART
   30 COLOR      FCE6 COM1         FCF5 COM2         FCFB COM3
 D0BF COMINIT    D0B5 COMINIT1     D000 COMMAND      ?D011 COMMAND1
```



```
 C24F COMMPORT   C24C COMOUT       C200 COMSLOT      CF8C COMTBL
 C348 COPYROM2   C338 COPYROM      FDED COUT         FDF0 COUT1
 FDF6 COUTZ      ?FD8B CROUT1      FC62 CR           FEF6 CRMON
 FD8E CROUT      FC85 CRRTS          37 CSWH           36 CSWL
 CD2A CTLADR     CD54 CTLCHAR0     CD58 CTLCHAR      CD6F CTLDONE
 FCA4 CTLDO      CD80 CTLGO1       CD71 CTLGO          14 CTLNUM
 CD91 CTLOFF     CD95 CTLON        CD15 CTLTAB       07FB CURSOR
 C12B CVBUT      C124 CVMOVED      C118 CVNOVBL        25 CV
 FDB6 DATAOUT    BFFB DATA         D95A DC1          D957 DCERR
 D94E DCRTS      FBBC DCX          FEE2 DECCH        C2B6 DEFAULT
 C2DF DEFCOM     C2C7 DEFFF        C2EA DEFIDX       D247 DEFIDX2
 C2BC DEFLOOP    C6D9 DENIB1       C6D7 DENIBL       C22B DEVNO
 C142 DEVNO2     DC00 DIAGCODE     2000 DIAGDEST     D48E DIAGS
 1FFF DIAGSTART  FF8A DIG          D0AD DIGLOOP      D0C7 DISABLE
?C983 DISLIN     0356 DNIBL        CBC2 DOCLR        FBB4 DOCOUT1
 FB54 DOCTL      C9D8 DOINST       C46E DOIT4        C9F4 DOLIN
 C471 DONE4      C186 DONE         FD20 DONXTCUR     FECE DOPR0
 C462 DOS24      002C DOSCAT       D840 DOSCONV2     D94F DOSCONV
 C4D1 DOSENT4      80 DOSERR       9D1E DOSINIT      C4E9 DOSOK4
 D927 DOSPATCH   C4E0 DOSSLT4      A6C3 DOSSYN       C566 DQUIT
?C60B DRV2ENT    D469 DV10LOOP     D470 DV10LT       D0C5 ENABLE
 C454 ENT4       C111 ENTR1        C219 ENTR         C44E ENTRY4
 C9C9 ERR2       ?C9CB ERR3        04F8 ERROR        F8A1 ERR
   9B ESC        CCD7 ESC0       ?CCE3 ESC1          CCE5 ESC2
 CCC0 ESC3       CD0C ESCCHAR      0638 ESCHAR       0013 ESCNUM
 CCED ESCRDKEY   CCF8 ESCTAB       D81F EXEC2        D824 EXEC3
 D800 EXECUTE    C275 EXIT1        C273 EXITX        C63D EXTENT1
?C65C EXTENT     0538 EXTINT       05F9 EXTINT2      F800 F8ORG
 FBB3 F8VERSION  C140 FIXCH        C80E FIXLC      ?FA9B FIXSEV
 D077 FLAGIT     06B8 FLAGS      ?D1EE FLUSH         DA24 FMDBMAP
 DA30 FMDOK      DA0F FMDOS      ?D9E2 FMNOSP        DA09 FMPAS
 D9F9 FMPMAP1    D9FB FMPMAP2      F962 FMT1         F9A6 FMT2
 DA08 FMTDONE    CD67 FNDCTL         2E FORMAT     ?C648 FUGIT
 F847 GBASCALC     27 GBASH          26 GBASL        C8C9 GBBRK
 F856 GBCALC     C321 GBDONE       C8C1 GBNOC        C30D GBNOOVR
 C8C7 GBNOTROM   C346 GDEAT        C334 GDNOLF       C340 GDNXON
 C348 GDOK       C393 GETALT1      C398 GETALT2      C37C GETALT
 C2F7 GETBUF     C2FD GETBUF2      C3A6 GETCOUT      CCA7 GETCUR1
 CCAD GETCUR2    CCB7 GETCUR3      CC9D GETCUR       CCBF GETCURX
 C322 GETDATA    F8A5 GETFMT       C9E7 GETI1        FC80 GETINDX
 C986 GETINST1   C816 GETLC      ?FD6F GETLN1      ?FD6A GETLN
 FD67 GETLNZ     FFA7 GETNUM       C98F GETOP        C2B2 GETSTAT2
 C2AC GETSTAT    CB57 GETST        C5B4 GETUP        CEFA GETX
?CF06 GETY       CF38 GKEY         C826 GLCBNK1      C829 GLCDONE
 0011 GLUIDX     C5EE GOBASICIN    C8A7 GOBREAK      CB25 GODDONE
 CB22 GODREG     C96E GOERR        C9EC GOERR2         06 GOODF8
 C278 GOREMOTE   C19B GOSER3       C279 GOTERM       FEB6 GO
 CB0D GODSP    ?FD25 GOTKEY        F8CC GOTONE       C2C1 GSTNOINT
 C2B4 GSTTST       2C H2           C4C8 HANGX        C4ED HANGY
 D489 HDDONE     D464 HDLOOP       D45E HDPOS2     ?FCC9 HEADR
 D452 HEXDEC2    D441 HEXTODEC   ?C057 HIRES       ?F819 HLINE
 F81C HLINE1     CDA5 HOMECUR      FC58 HOME         CE1B HOOKITUP
 CE20 HOOKUP     D600 I.NITMOUSE     08 IBBUFP         0C IBCMD
   02 IBDRVN   ?  05 IBSECT          01 IBSLOT         0D IBSTAT
   04 IBTRK      F897 IEVEN        0200 INBUF        D084 INCMD1
 D03C INCMD3     C705 INENT        0200 IN           D022 INCMD
 D032 INCMD2     FF15 INDX         CB05 INITBL       C740 INITMOUSE
 FB2F INIT     ?FE8B INPORT        FE8D INPRT        F882 INSDS1
```

F88E	INSDS2	F8D0	INSTDSP	CC12	INVERT	32	INVFLG
CC1C	INVX	C000	IOADR	49	IOBPH	48	IOBPL
? 27	IOERR	FEDE	IOPRT1	FEAB	IOPRT2	FE9B	IOPRT
D839	IORTS	C078	IOUDSBL	C079	IOUENBL	0009	IOUIDX
C058	IOU	C82A	IRQ21	C826	IRQ2	C834	IRQ3
C83E	IRQ4	C848	IRQ5	C85B	IRQ6	C85E	IRQ7
C870	IRQ8	C88C	IRQDN1	C88E	IRQDN2	C8A4	IRQDN5
C87F	IRQDONE	C804	IRQENT	?03FE	IRQLOC	?FA40	IRQ
C896	IRQDN3	C89C	IRQDN4	C882	IRQLCOK	CF86	IRQTBLE
FFFE	IRQVECT	C663	ISMRK1	C3C3	JMPDEST	C32C	JPINIT
C32F	JPREAD	C335	JPSTAT	C332	JPWRITE	C010	KBDSTRB
FB88	KBDWAIT	C000	KBD	FD1B	KEYIN	?FD18	KEYIN0
39	KSWH	38	KSWL	CFE1	LACR	CFDE	LADIG
CFE4	LADONE	C08B	LCBANK1	C083	LCBANK2	2F	LENGTH
8A	LFEED	FC66	LF	0400	LINE1	FE63	LIST2
FE5E	LIST	2C	LMNEM	00	LOC0	01	LOC1
CFCB	LOOKASC	FD38	LOOKPICK	C056	LORES	FE22	LT2
FE20	LT	C746	M..OVEIRQ	? 40	M.40	08	M.CTL
20	M.CTL2	10	M.CURSOR	08	M.GOXY	01	M.MOUSE
CF9A	M.OVEIRQ	80	M.PASCAL	04	M.VMODE	44	MACSTAT
DA39	MAKECAT	C58E	MAKTBL	2E	MASK	D202	MASK1
D20D	MASK2	05F8	MAXH	04F8	MAXL	077D	MAXXH
067D	MAXXL	?07FD	MAXYH	?06FD	MAXYL	C700	MBASIC
C5EA	MBBAD	DA77	MC0	DA8E	MCADD	DA8B	MCADD2
DA44	MCBOOT	DA53	MCBYTE	DAA6	MCDONE	DA7D	MCFE
DA71	MCNTNM	DA68	MCNTSZ	C3D0	MEM1	C3D8	MEM2
C3F3	MEM3	C3F5	MEM4	C3FA	MEM5	C405	MEM6
C412	MEM7	C42A	MEM8	C42C	MEM9	C431	MEMA
C440	MEMB	C44F	MEMC	C456	MEMD	C472	MEMERROR
C46C	MEMF	0578	MINH	FE6C	MINI	C9C7	MINIERR
0478	MINL	057D	MINXH	047D	MINXL	?05FD	MINYH
?04FD	MINYL	CFA9	MIRQLP	CFC0	MIRQSTD	?C052	MIXCLR
C053	MIXSET	0001	MMUIDX	F9C0	MNEML	FA00	MNEMR
F8BE	MNNDX1	F8C2	MNNDX2	F8C9	MNNDX3	FDAD	MOD8CHK
31	MODE	FF69	MONZ	FF65	MON	?FF59	MONITOR
067F	MOUARM	C063	MOUBUT	C048	MOUCLR	?C058	MOUDSBL
?C059	MOUENBL	07FF	MOUMODE	C100	MOUSEINT	CD9F	MOUSOFF
CD99	MOUSON	077F	MOUSTAT	0478	MOUTEMP	C066	MOUX1
057F	MOUXH	C015	MOUXINT	047F	MOUXL	C067	MOUY1
05FF	MOUYH	C017	MOUYINT	04FF	MOUYL	C972	MOV1
20	MOVARM	C34E	MOVEAUX	FE2C	MOVE	C361	MOVEC2M
CFA0	MOVEIRQ	C367	MOVELOOP	C393	MOVERET	C367	MOVESTRT
C970	MOVINST	02	MOVMODE	C900	MPADDLE	D179	MSLOOP
07F8	MSLOT	D177	MSWAIT	AA	NAMEFLG	CAFF	NBRNCH
0300	NBUF1	28	NDERR	FBB0	NEWADV1	FBA0	NEWADV
FA47	NEWBRK	FC99	NEWC1	FC90	NEWCLEOLZ	FC8D	NEWCLREOL
FC73	NEWCR	CCCC	NEWESC	C803	NEWIRQ	?FA81	NEWMON
FC38	NEWOP1	FC35	NEWOPS	CAD1	NEWPCL	FC88	NEWVTABZ
FC86	NEWVTAB	C371	NEXTA1	03FB	NMI	CA3B	NNBL
D020	NOCMD2	D01F	NOCMD	C71A	NOERROR	?FD45	NOESC1
FD4A	NOESC2	FD44	NOESCAPE	C254	NOESC	FAA3	NOFIX
C5AA	NOPATRN	C371	NOREAD	D166	NOSHIFT	D6CC	NOSTAT2
C36A	NOT1	C1B2	NOTACIA	FD5F	NOTCR1	FD4D	NOTCR
CC6B	NOTINV2	CC53	NOTINV	?CC68	NOTINV1	FEA7	NOTPRT0
FB94	NOWAIT	C82A	NTBL	03B8	NUMBANKS	077E	NUMBER
0016	NUMOPS	FCBA	NXTA1	FCB4	NXTA4	FF98	NXTBAS
FF90	NXTBIT	FFA2	NXTBS2	C9F8	NXTCH	FD75	NXTCHAR
FFAD	NXTCHR	?F85F	NXTCOL	077B	NXTCUR	FF73	NXTITM

CA06	NXTMN	C9BD	NXTOP	FA59	OLDBRK	047B	OLDCH
0679	OLDCUR	067A	OLDCUR2	?FF59	OLDRST	D07F	ONELETTER
FEC2	OPRT0	FEFE	OPTBL	057B	OURCH	05FB	OURCV
C707	OUTENT	?FE95	OUTPORT	FE97	OUTPRT	C1D5	P1ERR
C19E	P1INIT	C1AF	P1READ2	C1A8	P1READ	C9AD	P1SKIP
C1BB	P1STATUS	C1CE	P1STRD	C1CC	P1STWR	C1B4	P1WRITE
C211	P2INIT	C213	P2READ	C217	P2STATUS	C215	P2WRITE
C064	PADDL0	49	PADDR	C680	PARMTBL	CF71	PASCALC
0078	PASCAT	?CF7F	PASCLC2	CC0B	PASINVERT	CF35	PASREAD
C850	PASSKIP1	C23D	PBFULL	47	PBLOCK	C235	PBOK
45	PBUFF	F954	PCADJ2	F953	PCADJ	F956	PCADJ3
F95C	PCADJ4	C48D	PCBAD4	C489	PCCMD4	C48F	PCERR4
C4B2	PCGTP4	3B	PCH	CAB4	PCINC2	CAB6	PCINC3
3A	PCL	D86A	PCNTL	D873	PCNTOK	C880	PCNV
C5F8	PCNVRST	C494	PCONV4	47	PCOUNT	C4BD	PCPARMS4
11	PCREVNUM	?C4B5	PCSKP4	C4A8	PCSVZP4	CF19	PCTL
C918	PDOK	C90D	PDON	CC3D	PICK1	CC33	PICK2
95	PICK	CC3F	PICK3	CC4A	PICK4	CC1D	PICKY
CF41	PINIT	CEBC	PIORDY	F800	PLOT	F80E	PLOT1
CEC0	PNOTRDY	C702	PNULL	06B8	POWERUP	43	PPARM
FD92	PRA1	F910	PRADR1	F914	PRADR2	F926	PRADR3
F92A	PRADR4	F930	PRADR5	D8E9	PRBAD2	D8EC	PRBAD3
C5F3	PRBADZ	C5EE	PRBAD	F94A	PRBL2	?F94C	PRBL3
F948	PRBLNK	FDDA	PRBYTE	D8B9	PRDBLK	C5E3	PRDONE
D8DE	PRDREAD	?FB1E	PREAD	D83A	PREAD2.%	FB25	PREAD2
?FF2D	PRERR	CEF7	PRET	?FDE3	PRHEX	FDE5	PRHEX%
C5C9	PRLAST	C5D3	PRLOOP2	C5B5	PRLOOP	C5AC	PRMAIN
C5EC	PRMAIN2	F8F5	PRMN1	F8F9	PRMN2	C166	PRNOW
?F941	PRNTAX	F8DB	PRNTBL	F8D4	PRNTOP	?F944	PRNTX
F940	PRNTYX	C14A	PRNT	FFFF	PROCAT	C5D9	PRODD
BF00	PROFLAG	? 03	PROFORM	33	PROMPT	? 01	PROREAD
? 00	PROSTAT	? 02	PROWRIT	FD96	PRYX2	CF66	PS1
CF51	PSETUP	CF54	PSETUP2	CF30	PSETX	D880	PST0
D853	PSTAT0	CEB1	PSTATUS	47	PSTAT	CEBE	PSTERR
D88A	PSTMOV	?C070	PTRIG	44	PUNIT	C228	PUTBUF
?D43B	PUTINBUF	CE3B	PVMODE	C65A	PWDONE	04B8	PWDTH
C640	PWLAST	C62C	PWLOOP	C64A	PWLOOP2	C663	PWMAIN2
C623	PWMAIN	C650	PWODD	CEDD	PWR1	D8BD	PWRBLK
FAFD	PWRCON	03F4	PWREDUP	CEF4	PWRET	D83D	PWRITE2
CEC2	PWRITE	CEF1	PWRITERET	FAA6	PWRUP	FB12	PWRUP2
D830	PZCMD	D834	PZCNT	D836	PZCNT2	D4DA	QTBL
CE45	QUIT	CE44	QX	D241	R.GETALT1	D22A	R.GETALT
D246	R.GETALT2	C484	RATS4	?C060	RD40SW	C018	RD80COL
C01F	RD80VID	C63F	RDADR	C016	RDALTZP	C6A8	RDAT0
C6AA	RDAT1	C6BA	RDAT2	C6BC	RDAT3	C6CB	RDAT4
C6A6	RDATA	C003	RDCARDRAM	?FD35	RDCHAR	C642	RDDHDR
C656	RDHD0	C65E	RDHD1	C667	RDHD2	C671	RDHD3
?C01D	RDHIRES	FD0C	RDKEY	C011	RDLCBNK2	C012	RDLCRAM
C002	RDMAINRAM	?C01B	RDMIX	C01C	RDPAGE2	C013	RDRAMRD
C014	RDRAMWRT	C685	RDSEC1	C687	RDSEC2	C68F	RDSEC3
C683	RDSECT	FAE4	RDSP1	C01A	RDTEXT	C019	RDVBLBAR
?FEFD	READ	FAD7	REGDSP	FEBF	REGZ	C961	REL1
C96B	REL2	?F938	RELADR	C955	REL	FA62	RESET
FABD	RESET.X	C354	RESETLC	FF3F	RESTORE	?FF44	RESTR1
C641	RETRY1	C657	RETRY	0101	REVNUM	FADA	RGDSP1
FB02	RGDSP2	C866	RMESS	2D	RMNEM	4F	RNDH
4E	RNDL	C028	ROMBANK	C081	ROMIN	C37B	ROMOK
0478	ROMSTATE	C473	RSLOOP4	C853	RSWTBL	CF94	RTBL

F80C RTMASK	F87F RTMSKZ	2D RTNH	?CAD5 RTNJMP
CAD9 RTNJMP2	2C RTNL	F831 RTS1	FBEF RTS2B
F961 RTS2	FB2E RTS2D	FBFC RTS3	?FDC5 RTS4C
?FC34 RTS4	FCC8 RTS4B	FE17 RTS5	?FCB3 RTS6
BD00 RWTS	?FF4C SAV1	FF4A SAVE	BFFB SCNTL
BFFA SCOMD	CE58 SCR1	CE5E SCR2	CE66 SCR3
CE79 SCR4	CE82 SCR5	CE8B SCR6	CE96 SCR7
?CE8D SCR8	CEAD SCR9	05B8 SCREEN	CBB9 SCRL3
CB9B SCRLEVEN	CBA2 SCRLFT	CB6D SCRLIN	CBB0 SCRLODD
F879 SCRN2	?F871 SCRN	CE80 SCRN48	CE53 SCRN84
CB30 SCROLLDN	CB38 SCROLLIT	CB35 SCROLLUP	?FC70 SCROLL
BFF8 SDATA	C61F SEEKZERO	C27F SERIN	C11C SERISOUT
03B8 SERMODE	C18F SEROUT2	C18A SEROUT	C24F SEROUT3
C255 SEROUT4	C117 SERPORT	C189 SERRTS	C100 SERSLOT
C144 SERVID	CDC0 SET40	CDBE SET80	C001 SET80COL
C00D SET80VID	C00F SETALTCHAR	C009 SETALTZP	?C059 SETAN0
?C05B SETAN1	?C05D SETAN2	C05F SETAN3	C182 SETCH
?F864 SETCOL	FEEE SETCUR1	FEEC SETCUR	CB67 SETDBAS
?FB40 SETGR	CE23 SETHOOKS	FE86 SETIFLG	FE80 SETINV
?D638 SETIOU	CDA1 SETIT	FE89 SETKBD	FE1D SETMDZ
FE18 SETMODE	FE84 SETNORM	?FAA9 SETPG3	FAAB SETPLP
?FB6F SETPWRC	C360 SETROM	CB88 SETSRC	C008 SETSTDZP
D1A0 SETTERM	?FB39 SETTXT	C21C SETUP	CB83 SETUP2
FE93 SETVID	C82A SETV	FB4B SETWND	CE1A SETX
CBC1 SEV1	CC4C SHOWCUR	C5C4 SHOWINST	C28E SIDATA
D642 SILOOP	C2AC SINOMOD	C205 SIN	D649 SINOCH
C280 SINOKBD	FC SIZEFLG	0478 SIZETEMP	FE SKPFE
CBA8 SKPLFT	CBB4 SKPRT	0778 SL.DEVNO	D9D3 SL.FORMAT
0678 SL.LCSTATE	07F8 SL.MSLOT	C580 SL.PREAD	D874 SL.PSTATUS
C5F7 SL.PWRITE	03B8 SL.SCRN1	?0438 SL.SCRN2	?04B8 SL.SCRN3
?0538 SL.SCRN4	?05B8 SL.SCRN5	?0638 SL.SCRN6	06B8 SL.SCRN7
?0738 SL.SCRN8	C74C SLBOOT	? 04 SLOT	2B SLOTZ
C1 SLTDMY	C752 SLXEQ	C86C SMESS	D650 SMINVALID
C2AA SODONE	03F2 SOFTEV	C28D SOOK	C25E SORDY
C2AB SORTS	C286 SOTST	C207 SOUT	C030 SPKR
49 SPNT	BFF9 SSTAT	D860 STOLP	CF29 STARTXY
C667 STATTBL	48 STATUS	D86E STBAD	D1B9 STCLR
FE71 STEPZ	CA43 STEP	FB65 STITLE	FBF0 STORADV
C3B8 STORCH	C3DB STORE1	C3EE STORE2	?C3F2 STORE3
C3C1 STORE	C3F9 STORE5	?FE0B STOR	?C3F7 STORE4
C3B3 STORY	D1F4 STRTS	D1C0 STSET	D1C9 STWASOK
FFE3 SUBTBL	C56B SUC2	C875 SUCCESS	C22F SUDODEF
C245 SUDONE	C232 SUNODEF	C240 SUOUT	C704 SW.INITMOUSE
C6EE SW.MCLAMP	C6E3 SW.MCLEAR	C6F9 SW.MHOME	C6D8 SW.MREAD
C6CD SW.MTSTINT	C6C2 SW.SETMOU	?C7C7 SWATALK	C7AF SWAUX
C79D SWBASICIN	C4EF SWCHTST	C7A9 SWCMD	C806 SWCMD3
C537 SWERR	C7DF SWGETB	C7D3 SWGETST	C78E SWIRQ2
C7BB SWMINT	?C797 SWPCNV	C7D9 SWREAD	?C788 SWRESET
C788 SWRESET2	?C780 SWRTI2	C780 SWRTI	C784 SWRTS2
C784 SWRTS	C787 SWRTSOP	C7CD SWSER3	C6B4 SWSL.BT
C797 SWSTHK2	C7F1 SWSTHK3	C7A3 SWSTTM	C7F1 SWSTTM3
C82F SWTBL0	C841 SWTBL1	C4F1 SWTST1	C4F3 SWTST2
C4FE SWTST3	C508 SWTST4	C51A SWTST5	C521 SWTST6
C533 SWTST7	C7B5 SWXFER	?C7EB SWXFGO	C7EB SWXFGO2
C7E5 SWZZNM	C79D SWZZQT2	C7F6 SWZZQT3	?FB5B TABV
C15E TAB	C592 TBLLOOP	C5A0 TBLLOOP2	06F8 TEMP
04F8 TEMP1	0578 TEMPA	4A TEMPPTR	05F8 TEMPY
C27C TERM1	DF TERMCUR	C25E TESTKBD	D995 TESTSIZE

0800 THBUF	?FB09 TITLE	C15C TOOFAR	FFBE TOSUB
FE6F TRACE	06FF TRKEY	067C TRSER	D9A0 TSLOOP
D9CB TSNORAM	C3C6 TSTMEM	D4C2 TSTMEM2	D497 TSTZPG
05FC TWKEY	057C TWSER	C050 TXTCLR	C054 TXTPAGE1
C055 TXTPAGE2	C051 TXTSET	05FA TYPHED	00 UCSPACE
CC93 UD2	43 UNIT	CC70 UPDATE	C399 UPSHIFT0
C39B UPSHIFT	FC1A UP	FECA USR	03F8 USRADR
C1DB UTSMSG	2D V2	C070 VBLCLR	C019 VBLINT
0C VBLMODE	FE36 VERIFY	067B VFACTV	FE58 VFYOK
CE31 VIDMODE	FC04 VIDOUT1	FBFD VIDOUT	FB78 VIDWAIT
F828 VLINE	F826 VLINEZ	04FB VMODE	FC22 VTAB
FB59 VTAB23	FC30 VTAB40	FC24 VTABZ	FCA8 WAIT
FCA9 WAIT2	FCAA WAIT3	FEEB WDTHCH	CDD5 WIN0
CDE0 WIN1	CDED WIN2	CDF2 WIN3	CDD2 WIN40
CE02 WIN4	CE18 WIN5	CDD4 WIN80	23 WNDBTM
20 WNDLFT	CE0A WNDREST	22 WNDTOP	21 WNDWDTH
C005 WRCARDRAM	?FECD WRITE	C004 WRMAINRAM	CD8D X.CUR.OFF
CD89 X.CUR.ON	D6A3 X.MCLAMP	D668 X.MCLEAR	D651 X.MHOME
D679 X.MREAD	D6C2 X.MTSTINT	D621 X.SETMOU	CDB7 X.SI
CDB0 X.SO	C3A5 X.UPSHIFT	FDB3 XAM	FDA3 XAM8
FDC6 XAMPM	FEB0 XBASIC	C8E6 XBITKBD	C8F9 XBKB1
C8FB XBKB2	CAA6 XBRK	06FB XCOORD	D843 XDIAG
DB30 XDIAGZ	DB32 XDLOOP	C3B0 XFERZP	C397 XFER
C3C0 XFERAZP	C3AA XFERC2M	CAC9 XJMPAT	CAC8 XJMP
CAE3 XJMPATX	CAC0 XJSR	CAEE XJXNOC	C5CF XMBASIC
C5DC XMBOUT	D674 XMCDONE	C734 XMCLAMP	C72E XMCLEAR
C1AD XMDONE	D657 XMH2	D655 XMHLOOP	C73A XMHOME
D69C XMRD2	C728 XMREAD	C722 XMTSTINT	C8D4 XNOKEY
C2D5 XNOSBUF	93 XOFF	91 XON	CA98 XQ1
CA9A XQ2	CA64 XQINIT	CA50 XQNOBT0	CA90 XQNTBRA
3C XQT	CA4A XQWAIT	D690 XRBUT	D697 XRBUT2
C2F4 XRDDONE	C8D5 XRDKBD	C2E9 XRDNOBUF	C2C9 XRDSER2
C2C3 XRDSER	D8BE XREAD2	D0DB XREADY	D89E XREAD
46 XREG	C8CC XRKBD1	D607 XRLOOP	CAAC XRTI
CAB0 XRTS	?D8A3 XRWCMN	C71C XSETMOU	D636 XSOFF
D846 XSTATUS	0578 XVAL	D8A2 XWRITE	?C100 XXX
0008 YHI	47 YREG	35 YSAV1	34 YSAV
05F8 YVAL	FD ZERS	FFC7 ZMODE	D49B ZP1
D4A4 ZP2	D4B7 ZP3	D4BC ZPERROR	C464 ZSAVE4
0A ZUSED	D4CE ZZLOOP	D4C5 ZZNM	CE4D ZZQUIT

```
   00 LOC0          ? 00 BOOTBLK       ? 00 PROSTAT         00 UCSPACE
   01 BADCMD          01 LOC1            01 M.MOUSE       ? 01 PROREAD
   01 IBSLOT        0001 MMUIDX        ? 02 BOOTJMP         02 IBDRVN
 ? 02 PROWRIT         02 MOVMODE       ? 03 PROFORM         04 IBTRK
   04 BUTMODE         04 BADPCNT       ? 04 SLOT            04 M.VMODE
 ? 05 IBSECT          06 GOODF8          08 M.CTL           08 M.GOXY
 0008 YHI             08 IBBUFP        0009 IOUIDX          0A ZUSED
   0C VBLMODE         0C IBCMD           0D IBSTAT          0D CHARCR
   10 M.CURSOR        11 BADUNIT       0011 GLUIDX          11 PCREVNUM
 0013 ESCNUM          14 CTLNUM        0016 NUMOPS          20 WNDLFT
   20 M.CTL2          20 MOVARM          21 BADCTL          21 WNDWDTH
   22 WNDTOP          23 WNDBTM          24 CH              25 CV
   26 GBASL           27 GBASH         ? 27 IOERR           28 NDERR
   28 BASL            29 BASH            2A BAS2L           2B BAS2H
   2B SLOTZ           2C H2            002C DOSCAT          2C LMNEM
   2C RTNL            2D BADBLK          2D RMNEM           2D V2
   2D RTNH            2E MASK            2E FORMAT          2F LENGTH
   30 COLOR           31 MODE            32 INVFLG          33 PROMPT
   34 YSAV            35 YSAV1           36 CSWL            37 CSWH
   38 KSWL            39 KSWH            3A PCL             3B PCH
   3C A1L             3C XQT             3C BOOTTMP         3D A1H
   3E A2L             3F A2H             40 A3L           ? 40 M.40
   41 A3H             42 CMMAND          42 A4L             43 UNIT
   43 PPARM           43 A4H             44 A5L             44 MACSTAT
   44 BUFFER          44 PUNIT           45 A5H             45 PBUFF
   45 ACC             46 BLOCK           46 XREG            47 YREG
   47 PSTAT           47 PCOUNT          47 PBLOCK          48 STATUS
   48 IOBPL           49 PADDR           49 IOBPH           49 SPNT
   4A TEMPPTR         4E RNDL            4F RNDH            4F BOOTDEV
 0078 PASCAT          80 DOSERR          80 M.PASCAL        8A LFEED
   91 XON             93 XOFF            95 PICK            9B ESC
   AA NAMEFLG         BF CMDCUR          C1 SLTDMY          DF TERMCUR
   FC SIZEFLG         FD ZERS            FE SKPFE         0101 REVNUM
 0200 IN            0200 INBUF         0214 BINL          0215 BINH
 0300 NBUF1         0356 DNIBL         03B8 SERMODE       03B8 NUMBANKS
 03B8 SL.SCRN1      03F0 BRKV          03F2 SOFTEV        03F4 PWREDUP
?03F5 AMPERV        03F8 USRADR        03FB NMI           ?03FE IRQLOC
 0400 LINE1         0438 ASTAT        ?0438 SL.SCRN2      0478 ROMSTATE
 0478 SIZETEMP      0478 MOUTEMP       0478 MINL          047B OLDCH
 047D MINXL         047F MOUXL         04B8 PWDTH        ?04B8 SL.SCRN3
 04F8 ERROR         04F8 MAXL          04F8 TEMP1         04FB VMODE
 04FC ACIABUF      ?04FD MINYL         04FF MOUYL         0538 EXTINT
?0538 SL.SCRN4      0578 MINH          0578 TEMPA         0578 XVAL
 057B OURCH         057C TWSER         057D MINXH         057F MOUXH
 05B8 SCREEN       ?05B8 SL.SCRN5      05F8 MAXH          05F8 TEMPY
 05F8 YVAL          05F9 EXTINT2       05FA TYPHED        05FB OURCV
 05FC TWKEY        ?05FD MINYH         05FE CHARBUF       05FF MOUYH
 0638 ESCHAR       ?0638 SL.SCRN6      0678 SL.LCSTATE    0679 OLDCUR
 067A OLDCUR2       067B VFACTV        067C TRSER         067D MAXXL
 067F MOUARM        06B8 FLAGS         06B8 POWERUP       06B8 SL.SCRN7
 06F8 TEMP          06FB XCOORD       ?06FD MAXYL         06FF TRKEY
 0738 COL          ?0738 SL.SCRN8      0778 SL.DEVNO      077B NXTCUR
 077D MAXXH         077E NUMBER        077F MOUSTAT       07F8 MSLOT
 07F8 SL.MSLOT      07FB CURSOR       ?07FD MAXYH         07FF MOUMODE
 0800 THBUF         0800 BOOTBUF       1FFF DIAGSTART     2000 DIAGDEST
 9D1E DOSINIT       A6C3 DOSSYN        BD00 RWTS          BF00 PROFLAG
 BFF8 ADDRL         BFF8 SDATA         BFF9 SSTAT         BFF9 ADDRM
 BFFA ADDRH         BFFA SCOMD         BFFB SCNTL         BFFB DATA
```



```
 C000 CLR80COL      C000 IOADR         C000 KBD           C001 SET80COL
 C002 RDMAINRAM     C003 RDCARDRAM     C004 WRMAINRAM     C005 WRCARDRAM
 C008 SETSTDZP      C009 SETALTZP      C00C CLR80VID      C00D SET80VID
 C00E CLRALTCHAR    C00F SETALTCHAR    C010 KBDSTRB       C011 RDLCBNK2
 C012 RDLCRAM       C013 RDRAMRD       C014 RDRAMWRT      C015 MOUXINT
 C016 RDALTZP       C017 MOUYINT       C018 RD80COL       C019 VBLINT
 C019 RDVBLBAR      C01A RDTEXT      ?C01B RDMIX          C01C RDPAGE2
?C01D RDHIRES       C01E ALTCHARSET    C01F RD80VID       C028 ROMBANK
 C030 SPKR          C048 MOUCLR        C050 TXTCLR        C051 TXTSET
?C052 MIXCLR        C053 MIXSET        C054 TXTPAGE1      C055 TXTPAGE2
 C056 LORES       ?C057 HIRES        ?C058 MOUDSBL      ?C058 CLRAN0
 C058 IOU         ?C059 MOUENBL      ?C059 SETAN0       ?C05A CLRAN1
?C05B SETAN1      ?C05C CLRAN2       ?C05D SETAN2       ?C05E CLRAN3
 C05F SETAN3      ?C060 RD40SW       C061 BUTN0         C062 BUTN1
 C063 MOUBUT        C064 PADDL0      C066 MOUX1         C067 MOUY1
 C070 VBLCLR      ?C070 PTRIG        C078 IOUDSBL       C079 IOUENBL
 C081 ROMIN         C083 LCBANK2     C08B LCBANK1       C100 SERSLOT
?C100 XXX           C100 MOUSEINT      C111 ENTR1       C117 SERPORT
 C118 CVNOVBL       C11C SERISOUT      C124 CVMOVED       C12B CVBUT
 C130 CHOK          C136 CHKMOU        C140 FIXCH         C142 DEVNO2
 C144 SERVID        C14A PRNT          C14B CMLOOP        C155 CMXMOV
 C15C TOOFAR        C15E TAB           C166 PRNOW         C168 CMLOK
 C170 CMNT0         C175 CMRGHT        C182 SETCH         C182 CMROK
 C186 DONE          C189 SERRTS        C18A SEROUT        C18A CMNOINT
 C18E CMNOY         C18F SEROUT2       C19B GOSER3        C19E P1INIT
 C1A1 CMNOVBL       C1A8 P1READ        C1AD XMDONE        C1AF P1READ2
 C1B2 NOTACIA       C1B3 ACDONE        C1B4 ACIAINT       C1B4 P1WRITE
 C1BA ACIAINT2      C1BB P1STATUS      C1C2 ACIATST       C1CC P1STWR
 C1CE P1STRD        C1D5 P1ERR         C1D6 AITST2        C1DB UTSMSG
 C1DC AIPORT2       C200 COMSLOT       C200 AINOFLSH      C205 SIN
 C207 SOUT          C208 AIEATIT       C20A AIPASS        C211 P2INIT
 C213 P2READ        C215 P2WRITE       C217 P2STATUS      C219 ENTR
 C21C SETUP         C228 PUTBUF        C22B DEVNO         C22F SUDODEF
 C232 SUNODEF       C234 CHARPTR       C235 PBOK          C23D PBFULL
 C240 SUOUT         C245 SUDONE        C24C COMOUT        C24C AIAUX
 C24D AIEAT         C24E ACIADONE      C24F SEROUT3       C24F COMMPORT
 C254 NOESC         C255 SEROUT4       C25E TESTKBD       C25E SORDY
 C273 EXITX         C275 EXIT1         C278 GOREMOTE      C279 GOTERM
 C27C TERM1         C27F SERIN         C280 SINOKBD       C286 SOTST
 C28D SOOK          C28E SIDATA        C2A5 CLRCOL        C2AA SODONE
 C2AB SORTS         C2AC SINOMOD       C2AC GETSTAT       C2B2 GETSTAT2
 C2B4 GSTTST        C2B6 DEFAULT       C2BC DEFLOOP       C2C1 GSTNOINT
 C2C3 XRDSER        C2C7 DEFFF         C2C9 XRDSER2       C2D5 XNOSBUF
 C2DF DEFCOM        C2E9 XRDNOBUF      C2EA DEFIDX        C2F4 XRDDONE
 C2F7 GETBUF        C2FD GETBUF2     ?C300 C3ENTRY        C305 C3KEYIN
 C307 C3COUT1       C30D GBNOOVR       C317 BASICINIT     C321 GBDONE
 C322 GETDATA       C324 BASICENT    ?C326 BPRINT         C329 BINPUT
 C32C JPINIT        C32F JPREAD        C332 JPWRITE       C334 GDNOLF
 C335 JPSTAT        C338 COPYROM       C340 GDNXON        C346 GDEAT
 C348 GDOK          C348 COPYROM2      C34E MOVEAUX       C354 RESETLC
 C360 SETROM        C361 MOVEC2M       C367 MOVESTRT      C367 MOVELOOP
 C36A NOT1          C371 NEXTA1        C371 NOREAD        C37B ROMOK
 C37C GETALT        C37F CO1           C38A C03           C393 MOVERET
 C393 GETALT1       C397 XFER          C398 GETALT2       C399 UPSHIFT0
 C39B UPSHIFT       C3A5 X.UPSHIFT     C3A6 GETCOUT       C3AA XFERC2M
 C3B0 XFERZP        C3B3 STORY         C3B8 STORCH        C3C0 XFERAZP
 C3C1 STORE         C3C3 JMPDEST       C3C6 TSTMEM        C3D0 MEM1
 C3D8 MEM2          C3DB STORE1        C3EE STORE2      ?C3F2 STORE3
```

```
C3F3 MEM3          C3F5 MEM4          ?C3F7 STORE4       C3F9 STORE5
C3FA MEM5          C405 MEM6          C40E BOOT4         C412 MEM7
C428 BTOK4         C42A MEM8          C42C MEM9          C431 MEMA
C439 BTOK4.1       C43D BTOK4.2       C440 MEMB          C448 BTOK4.3
C44E ENTRY4        C44F MEMC          C454 ENT4          C456 MEMD
C462 DOS24         C464 ZSAVE4        C46C MEMF          C46E DOIT4
C471 DONE4         C472 MEMERROR      C473 BADBITS       C473 RSLOOP4
C47D BBITS1        C481 CLRSTS        C484 RATS4         C489 PCCMD4
C48D PCBAD4        C48F PCERR4        C491 CLRS          C494 PCONV4
C4A8 PCSVZP4       C4A9 BADMAIN       C4B0 BADPRIM       C4B2 PCGTP4
?C4B5 PCSKP4       C4BB BBITS2        C4BD PCPARMS4      C4C8 HANGX
C4CA BADSWTCH      C4CE BSWTCH1       C4D1 DOSENT4       C4D8 BSWTCH2
C4E0 DOSSLT4       C4E4 BSWTCH2A      C4E7 BSWTCH3       C4E9 DOSOK4
C4ED HANGY         C4EF SWCHTST       C4F1 SWTST1        C4F3 SWTST2
C4FE SWTST3        C504 CLICK         C508 SWTST4        C51A SWTST5
C521 SWTST6        C533 SWTST7        C537 SWERR         C53F BIGLOOP
?C543 BLP2         C547 BLP3          C556 BLP4          C566 DQUIT
C56B SUC2          C580 ATALK         C580 SL.PREAD      C58E MAKTBL
C592 TBLLOOP       C5A0 TBLLOOP2      C5AA NOPATRN       C5AC PRMAIN
C5B4 GETUP         C5B5 PRLOOP        C5BB APPLE2C       C5C4 SHOWINST
C5C9 PRLAST        C5CF XMBASIC       C5D3 PRLOOP2       C5D9 PRODD
C5DC XMBOUT        C5E3 PRDONE        C5EA MBBAD         C5EC PRMAIN2
C5EE PRBAD         C5EE GOBASICIN     C5F3 PRBADZ        C5F5 BOOTFAIL
C5F7 SL.PWRITE     C5F8 PCNVRST       ?C60B DRV2ENT      C61F SEEKZERO
C623 PWMAIN        C62C PWLOOP        C63D EXTENT1       C63F RDADR
C640 PWLAST        C641 RETRY1        C642 RDDHDR        ?C648 FUGIT
C64A PWLOOP2       C650 PWODD         C656 RDHD0         C657 RETRY
C65A PWDONE        ?C65C EXTENT       C65E RDHD1         C663 PWMAIN2
C663 ISMRK1        C667 RDHD2         C667 STATTBL       C671 RDHD3
C680 PARMTBL       C683 RDSECT        C685 RDSEC1        C687 RDSEC2
C68F RDSEC3        C69A CMDTBL        C6A2 BADRD1        C6A6 RDATA
C6A8 RDAT0         C6AA RDAT1         C6B4 SWSL.BT       C6BA RDAT2
C6BC RDAT3         C6C2 SW.SETMOU     C6CB RDAT4         C6CD SW.MTSTINT
C6D3 BADREAD       C6D7 DENIBL        C6D8 SW.MREAD      C6D9 DENIB1
C6E3 SW.MCLEAR     C6EE SW.MCLAMP     C6F9 SW.MHOME      C700 MBASIC
C702 PNULL         C704 SW.INITMOUSE  C705 INENT         C707 OUTENT
C71A NOERROR       C71C XSETMOU       C722 XMTSTINT      C728 XMREAD
C72E XMCLEAR       C734 XMCLAMP       C73A XMHOME        C740 INITMOUSE
C746 M..OVEIRQ     C74C SLBOOT        C752 SLXEQ         C780 SWRTI
?C780 SWRTI2       C784 SWRTS         C784 SWRTS2        C787 SWRTSOP
?C788 SWRESET      C788 SWRESET2      C78E SWIRQ2        C797 SWSTHK2
?C797 SWPCNV       C79D SWBASICIN     C79D SWZZQT2       C7A3 SWSTTM
C7A9 SWCMD         C7AF SWAUX         C7B5 SWXFER        C7BB SWMINT
C7C1 BANGER        ?C7C7 SWATALK      C7CD SWSER3        C7D3 SWGETST
C7D9 SWREAD        C7DF SWGETB        C7E5 SWZZNM        C7EB SWXFGO2
?C7EB SWXFGO       C7F1 SWSTTM3       C7F1 SWSTHK3       C7F6 SWZZQT3
C803 NEWIRQ        C804 IRQENT        C806 SWCMD3        C80E FIXLC
C816 GETLC         C826 GLCBNK1       C826 IRQ2          C829 GLCDONE
C82A NTBL          C82A SETV          C82A IRQ21         C82F SWTBL0
C834 IRQ3          C83E IRQ4          C841 SWTBL1        C848 IRQ5
C850 PASSKIP1      C853 RSWTBL        C85B IRQ6          C85E IRQ7
C866 RMESS         C86C SMESS         C870 IRQ8          C875 SUCCESS
C87F IRQDONE       C880 PCNV          C882 IRQLCOK       C88C IRQDN1
C88E IRQDN2        C896 IRQDN3        C89C IRQDN4        C8A4 IRQDN5
C8A7 GOBREAK       C8C1 GBNOC         C8C7 GBNOTROM      C8C9 GBBRK
C8CC XRKBD1        C8D4 XNOKEY        C8D5 XRDKBD        C8E6 XBITKBD
C8F9 XBKB1         C8FB XBKB2         C900 MPADDLE       C90D PDON
C918 PDOK          C91D AMOD1         C93A AMOD2         C93B AMOD4
```



```
C93C AMOD3         C94A AMOD5         C94F AMOD6         C955 REL
C961 REL1          C96B REL2          C96E GOERR         C970 MOVINST
C972 MOV1          ?C983 DISLIN       C986 GETINST1      C98F GETOP
C9AD P1SKIP        C9BD NXTOP         C9C7 MINIERR       C9C9 ERR2
?C9CB ERR3         C9D8 DOINST        C9E7 GETI1         C9EC GOERR2
C9F4 DOLIN         C9F8 NXTCH         CA06 NXTMN         CA29 AMOD7
CA38 AMOD8         CA3B NNBL          CA43 STEP          CA4A XQWAIT
CA50 XQNOBT0       CA64 XQINIT        CA90 XQNTBRA       CA98 XQ1
CA9A XQ2           CAA6 XBRK          CAAC XRTI          CAB0 XRTS
CAB4 PCINC2        CAB6 PCINC3        CAC0 XJSR          CAC8 XJMP
CAC9 XJMPAT        CAD1 NEWPCL        ?CAD5 RTNJMP       CAD9 RTNJMP2
CAE3 XJMPATX       CAEE XJXNOC        CAF1 BRANCH        CAFF NBRNCH
CB05 INITBL        CB0D GODSP         CB22 GODREG        CB25 GODDONE
CB30 SCROLLDN      CB35 SCROLLUP      CB38 SCROLLIT      CB4E CHKRT
CB57 GETST         CB67 SETDBAS       CB6D SCRLIN        CB83 SETUP2
CB88 SETSRC        CB9B SCRLEVEN      CBA2 SCRLFT        CBA8 SKPLFT
CBB0 SCRLODD       CBB4 SKPRT         CBB9 SCRL3         CBC1 SEV1
CBC2 DOCLR         CBC7 CLR40         CBCF CLRHALF       CBDA CLR80
CBEE CLR0          CBF1 CLR2          CBFC CLR1          CC02 CLR3
CC04 CLRPORT       CC0B PASINVERT     CC12 INVERT        CC1C INVX
CC1D PICKY         CC33 PICK2         CC3D PICK1         CC3F PICK3
CC4A PICK4         CC4C SHOWCUR       CC53 NOTINV        ?CC68 NOTINV1
CC6B NOTINV2       CC70 UPDATE        CC93 UD2           CC99 CLRKBD
CC9D GETCUR        CCA7 GETCUR1       CCAD GETCUR2       CCB7 GETCUR3
CCBF GETCURX       CCC0 ESC3          CCCC NEWESC        CCD7 ESC0
?CCE3 ESC1         CCE5 ESC2          CCED ESCRDKEY      CCF8 ESCTAB
CD0C ESCCHAR       CD15 CTLTAB        CD2A CTLADR        CD54 CTLCHAR0
CD58 CTLCHAR       CD67 FNDCTL        CD6F CTLDONE       CD71 CTLGO
?CD7D CGO          CD80 CTLGO1        CD89 X.CUR.ON      CD8D X.CUR.OFF
CD91 CTLOFF        CD95 CTLON         CD99 MOUSON        CD9B CLRIT
CD9F MOUSOFF       CDA1 SETIT         CDA5 HOMECUR       CDB0 X.SO
CDB7 X.SI          CDBE SET80         CDC0 SET40         CDCD CHK80
CDD2 WIN40         CDD4 WIN80         CDD5 WIN0          CDE0 WIN1
CDED WIN2          CDF2 WIN3          CE02 WIN4          CE0A WNDREST
CE18 WIN5          CE1A SETX          CE1B HOOKITUP      CE20 HOOKUP
CE23 SETHOOKS      CE31 VIDMODE       CE3B PVMODE        CE44 QX
CE45 QUIT          CE4D ZZQUIT        CE53 SCRN84        CE58 SCR1
CE5E SCR2          CE66 SCR3          CE79 SCR4          CE80 SCRN48
CE82 SCR5          CE8B SCR6          ?CE8D SCR8         CE96 SCR7
CEAD SCR9          CEB1 PSTATUS       CEBC PIORDY        CEBE PSTERR
CEC0 PNOTRDY       CEC2 PWRITE        CEDD PWR1          CEF1 PWRITERET
CEF4 PWRET         CEF7 PRET          CEFA GETX          ?CF06 GETY
CF19 PCTL          CF29 STARTXY       CF30 PSETX         CF35 PASREAD
CF38 GKEY          CF41 PINIT         CF51 PSETUP        CF54 PSETUP2
CF66 PS1           CF71 PASCALC       ?CF7F PASCLC2      CF86 IRQTBLE
CF8C COMTBL        CF94 RTBL          CF9A M.OVEIRQ      CFA0 MOVEIRQ
CFA9 MIRQLP        CFC0 MIRQSTD       CFC3 CLRKBD2       CFCB LOOKASC
CFDE LADIG         CFE1 LACR          CFE4 LADONE        ?CFFF CLRROM
D000 COMMAND       ?D011 COMMAND1     D01F NOCMD         D020 NOCMD2
D022 INCMD         D032 INCMD2        D03C INCMD3        ?D068 CMD2NULL
D077 FLAGIT        D07F ONELETTER     D084 INCMD1        D087 CMD2LOOP
D08F BACKTO1       D091 CMDLOOP       D0A2 CMDZ2         D0A5 CKDIG
D0AD DIGLOOP       D0B5 COMINIT1      D0BF COMINIT       D0C5 ENABLE
D0C7 DISABLE       D0DB XREADY        D0DE CDONE         D0EA CMSET
D0F4 CMD2L         D0FD CMD2FOUND     D10A CMFOUND       D112 CMD.C
D11B CMD.C1        D126 CMDZ          D12F CMDN          D12F CMDCR
D139 CMDK          D139 CMDI          D139 CMDL          D13A CMDI2
D148 CDONE2        D14B CMDP          D14C CMDD          D150 CMDB
```

```
 D166 NOSHIFT      D16C CMDS         D177 MSWAIT       D179 MSLOOP
 D183 CMDP2        D188 CMDR         D197 CMDQ         D199 CMDT
 D1A0 SETTERM      D1B9 STCLR        D1C0 STSET        D1C9 STWASOK
 D1D8 CMDT2        D1EA CMDT3       ?D1EE FLUSH        D1F4 STRTS
 D1F5 CMDTABLE     D202 MASK1        D20D MASK2        D218 CMDLIST
 D225 CMD2LIST     D22A R.GETALT     D241 R.GETALT1    D246 R.GETALT2
 D247 DEFIDX2      D400 BASICIN     ?D43B PUTINBUF     D441 HEXTODEC
 D452 HEXDEC2      D45E HDPOS2       D464 HDLOOP       D469 DV10LOOP
 D470 DV10LT       D489 HDDONE       D48E DIAGS        D497 TSTZPG
 D49B ZP1          D4A4 ZP2          D4B7 ZP3          D4BC ZPERROR
 D4C2 TSTMEM2      D4C5 ZZNM         D4CE ZZLOOP       D4DA QTBL
 D600 I.NITMOUSE   D607 XRLOOP       D621 X.SETMOU     D636 XSOFF
?D638 SETIOU       D642 SILOOP       D649 SINOCH       D650 SMINVALID
 D651 X.MHOME      D655 XMHLOOP      D657 XMH2         D668 X.MCLEAR
 D674 XMCDONE      D679 X.MREAD      D690 XRBUT        D697 XRBUT2
 D69C XMRD2        D6A3 X.MCLAMP     D6C2 X.MTSTINT    D6CC NOSTAT2
 D800 EXECUTE      D81F EXEC2        D824 EXEC3        D830 PZCMD
 D834 PZCNT        D836 PZCNT2       D839 IORTS        D83A PREAD2.Z
 D83D PWRITE2      D840 DOSCONV2     D843 XDIAG        D846 XSTATUS
 D853 PSTAT0       D860 ST0LP        D86A PCNTL        D86E STBAD
 D873 PCNTOK       D874 SL.PSTATUS   D880 PST0        D88A PSTMOV
 D89E XREAD        D8A2 XWRITE      ?D8A3 XRWCMN       D8B9 PRDBLK
 D8BD PWRBLK       D8BE XREAD2       D8DE PRDREAD      D8E9 PRBAD2
 D8EC PRBAD3       D8EF BOOT.SL      D907 BTNODOS      D913 BTMV
 D922 BTFAIL       D923 BTCMD        D927 DOSPATCH     D94E DCRTS
 D94F DOSCONV      D957 DCERR        D95A DC1          D995 TESTSIZE
 D9A0 TSLOOP       D9CB TSNORAM      D9D3 SL.FORMAT   ?D9E2 FMNOSP
 D9F9 FMPMAP1      D9FB FMPMAP2      DA08 FMTDONE      DA09 FMPAS
 DA0F FMDOS        DA24 FMDBMAP      DA30 FMDOK        DA39 MAKECAT
 DA44 MCBOOT       DA53 MCBYTE       DA68 MCNTSZ       DA71 MCNTNM
 DA77 MC0          DA7D MCFE         DA8B MCADD2       DA8E MCADD
 DAA6 MCDONE       DAA7 CATTBL       DB30 XDIAGZ       DB32 XDLOOP
 DC00 DIAGCODE     E000 BASIC        E003 BASIC2       F800 F8ORG
 F800 PLOT         F80C RTMASK       F80E PLOT1       ?F819 HLINE
 F81C HLINE1       F826 VLINEZ       F828 VLINE        F831 RTS1
?F832 CLRSCR       F836 CLRTOP       F838 CLRSC2       F83C CLRSC3
 F847 GBASCALC     F856 GBCALC      ?F85F NXTCOL      ?F864 SETCOL
?F871 SCRN         F879 SCRN2        F87F RTMSKZ       F882 INSDS1
 F88E INSDS2       F897 IEVEN        F8A1 ERR          F8A5 GETFMT
 F8BE MNNDX1       F8C2 MNNDX2       F8C9 MNNDX3       F8CC GOTONE
 F8D0 INSTDSP      F8D4 PRNTOP       F8DB PRNTBL       F8F5 PRMN1
 F8F9 PRMN2        F910 PRADR1       F914 PRADR2       F926 PRADR3
 F92A PRADR4       F930 PRADR5      ?F938 RELADR       F940 PRNTYX
?F941 PRNTAX      ?F944 PRNTX        F948 PRBLNK       F94A PRBL2
?F94C PRBL3        F953 PCADJ        F954 PCADJ2       F956 PCADJ3
 F95C PCADJ4       F961 RTS2         F962 FMT1         F9A6 FMT2
 F9B4 CHAR2        F9BA CHAR1        F9C0 MNEML        FA00 MNEMR
?FA40 IRQ          FA47 NEWBRK      ?FA4C BREAK        FA59 OLDBRK
 FA62 RESET       ?FA81 NEWMON       FA85 BEEPSKIP    ?FA9B FIXSEV
 FAA3 NOFIX        FAA6 PWRUP       ?FAA9 SETPG3       FAAB SETPLP
?FABA AUTOSCAN     FABD RESET.X      FAD7 REGDSP       FADA RGDSP1
 FAE4 RDSP1        FAFD PWRCON       FB02 RGDSP2      ?FB09 TITLE
 FB12 PWRUP2      ?FB1E PREAD        FB25 PREAD2       FB2E RTS2D
 FB2F INIT        ?FB39 SETTXT      ?FB40 SETGR        FB4B SETWND
 FB54 DOCTL        FB59 VTAB23      ?FB5B TABV         FB60 APPLEII
 FB65 STITLE      ?FB6F SETPWRC      FB78 VIDWAIT      FB88 KBDWAIT
 FB94 NOWAIT       FBA0 NEWADV       FBB0 NEWADV1      FBB3 F8VERSION
 FBB4 DOCOUT1      FBBC DCX          FBC1 BASCALC      FBD0 BASCLC2
```



```
 FBD9 CHKBELL      ?FBDD BELL1       FBE4 BELL2        FBEF RTS2B
 FBF0 STORADV      ?FBF4 ADVANCE     FBF8 ADV2         FBFC RTS3
 FBFD VIDOUT        FC04 VIDOUT1    ?FC10 BS           FC1A UP
 FC22 VTAB          FC24 VTABZ        FC30 VTAB40      ?FC34 RTS4
 FC35 NEWOPS        FC38 NEWOP1       FC42 CLREOP       FC44 CLREOP2
 FC46 CLEOP1        FC58 HOME         FC5D CLREOP1      FC62 CR
 FC66 LF           ?FC70 SCROLL       FC73 NEWCR        FC80 GETINDX
 FC85 CRRTS         FC86 NEWVTAB      FC88 NEWVTABZ     FC8D NEWCLREOL
 FC90 NEWCLEOLZ     FC99 NEWC1        FC9C CLREOL       FC9E CLEOLZ
 FCA0 CLRLIN        FCA4 CTLDO        FCA8 WAIT         FCA9 WAIT2
 FCAA WAIT3        ?FCB3 RTS6         FCB4 NXTA4        FCBA NXTA1
 FCC8 RTS4B        ?FCC9 HEADR        FCCA COLDSTART    FCD0 BLAST
 FCE6 COM1          FCF5 COM2         FCFB COM3         FD0C RDKEY
?FD18 KEYIN0        FD1B KEYIN        FD20 DONXTCUR    ?FD25 GOTKEY
?FD35 RDCHAR        FD38 LOOKPICK     FD44 NOESCAPE    ?FD45 NOESC1
 FD4A NOESC2        FD4D NOTCR        FD5F NOTCR1       FD62 CANCEL
 FD67 GETLNZ       ?FD6A GETLN       ?FD6F GETLN1      FD71 BCKSPC
 FD75 NXTCHAR       FD84 ADDINP      ?FD8B CROUT1       FD8E CROUT
 FD92 PRA1          FD96 PRYX2        FDA3 XAM8         FDAD MOD8CHK
 FDB3 XAM           FDB6 DATAOUT     ?FDC5 RTS4C       FDC6 XAMPM
 FDD1 ADD           FDDA PRBYTE      ?FDE3 PRHEX       FDE5 PRHEXZ
 FDED COUT          FDF0 COUT1        FDF6 COUTZ        FE00 BL1
 FE04 BLANK        ?FE0B STOR         FE17 RTS5         FE18 SETMODE
 FE1D SETMDZ        FE20 LT           FE22 LT2          FE2C MOVE
 FE36 VERIFY        FE58 VFYOK        FE5E LIST         FE63 LIST2
 FE6C MINI          FE6F TRACE        FE71 STEPZ        FE75 A1PC
 FE78 A1PCLP        FE7F A1PCRTS      FE80 SETINV       FE84 SETNORM
 FE86 SETIFLG       FE89 SETKBD      ?FE8B INPORT       FE8D INPRT
 FE93 SETVID       ?FE95 OUTPORT      FE97 OUTPRT       FE9B IOPRT
 FEA7 NOTPRT0       FEAB IOPRT2       FEB0 XBASIC       FEB3 BASCONT
 FEB6 GO            FEBF REGZ         FEC2 OPRT0        FECA USR
?FECD WRITE         FECE DOPR0        FEDE IOPRT1       FEE2 DECCH
 FEE9 CLRCH         FEEB WDTHCH       FEEC SETCUR       FEEE SETCUR1
 FEF6 CRMON        ?FEFD READ         FEFE OPTBL        FF15 INDX
?FF2D PRERR         FF3A BELL         FF3F RESTORE     ?FF44 RESTR1
 FF4A SAVE         ?FF4C SAV1        ?FF59 OLDRST      ?FF59 MONITOR
 FF65 MON           FF69 MONZ         FF73 NXTITM       FF7A CHRSRCH
 FF8A DIG           FF90 NXTBIT       FF98 NXTBAS       FFA2 NXTBS2
 FFA7 GETNUM        FFAD NXTCHR       FFBE TOSUB        FFC7 ZMODE
 FFCC CHRTBL        FFE3 SUBTBL       FFFE IRQVECT      FFFF PROCAT
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 15-JAN-84 21:28
** TOTAL LINES ASSEMBLED  6885
** FREE SPACE PAGE COUNT   29
```

```
SOURCE   FILE #01 =>PC
 INCLUDE FILE #02 =>PC.EQUATES
 INCLUDE FILE #03 =>PC.BOOTSPACE
 INCLUDE FILE #04 =>PC.BOOT
 INCLUDE FILE #05 =>PC.PACKET
 INCLUDE FILE #06 =>PC.CREAD
 INCLUDE FILE #07 =>PC.MAIN
0000:        0101    1 version   equ   $101          ;v1.0.1
0000:        0101    2           X6502
0000:                3           MSB   ON

0000:                4           lst   on,vsym,asym
```



```
0000:                7 ; PPPP  RRRR   OOO  TTTTT  OOO   CCC   OOO  L
0000:                8 ; P   P R   R O   O   T   O   O C   C O   O L
0000:                9 ; PPPP  RRRR  O   O   T   O   O C     O   O L
0000:               10 ; P     R R   O   O   T   O   O C   C O   O L
0000:               11 ; P     R  R   OOO    T    OOO   CCC   OOO  LLLLL
0000:               12 ;
0000:               13 ;  CCC   OOO  N   N V   V EEEEE RRRR  TTTTT EEEEE RRRR
0000:               14 ; C   C O   O NN  N V   V E     R   R   T   E     R   R
0000:               15 ; C     O   O N N N V   V EEEE  RRRR    T   EEEE  RRRR
0000:               16 ; C   C O   O N  NN  V V  E     R R     T   E     R R
0000:               17 ;  CCC   OOO  N   N   V   EEEEE R  R    T   EEEEE R  R


0000:               19 ************************************************
0000:               20 *
0000:               21 * UniDisk 3.5 Driver Firmware   Version 1.0.1
0000:               22 *
0000:               23 * Written by Michael Askins   May 15, 1985
0000:               24 * Revised by M. Askins and R. Chiang   April 10, 1986
0000:               25 *
0000:               26 * Copyright Apple Computer, Inc. 1985,1986
0000:               27 * All Rights Reserved
0000:               28 *
0000:               29 ************************************************
```

```
01 PC          Protocol Converter Code for Tiger   20-OCT-86  06:29 PAGE 3

0000:          ****************************************************************
0000:          *                                                              *
0000:          *  Modification History:                                       *
0000:          *                                                              *
0000:          * Rel  Date       Who  Action                                  *
0000:          * ------------------------------------------------------------ *
0000:          * ***  18 Dec 84  MSA  RELEASE VERSION 0.02 (Sony)             *
0000:          *      10 Jan 85  MSA  Added //c support:                      *
0000:          *                        General conditional assembly overhead *
0000:          *      16 Jan 85  MSA  Added retries and timeouts              *
0000:          *                      MSlot handled correctly                 *
0000:          *                      Finished Boot code                      *
0000:          *                      Altered ProDOS errors - add $27 catchall *
0000:          *      18 Jan 85  MSA  Remove call to WAIT in monitor          *
0000:          *                      Add Boot failure messages               *
0000:          *      22 Jan 85  MSA  Add IWM reconfigure for //c version     *
0000:          *      23 Jan 85  MSA  Move Comm routines to $C800 ($C900)     *
0000:          *                      Fixed zero page preservation            *
0000:          * ***  23 Jan 85  MSA  RELEASE VERSION 0.03 (Apple)            *
0000:          *      25 Jan 85  MSA  Swap slot dep read and boot code (//c)  *
0000:          *                      Add other //c differences...            *
0000:          *      30 Jan 85  MSA  Add auxtype byte                        *
0000:          *                      Fix comm error on receive packet        *
0000:          *                      Fix checksum to include MSBs of overhead *
0000:          *      07 Feb 85  MSA  Add COUT support on boot fail           *
0000:          * ***  08 Feb 85  MSA  RELEASE VERSION 1.00A (alpha)           *
0000:          *      22 Feb 85  MSA  Add bytecount in X,Y on PC calls        *
0000:          *                      Change hard reset time to 1 ms (was 83) *
0000:          *                      Crunched code by adding ClrPhases       *
0000:          *                      Add zeroing of third block byte (ProDOS) *
0000:          *      06 Mar 85  MSA  Fixed slot 7 goof (stack screw up)      *
0000:          *                      No clear phases on retries              *
0000:          *                      Hard reset time to 40 ms                *
0000:          *                      Pass #parms instead of unit# and no chk *
0000:          *                      Init code (all reset vs. comm reset)    *
0000:          *                      Add 2 bytes to pass a full 9 byte cmd   *
0000:          *      16 Mar 85  MSA  Fix bytecount on retries                *
0000:          *                      Boot block must be $800=$01, $801<>$00  *
0000:          *      17 Mar 85  MSA  Remove WRREQ while waiting for motor TO *
0000:          *                      Remove glitch on /ENBL2 in AssignID     *
0000:          *      20 Mar 85  MSA  Add interrupt on/off/poll support       *
0000:          *                      Reset pulse to 80 ms                    *
0000:          *                      //c delay of 100 ms on initial AssignID *
0000:          *                      ID bytes changed                        *
0000:          *                      Retransmit implemented (RecPack)        *
0000:          *                      Add send data packet retries (5)        *
0000:          *                      Rearrange PC stack adjust               *
0000:          *                      Add //c Appletalk vector                *
0000:          *      24 Mar 85  MSA  Add //c millisecond wait each call      *
0000:          * ***  25 Mar 85  MSA  RELEASE VERSION 1.00B (beta) (//e)      *
0000:          *      18 Apr 85  MSA  Clear decimal mode                      *
0000:          *                      Eight bytes are returned on stat unit#0 *
0000:          *                      Stat Unit#0 scode<>0 is rejected        *
0000:          *                      X and Y set to 0008 on status unit#0    *
0000:          *                      Enable interrupts done correctly        *
0000:          *                      Add unit#0 parameter count checking     *
0000:          * ***  22 Apr 85  MSA  RELEASE VERSION 1.01B                   *
0000:          * ***  15 May 85  MSA  RELEASE VERSION 1.0                     *
```

```
01 PC          Protocol Converter Code for Tiger   20-OCT-86  06:29 PAGE 4

0000:          *      10 Apr 86   RC  removed reference to AppleTalk          *
0000:          *      10 Apr 86   RC  forgot to add hi byte of auxptr in      *
0000:          *                      divide7 routine                         *
0000:          *      10 Apr 86   RC  returns write protect from old Lirons   *
0000:          * ***  10 Apr 86   RC  RELEASE VERSION 1.0.1                   *
0000:          *                  RC  v1.0.1 is to be used with Tiger only    *
0000:          *                                                              *
0000:          ****************************************************************


----- NEXT OBJECT FILE NAME IS PC.0
C500:        C500          org   $C500
C500:                      include pc.equates
```

```
C500:                    2 *
C500:       00BF         3 PDIDByte  equ   $BF          ;ProDOS attributes byte
C500:       0000         4 PCID2     equ   $0           ;This means a Liron card
C500:                    5 *
C500:                    6 ************************
C500:                    7 *                      *
C500:                    8 *  Zero Page  (temps)  *
C500:                    9 *                      *
C500:                   10 ************************
C500:                   11 *
0000:                   12          dsect
0000:       0040        13 zeropage  equ   $0040
0040:       0040        14          org   zeropage
0040:                   15 *
0040:00                 16 checksum  dfb   0
0041:00                 17 topbits   dfb   0
0042:00                 18 CMDCode   dfb   0           ;ProDOS parameter passing area
0043:       0043        19 CMDPCount equ   *
0043:00                 20 CMDUnit   dfb   0
0044:       0044        21 CMDBuffer equ   *
0044:00                 22 CMDBufferl dfb  0
0045:00                 23 CMDBufferh dfb  0
0046:       0046        24 CMDSCode  equ   *
0046:       0046        25 CMDBlock  equ   *
0046:00                 26 CMDBlockl dfb   0
0047:00                 27 CMDBlockh dfb   0
0048:00                 28 CMDBlocks dfb   0
0049:00                 29 CMDSpare1 dfb   0
004A:00                 30 CMDSpare2 dfb   0
004B:       004B        31 rcvbuf    equ   *
004B:00                 32 grp7ctr   dfb   0
004C:00                 33 oddbytes  dfb   0
004D:       004D        34 statbyte  equ   *
004D:       004D        35 bytecount equ   *
004D:       004D        36 bytecountl equ  *
004D:       004D        37 next      equ   *
004D:00                 38 next1     dfb   0
004E:       004E        39 AuxType   equ   *
004E:       004E        40 bytecounth equ  *
004E:00                 41 next2     dfb   0
004F:       004F        42 RPacketType equ *
004F:00                 43 next3     dfb   0
0050:       0050        44 DeviceID  equ   *
0050:00                 45 next4     dfb   0
0051:       0051        46 HostID    equ   *
0051:00                 47 next5     dfb   0
0052:       0052        48 pointer   equ   *
0052:00                 49 next6     dfb   0
0053:00                 50 next7     dfb   0
0054:00 00              51 buffer    dw    0
0056:       0056        52 auxptr    equ   *
0056:00 00              53 buffer2   dw    0
0058:00                 54 slot      dfb   0
0059:       0059        55 temp      equ   *
0059:00                 56 tbodd     dfb   0
005A:00                 57 Unit      dfb   0           ;Current target unit
005B:00                 58 WPacketType dfb 0
005C:                   59 *
```



```
005C:                   60 *
005C:       001C        61 ZPSize    equ   *-zeropage
005C:                   62 *
005C:                   63 *
C500:                   64          dend
C500:                   65 *
C500:       CFFF        66 ClearIOROMs equ $CFFF
C500:       0100        67 stack     equ   $100
C500:                   68 *
C500:                   69 *
C500:                   70 ************************
C500:                   71 *                      *
C500:                   72 *   Screenhole Storage *
C500:                   73 *                      *
C500:                   74 ************************
C500:                   75 *
C500:                   76 * The screenhole layout is as follows:
C500:                   77 *
C500:                   78 *                //e          //c
C500:                   79 *  ----------------------------------
C500:                   80 *  ProFlag      $478+n        $478
C500:                   81 *  Retry        $4F8+n        $4F8
C500:                   82 *  SHTempl      $578+n        $578
C500:                   83 *  SHTempX      $5F8+n        $5F8
C500:                   84 *  SHTempY      $678+n        $678
C500:                   85 *  Power1       $6F8+n        ----
C500:                   86 *  Power2       $778+n        ----
C500:                   87 *  NumDevices   $7F8+n        $6FE
C500:                   88 *  SvBcL        $6F8          $6F8
C500:                   89 *  SvBcH        $778          $778
C500:                   90 *
C500:       0473        91 scholes   equ   $473          ;Use the slot 0 sholes for temps
C500:                   92 *
C500:       0473        93 ProFlag   equ   scholes
C500:       04F3        94 Retry     equ   scholes+$80
C500:       0573        95 SHTempl   equ   scholes+$100
C500:       0573        96 Retry2    equ   SHTempl
C500:       05F3        97 SHTempX   equ   scholes+$180
C500:       0673        98 SHTempY   equ   scholes+$200
C500:       06F9        99 NumDevices equ  $6F9          ;Actually in slot 6
C500:                  100 *
C500:       06F8       101 SvBcL     equ   $6F8
C500:       0778       102 SvBcH     equ   $778
C500:                  103 *
C500:       0025       104 cv        equ   $25
C500:       0024       105 ch        equ   $24
C500:       FC22       106 vtab      equ   $FC22
C500:       FDED       107 cout      equ   $FDED
C500:       07DB       108 bootscrn  equ   $7DB
C500:       07F8       109 MSlot     equ   $7F8
C500:       FE93       110 setvid    equ   $FE93
C500:       FE89       111 setkbd    equ   $FE89
C500:       FABA       112 AutoScan  equ   $FABA
C500:       E000       113 Basic     equ   $E000
C500:       0000       114 loc0      equ   $0           ;Boot parms
C500:       0001       115 loc1      equ   $1           ;
C500:                  116 *
C500:       C797       117 SWPROTO   equ   $C797        ;//c bank switch to $C800
```

```
C500:       C784  118 SWRTS2    equ   $C784        ;RTS to bank 1
C500:             119 *
C500:             120 *
C500:             121 ************************
C500:             122 *                      *
C500:             123 *    General Equates   *
C500:             124 *                      *
C500:             125 ************************
C500:             126 *
C500:       00A5  127 PBBValue  equ   $A5         ;Powerup Byte Base Value
C500:       00FF  128 PBCValue  equ   $FF         ;Powerup Byte Complement Value
C500:             129 *
C500:       0000  130 PowerReset equ  $00
C500:       0080  131 CommReset equ   $80
C500:             132 *
C500:       0032  133 bsyto1    equ   50          ;(.55 ms) T/O on /BSY before send
C500:       000A  134 bsyto2    equ   10          ;(.12 ms) T/O on /BSY after send
C500:       001E  135 statmto   equ   30          ;30 bytes stat mark timeout
C500:       0009  136 cmdlength equ   9           ;Command packet length
C500:       00C3  137 packetbeg equ   $C3         ;Mark at beginning of packet
C500:       00C8  138 packetend equ   $C8         ;End of packet mark
C500:       0080  139 cmdmark   equ   $80         ;Command packet identifier
C500:       0081  140 statmark  equ   $81         ;Status Packet identifier
C500:       0082  141 datamark  equ   $82         ;Data Packet identifier
C500:             142 *
C500:       0007  143 iwmmode   equ   $07         ;No timer, asynch, latch
C500:             144 *
C500:       0000  145 SCDeviceStat equ 0          ;Get Device Specific Status
C500:       0001  146 SCGetDCB  equ   1           ;Get Dev Ctrl Block (modebits)
C500:       0002  147 SCRetNLStat equ 2           ;Return Newline Status
C500:       0003  148 SCGetDevInfo equ 3          ;Get Device Info Block
C500:             149 *
C500:       C080  150 iwm       equ   $C080
C500:             151 *
C500:       C080  152 reqclr    equ   iwm+0
C500:       C081  153 reqset    equ   iwm+1
C500:       C082  154 ca1clr    equ   iwm+2
C500:       C083  155 ca1set    equ   iwm+3
C500:       C084  156 ca2clr    equ   iwm+4
C500:       C085  157 ca2set    equ   iwm+5
C500:       C086  158 lstrbclr  equ   iwm+6
C500:       C087  159 lstrbset  equ   iwm+7
C500:       C088  160 monclr    equ   iwm+8
C500:       C089  161 monset    equ   iwm+9
C500:       C08A  162 enable1   equ   iwm+10
C500:       C08B  163 enable2   equ   iwm+11
C500:       C08C  164 16clr     equ   iwm+12
C500:       C08D  165 16set     equ   iwm+13
C500:       C08E  166 17clr     equ   iwm+14
C500:       C08F  167 17set     equ   iwm+15
C500:             168 *
C500:             169 *
C500:             170 * Error0 codes
C500:             171 *
C500:       0001  172 noanswer  equ   1
C500:       0002  173 nomark    equ   2
C500:       0004  174 wasreset  equ   4
C500:       0008  175 bytecmp   equ   8
```



```
C500:       0010  176 csumerr   equ   $10
C500:       0020  177 nopackend equ   $20
C500:       0040  178 bushog    equ   $40
C500:             179 *
C500:             180 * Command Codes
C500:             181 *
C500:       0000  182 StatusCmd equ   $00
C500:       0001  183 ReadCmd   equ   $01
C500:       0002  184 WriteCmd  equ   $02
C500:       0003  185 FormatCmd equ   $03
C500:       0004  186 ControlCmd equ  $04
C500:       0005  187 InitCmd   equ   $05
C500:             188 *
C500:             189 *
C500:       0040  190 Soft      equ   %01000000     ;The soft error bit in statbyte
C500:             191 *
C500:       0001  192 BadCmd    equ   $01
C500:       0004  193 BadPCnt   equ   $04
C500:       0006  194 BusErr    equ   $06
C500:       0011  195 BadUnit   equ   $11
C500:       001F  196 NoInt     equ   $1F
C500:       0021  197 BadCtl    equ   $21
C500:       0022  198 BadCtlParm equ  $22
C500:       0027  199 IOError   equ   $27
C500:       0028  200 NoDrive   equ   $28
C500:       002B  201 WriteProt equ   $2B
C500:       002D  202 BadBlock  equ   $2D
C500:       002F  203 OffLine   equ   $2F
C500:       0068  204 LastOne   equ   Soft+NoDrive
C500:       0067  205 SoftError equ   Soft+IOError
C500:             206 *
C500:       0010  207 SVMask1   equ   $10
C500:             208 *
C500:       0BB8  209 RC1       equ   3000          ;Send a command pack 3000 times (3 sec)
C500:       0005  210 RC2       equ   5             ;Data Packs get tried only 5 times
C500:             211 *
C500:             212 *
C500:             100           include pc.bootspace
```

```
C500:        0060    2 TheOff    equ   $60           ;On //c IWM in slot 6
C500:                3 *
C500:                4 * Here beginneth that code which resideth in the boot space
C500:                5 *  at the time the card resteth in slot the fifth.
C500:                6 *
C500:        C500    7 C500org   equ   *
C500:                8 *
C500:                9 * Auto Boot signature bytes
C500:               10 *  This is also the boot (auto & PR#5) entry point.
C500:               11 *
C500:A2 20          12           ldx   #$20
C502:A2 00          13           ldx   #$00
C504:A2 03          14           ldx   #$03
C506:               15 *
C506:C9 00          16           cmp   #0            ;Flag that this is a boot
C508:B0 17   C521   17           bcs   BootC
C50A:               18 *
C50A:               19 * Here is the ProDOS normal entry point
C50A:               20 *
C50A:        C50A   21 ProDOSEntry equ *
C50A:               22 *
C50A:               23 * Set up so that ProFLAG will have the top bit set
C50A:               24 *
C50A:38             25           sec
C50B:B0 01   C50E   26           bcs   *+3           ;Skip the clear
C50D:               27 *
C50D:               28 * This is the MLIxface entry point
C50D:               29 *
C50D:        C50D   30 MLIEntry  equ   *             ;Only use this label in //c version
C50D:18             31           clc
C50E:A2 05          32           ldx   #$05
C510:7E 73 04       33           ror   ProFLAG,x     ;ProFLAG[7]=1 if ProDOS, =0 if MLI
C513:18             34           clc                 ;This is not a boot entry
C514:               35 *
C514:               36 * Now save mslot and clear all $C800 ROMs
C514:               37 *
C514:        C514   38 bootcase5 equ   *
C514:A2 C5          39           ldx   #$C5          ;Load value for MSLOT
C516:8E F8 07       40           stx   MSlot
C519:A2 05          41           ldx   #$05
C51B:AD FF CF       42           lda   ClearIOROMs   ;Clear all $C800 latches but ours
C51E:               43 *
C51E:4C 97 C7       44           jmp   SWPROTO
C521:        C521   45 BootC     equ   *
C521:A2 05          46           ldx   #$05          ;Need slot number
C523:              101           include pc.boot
```



```
C523:                2 *
C523:        C523    3 Bootcode  equ   *
C523:86 58           4           stx   slot
C525:                5 *
C525:A9 C5           6           lda   #$C5
C527:8D F8 07        7           sta   MSlot
C52A:20 76 C5        8           jsr   reset
C52D:                9 *
C52D:A0 05          10           ldy   #5            ;Copy a command table
C52F:B9 70 C5       11 bc1       lda   boottab,y
C532:99 42 00       12           sta   cmdcode,y
C535:88             13           dey
C536:10 F7   C52F   14           bpl   bc1
C538:               15 *
C538:               16 * Now do the read from block zero
C538:               17 *
C538:20 0A C5       18           jsr   ProDOSEntry
C53B:B0 15   C552   19           bcs   bootfail      ;If fail, check loc
C53D:               20 *
C53D:AE 00 08       21           ldx   $800          ;If ($800)<>1 this is no A// boot disk
C540:CA             22           dex
C541:D0 0F   C552   23           bne   bootfail
C543:               24 *
C543:AE 01 08       25           ldx   $801          ;If $801 is zero, no boot
C546:F0 0A   C552   26           beq   bootfail
C548:               27 *
C548:               28 * It all looks okay.  Jump to the code with N0 in X.
C548:               29 *
C548:A5 58          30           lda   Slot
C54A:0A             31           asl   a
C54B:0A             32           asl   a
C54C:0A             33           asl   a
C54D:0A             34           asl   a
C54E:AA             35           tax
C54F:4C 01 08       36           jmp   $801          ;Jump to it
C552:               37 *
C552:               38 * Do this code if the boot can't be done.
C552:               39 *  If this was an autoboot (loc=$CN00), continue the slot scan.
C552:               40 *  If not, drop into basic after issuing appropriate message
C552:               41 *
C552:A2 10          42 bootfail ldx   #>bmsglen-1

C554:BD 5F C5       44 morchrs  lda   bootmsg,x
C557:9D DB 07       45          sta   bootscrn,x
C55A:CA             46          dex
C55B:10 F7   C554   47          bpl   morchrs
C55D:80 FE   C55D   48 coma     bra   coma          ;He's dead Jim.

C55F:C3 E8 E5 E3    50 bootmsg  asc   'Check        Disk Drive.'
C570:        0011   51 bmsglen  equ   *-bootmsg
C570:               52 *
C570:01 50 00 08    53 boottab  dfb   ReadCMD,$50,0,8,0,0 ;Read from 1st; blk0->$801
C576:               54 *
C576:               55 *
C576:               56 * this routine is called from the //c reset code.  it forces a
C576:               57 * reset of the PC Bus.  location 0 and 1 are being used by the
C576:               58 * autostart people.
```

```
04 PC.BOOT          Service Boot Request                20-OCT-86  06:29 PAGE 11

C576:        C576   60 Reset     equ   *
C576:A2 08          61           ldx   #8
C578:        C578   62 rst1      equ   *
C578:BD 83 C5       63           lda   rcode,x
C57B:95 02          64           sta   loc0+2,x
C57D:CA             65           dex
C57E:10 F8   C578   66           bpl   rst1
C580:4C 02 00       67           jmp   loc0+2

C583:        C583   69 rcode     equ   *
C583:20 0D C5       70           jsr   MLIEntry
C586:05             71           dfb   InitCMD
C587:09 00          72           dw    $0009
C589:60             73           rts

C58A:01 00          75 cmdlist   dfb   1,0           ;One parm - the unit $00


----- NEXT OBJECT FILE NAME IS PC.1
C5F5:        C5F5  103           org   $C5F5
C5F5:4C 52 C5      104           jmp   bootfail      ;Jump to the boot failure message
C5F8:4C 76 C5      105           jmp   reset         ;Reset vector
C5FB:00            106           dfb   PCID2
C5FC:00 00         107           dw    0
C5FE:BF            108           dfb   PDIDByte
C5FF:0A            109           dfb   >ProDOSEntry

----- NEXT OBJECT FILE NAME IS PC.2
C880:        C880  111           org   $C880
C880:4C 4C CD      112           jmp   Entry         ;The //c bank switch jumps here

C883:              114           include pc.packet
C883:                   1        lst   cyc
C883:                   2 *
```

```
05 PC.PACKET        Send a CBus Packet                  20-OCT-86  06:29 PAGE 12

C883:                   4 ******************************************************
C883:                   5 *                                                    *
C883:                   6 *  SendOnePack                  Send a CBus Packet   *
C883:                   7 *                                                    *
C883:                   8 *    This routine sends a packet of data across the  *
C883:                   9 *  bus.  The protocol is as follows:                 *
C883:                  10 *                                                    *
C883:                  11 *  REQ ______|2                          5|________  *
C883:                  12 *                                                    *
C883:                  13 * /BSY ___|1                 3              4|________  *
C883:                  14 *                                                    *
C883:                  15 *    1)  Device signals ready for data               *
C883:                  16 *    2)  Host signals data iminent                   *
C883:                  17 *    3)  Packet is transmitted (sync, command mark,  *
C883:                  18 *         ids, contents, checksum [msb=1])           *
C883:                  19 *    4)  Device signals packet recieved              *
C883:                  20 *    5)  Host finishes send data cycle               *
C883:                  21 *                                                    *
C883:                  22 *    The bytes are sent in slow mode (32 cycles/byte) *
C883:                  23 *  and the timing is critical.  Branches which should *
C883:                  24 *  not cross page boundaries are marked.             *
C883:                  25 *                                                    *
C883:                  26 *   Input:  buffer (2 bytes) <- ptr to data to send  *
C883:                  27 *           bytecount (2)    <- length (bytes) of data *
C883:                  28 *           packettype (1)   <- command or data packet *
C883:                  29 *           CMDUnit (1)      <- # of device to receive *
C883:                  30 *                                                    *
C883:                  31 *   Output: carry set- handshake error               *
C883:                  32 *                 clr- bytes sent                    *
C883:                  33 *                                                    *
C883:                  34 ******************************************************
C883:                  35 *
C883:        C883      36 SendOnePack equ *
C883:                  37 *
C883:                  38 * Prep for the transmission
C883:                  39 *
C883:20 61 CB    (6)   40        jsr   WritePrep     ;Does a bunch of stuff
C886:                  41 *
C886:                  42 * Enable PC chain.
C886:                  43 *
C886:20 7D CA    (6)   44        jsr   enablechain   ;This sets X reg
C889:A0 07       (2)   45        ldy   #iwmmode      ;This is the mode value
C88B:20 20 CC    (6)   46        jsr   SetIWMode     ;Don't mess unless we gotta
C88E:                  47 *
C88E:                  48 * Turn on the IWM
C88E:                  49 *
C88E:BD 8B C0    (4)   50        lda   enable2,x     ;Don't disturb //c internal drive
C891:BD 89 C0    (4)   51        lda   monset,x
C894:                  52 *
C894:                  53 * Loop until the chain becomes unbusy
C894:                  54 *
C894:A0 32       (2)   55        ldy   #bsyto1       ;Each loop is 11 microseconds
C896:BD 8E C0    (4)   56 ubsy1  lda   l7clr,x       ;Test if /BSY is hi or lo
C899:30 07   C8A2(3)   57        bmi   chainunbsy    ;If hi, bus is not busy
C89B:88          (2)   58        dey
C89C:D0 F8   C896(3)   59        bne   ubsy1         ;Keep trying
C89E:                  60 *
C89E:38          (2)   61        sec
```

```
C89F:4C CC C9    (3)   62        jmp   sd10
C8A2:                  63 *
C8A2:                  64 * Tell the bus that data is coming and send the sync bytes
C8A2:                  65 *  Sync is groups of eight 2's separated by a 6 (micS cell)
C8A2:                  66 *  (1111111100111111111001111111100 ...)
C8A2:                  67 *
C8A2:        C8A2      68 chainunbsy equ *
C8A2:BD 81 C0    (4)   69        lda   reqset,x      ;Raise REQ
C8A5:                  70 *
C8A5:A0 05       (2)   71        ldy   #5            ;Sync plus packet begin
C8A7:                  72 *
C8A7:A9 FF       (2)   73        lda   #$FF          ;Send out the 1st byte sync
C8A9:9D 8F C0    (5)   74        sta   17set,x
C8AC:                  75 *
C8AC:B9 D3 C9    (4)   76 ssb    lda   preamble,y
C8AF:                  77 *
C8AF:                  78 *
C8AF:1E 8C C0    (7)   79 ssd    asl   16clr,x       ;Wait 'til buffer empty
C8B2:90 FB   C8AF(3)   80        bcc   ssd
C8B4:                  81 *
C8B4:9D 8D C0    (5)   82        sta   16set,x
C8B7:88          (2)   83        dey
C8B8:10 F2   C8AC(3)   84        bpl   ssb           ;Back for more bytes
C8BA:                  85 *
C8BA:                  86 * Send over the desination ID
C8BA:                  87 *
C8BA:A5 5A       (3)   88        lda   Unit
C8BC:09 80       (2)   89        ora   #$80          ;Make the device ID
C8BE:20 50 CA    (6)   90        jsr   sendbyte
C8C1:                  91 *
C8C1:                  92 * Send the source ID (that's us... we're an $80)
C8C1:                  93 *
C8C1:20 4E CA    (6)   94        jsr   send80
C8C4:                  95 *
C8C4:                  96 * Send over the packet type (command or data)
C8C4:                  97 *
C8C4:A5 5B       (3)   98        lda   Wpackettype
C8C6:20 50 CA    (6)   99        jsr   sendbyte
C8C9:                 100 *
C8C9:                 101 * Send the Auxilliary Type byte (an $80 from this rev PC)
C8C9:                 102 *
C8C9:20 4E CA    (6)  103        jsr   send80
C8CC:                 104 *
C8CC:                 105 * Send the status byte (null for us), and length bytes
C8CC:                 106 *
C8CC:20 4E CA    (6)  107        jsr   send80
C8CF:A5 4C       (3)  108        lda   oddbytes
C8D1:09 80       (2)  109        ora   #$80
C8D3:20 50 CA    (6)  110        jsr   sendbyte
C8D6:A5 4B       (3)  111        lda   grp7ctr
C8D8:09 80       (2)  112        ora   #$80
C8DA:20 50 CA    (6)  113        jsr   sendbyte
C8DD:                 114 *
C8DD:                 115 * Now send the "oddbytes" part of the packet contents
C8DD:                 116 *
C8DD:A5 4C       (3)  117        lda   oddbytes      ;Get # of "odd" bytes
C8DF:F0 15   C8F6(3)  118        beq   sob2          ;Skip if no odd bytes
C8E1:                 119 *
```



```
C8E1:A0 FF       (2)  120        ldy   #$FF
C8E3:A5 59       (3)  121        lda   tbodd         ;Get the odd bytes msb's (A[7]=1)
C8E5:                 122 *
C8E5:1E 8C C0    (7)  123 sob1   asl   16clr,x       ;Do a write handshake
C8E8:90 FB   C8E5(3)  124        bcc   sob1
C8EA:9D 8D C0    (5)  125        sta   16set,x
C8ED:C8          (2)  126        iny
C8EE:B1 54       (5)  127        lda   (buffer),y    ;Get the data byte
C8F0:09 80       (2)  128        ora   #$80          ;Flip on the hi bit
C8F2:C4 4C       (3)  129        cpy   oddbytes      ;Are we done?
C8F4:90 EF   C8E5(3)  130        blt   sob1
C8F6:                 131 *
C8F6:                 132 * Now send over the groups of seven contents
C8F6:                 133 * Currently assume there must be at least one group of 'em
C8F6:                 134 *
C8F6:        C8F6     135 sob2   equ   *
C8F6:A5 4B       (3)  136        lda   grp7ctr       ;Check if there are groups to send
C8F8:D0 03   C8FD(3)  137        bne   sob3          ;=> At least one group
C8FA:4C 96 C9    (3)  138        jmp   datdone       ;Skip to send checksum
C8FD:                 139 *
C8FD:        C8FD     140 sob3   equ   *
C8FD:EA          (2)  141        nop                 ;Waste 2 cycles
C8FE:A0 00       (2)  142        ldy   #0
C900:A5 41       (3)  143 start  lda   topbits
C902:9D 8D C0    (5)  144        sta   16set,x
C905:                 145 *
C905:                 146 * Send first byte
C905:                 147 *
C905:A5 4D       (3)  148        lda   next1
C907:09 80       (2)  149        ora   #$80
C909:84 59       (3)  150        sty   temp          ;Swap Y for short handshake
C90B:BC 8C C0    (4)  151 ache1  ldy   16clr,x       ;Wait 'til buffer ready
C90E:10 FB   C90B(3)  152        bpl   ache1
C910:9D 8D C0    (5)  153        sta   16set,x       ;Send the byte
C913:A4 59       (3)  154        ldy   temp          ;Get back Y
C915:                 155 *
C915:                 156 * Prep the next "1st" byte for next time
C915:                 157 *
C915:B1 56       (5)  158        lda   (buffer2),y
C917:85 4D       (3)  159        sta   next1
C919:0A          (2)  160        asl   a
C91A:26 41       (5)  161        rol   topbits       ;Store the top bit
C91C:C8          (2)  162        iny                 ;Next byte
C91D:                 163 *
C91D:                 164 * It's possible that we're at a page boundary now. If so, bump the
C91D:                 165 *  hi order part of the pointer.
C91D:                 166 *
C91D:D0 05   C924(3)  167        bne   skip1
C91F:E6 57       (5)  168        inc   buffer2+1
C921:4C 26 C9    (3)  169        jmp   skip2
C924:48          (3)  170 skip1  pha                 ;Equalize the cases
C925:68          (4)  171        pla
C926:                 172 *
C926:                 173 * Push us ahead by an additional 8 cycles for margin reasons
C926:                 174 *  Plus I gotta get the topbits MSB set somehow...
C926:                 175 *
C926:        C926     176 skip2  equ   *
C926:A9 02       (2)  177        lda   #%00000010    ;Flip what will be MSB
```

```
C928:05 41      (3)  178        ora   topbits
C92A:85 41      (3)  179        sta   topbits
C92C:                180 *
C92C:                181 * Send the second byte
C92C:                182 *
C92C:A5 4E      (3)  183        lda   next2
C92E:09 80      (2)  184        ora   #$80
C930:9D 8D C0   (5)  185        sta   16set,x       ;Send the byte
C933:B1 56      (5)  186        lda   (buffer2),y
C935:85 4E      (3)  187        sta   next2
C937:0A         (2)  188        asl   a
C938:26 41      (5)  189        rol   topbits       ;Store the top bit
C93A:C8         (2)  190        iny                 ;Next byte
C93B:                191 *
C93B:                192 * Send the third byte
C93B:                193 *
C93B:A5 4F      (3)  194        lda   next3
C93D:09 80      (2)  195        ora   #$80
C93F:9D 8D C0   (5)  196        sta   16set,x       ;Send the byte
C942:B1 56      (5)  197        lda   (buffer2),y
C944:85 4F      (3)  198        sta   next3
C946:0A         (2)  199        asl   a
C947:26 41      (5)  200        rol   topbits       ;Store the top bit
C949:C8         (2)  201        iny                 ;Next byte
C94A:                202 *
C94A:                203 * Send the fourth byte
C94A:                204 *
C94A:A5 50      (3)  205        lda   next4
C94C:09 80      (2)  206        ora   #$80
C94E:9D 8D C0   (5)  207        sta   16set,x       ;Send the byte
C951:B1 56      (5)  208        lda   (buffer2),y
C953:85 50      (3)  209        sta   next4
C955:0A         (2)  210        asl   a
C956:26 41      (5)  211        rol   topbits       ;Store the top bit
C958:C8         (2)  212        iny                 ;Next byte
C959:                213 *
C959:                214 * After the first 256 bytes, we will cross pages here.  If we did
C959:                215 *  cross, bump the buffer pointer. If not, equalize the cases with
C959:                216 *  seven cycles of time wasting.
C959:                217 *
C959:D0 05   C960(3) 218        bne   skip3
C95B:E6 57      (5)  219        inc   buffer2+1
C95D:4C 62 C9   (3)  220        jmp   skip4
C960:48         (3)  221 skip3  pha
C961:68         (4)  222        pla
C962:        C962    223 skip4  equ   *
C962:                224 *
C962:                225 * Send the fifth byte
C962:                226 *
C962:A5 51      (3)  227        lda   next5
C964:09 80      (2)  228        ora   #$80
C966:9D 8D C0   (5)  229        sta   16set,x       ;Send the byte
C969:B1 56      (5)  230        lda   (buffer2),y
C96B:85 51      (3)  231        sta   next5
C96D:0A         (2)  232        asl   a
C96E:26 41      (5)  233        rol   topbits       ;Store the top bit
C970:C8         (2)  234        iny                 ;Next byte
C971:                235 *
```



```
C971:                236 * Send the sixth byte
C971:                237 *
C971:A5 52      (3)  238        lda   next6
C973:09 80      (2)  239        ora   #$80
C975:9D 8D C0   (5)  240        sta   16set,x       ;Send the byte
C978:B1 56      (5)  241        lda   (buffer2),y
C97A:85 52      (3)  242        sta   next6
C97C:0A         (2)  243        asl   a
C97D:26 41      (5)  244        rol   topbits       ;Store the top bit
C97F:C8         (2)  245        iny                 ;Next byte
C980:                246 *
C980:                247 * Send the last byte of the group
C980:                248 *
C980:A5 53      (3)  249        lda   next7
C982:09 80      (2)  250        ora   #$80
C984:9D 8D C0   (5)  251        sta   16set,x       ;Send the byte
C987:B1 56      (5)  252        lda   (buffer2),y
C989:85 53      (3)  253        sta   next7
C98B:0A         (2)  254        asl   a
C98C:26 41      (5)  255        rol   topbits       ;Store the top bit
C98E:C8         (2)  256        iny                 ;Next byte
C98F:                257 *
C98F:                258 * Now see if we have sent enough groups of seven
C98F:                259 *
C98F:C6 4B      (5)  260        dec   grp7ctr
C991:F0 03   C996(3) 261        beq   datdone
C993:                262 *
C993:                263 * Otherwise, back to do more.  Note it's too far for a branch.
C993:                264 *
C993:4C 00 C9   (3)  265        jmp   start
C996:                266 *
C996:                267 * Whew! Now send the damn checksum as two FM bytes
C996:                268 *
C996:        C996    269 datdone equ  *
C996:A5 40      (3)  270        lda   checksum      ;c7 c6 c5 c4 c3 c2 c1 c0
C998:09 AA      (2)  271        ora   #$AA          ; 1 c6  1 c4  1 c2  1 c0
C99A:BC 8C C0   (4)  272 scm1   ldy   16clr,x
C99D:10 FB   C99A(3) 273        bpl   scm1          ;Handshake this byte
C99F:9D 8D C0   (5)  274        sta   16set,x       ;These are even bits
C9A2:                275 *
C9A2:A5 40      (3)  276        lda   checksum      ;c7 c6 c5 c4 c3 c2 c1 c0
C9A4:4A         (2)  277        lsr   a             ; 0 c7 c6 c5 c4 c3 c2 c1
C9A5:09 AA      (2)  278        ora   #$AA          ; 1 c7  1 c5  1 c3  1 c1
C9A7:20 50 CA   (6)  279        jsr   sendbyte
C9AA:                280 *
C9AA:                281 * Send the end of packet mark
C9AA:                282 *
C9AA:A9 C8      (2)  283        lda   #packetend
C9AC:20 50 CA   (6)  284        jsr   sendbyte
C9AF:                285 *
C9AF:                286 * Wait until write underflow
C9AF:                287 *
C9AF:BD 8C C0   (4)  288 sd7    lda   16clr,x
C9B2:29 40      (2)  289        and   #$40
C9B4:D0 F9   C9AF(3) 290        bne   sd7           ;Still writing data
C9B6:                291 *
C9B6:9D 8D C0   (5)  292        sta   16set,x       ;Back to sense mode (dummy write)
C9B9:                293 *
```

```
C9B9:                   294 * Now wait until the drive acknowledges reciept of the
C9B9:                   295 *  string or until timeout
C9B9:                   296 *
C9B9:A0 0A       (2)    297         ldy   #bsyto2       ;Load timeout to see bsy low
C9BB:88          (2)    298 patch1 dey                   ;A little closer to an error
C9BC:D0 08   C9C6(3)    299         bne   sd9           ;There's still time
C9BE:                   300 *
C9BE:                   301 * Too much time has elapsed.  Drive didn't get string.
C9BE:                   302 *
C9BE:A9 01       (2)    303         lda   #noanswer     ;Report error in comm error byte
C9C0:       C9C0        304 dberror equ   *
C9C0:20 97 CA    (6)    305         jsr   SetXN0        ;For dberror entry
C9C3:38          (2)    306         sec                 ;Signal a problem
C9C4:B0 06   C9CC(3)    307         bcs   sd10
C9C6:                   308 *
C9C6:                   309 * See if drive has acknowledged the bytes yet
C9C6:                   310 *
C9C6:BD 8E C0    (4)    311 sd9     lda   17clr,x       ;Wait 'til /BSY lo
C9C9:30 F0   C9BB(3)    312         bmi   patch1
C9CB:                   313 *
C9CB:                   314 * Finish the sequence
C9CB:                   315 *
C9CB:18          (2)    316         clc                 ;This is a normal exit
C9CC:BD 80 C0    (4)    317 sd10    lda   reqclr,x      ;Set REQ lo
C9CF:BD 8C C0    (4)    318         lda   16clr,x       ;Back into read mode
C9D2:                   319 *
C9D2:                   320 * Pull back the bytecount in all cases
C9D2:                   321 *
C9D2:60          (6)    322         rts
C9D3:                   323 *
C9D3:                   324 *
C9D3:                   325 * This table, when sent in reverse order, provides a
C9D3:                   326 *  sync pattern used to synchronize the drive IWM with
C9D3:                   327 *  the data stream.  The first byte (last sent) is the
C9D3:                   328 *  packet begin mark.
C9D3:                   329 *
C9D3:C3                 330 preamble dfb packetbeg
C9D4:FF FC F3 CF        331 synctab dfb  $FF,$FC,$F3,$CF,$3F
C9D9:                   332 *
C9D9:                   333 *
```



```
C9D9:                   335 *
C9D9:                   336 * These routines are for wasting specific amounts of time
C9D9:                   337 *  This code segment should not cross page boundaries.
C9D9:                   338 *
C9D9:20 DE C9    (6)    339 waste32 jsr  waste14
C9DC:EA          (2)    340 waste18 nop
C9DD:EA          (2)    341 waste16 nop
C9DE:EA          (2)    342 waste14 nop
C9DF:60          (6)    343 waste12 rts
C9E0:                   344 *
C9E0:                   345 *
C9E0:       C9E0        346 markerr equ  *
C9E0:4C C0 C9    (3)    347         jmp  dberror
```

```
C9E3:                  349 ********************************************************
C9E3:                  350 *                                                      *
C9E3:                  351 *  ReceivePack       Get a packet from bus resident    *
C9E3:                  352 *                                                      *
C9E3:                  353 *                                                      *
C9E3:                  354 *  REQ _____|2                              5|_______  *
C9E3:                  355 *                                                      *
C9E3:                  356 *  /BSY ___|1                3              4|_______  *
C9E3:                  357 *                                                      *
C9E3:                  358 *    1)  Drive signals ready to send packet            *
C9E3:                  359 *    2)  Host signals ready to recieve data            *
C9E3:                  360 *    3)  Packet is transmitted (sync, mark, IDs, data, *
C9E3:                  361 *          checksum [msb=1])                           *
C9E3:                  362 *    4)  Drive signals packet dispatched               *
C9E3:                  363 *    5)  Host acknowledges reciept of packet           *
C9E3:                  364 *                                                      *
C9E3:                  365 *    The bytes are sent in slow mode (32 cycles/byte)  *
C9E3:                  366 *  and the timing is critical.  Branches which should  *
C9E3:                  367 *  not cross page boundaries are marked.               *
C9E3:                  368 *                                                      *
C9E3:                  369 *    Input:  buffer <- address where packet guts left  *
C9E3:                  370 *                                                      *
C9E3:                  371 *    Output: carry set- handshake error                *
C9E3:                  372 *                    clr- bytes recieved               *
C9E3:                  373 *            A <- error0 if carry set                  *
C9E3:                  374 *                                                      *
C9E3:                  375 ********************************************************
C9E3:                  376 *
C9E3:          C9E3    377 grabstatus equ *
C9E3:          C9E3    378 ReceivePack equ *
C9E3:                  379 *
C9E3:                  380 * Init the checksum
C9E3:                  381 *
C9E3:A9 00       (2)   382         lda   #$00
C9E5:85 40       (3)   383         sta   checksum
C9E7:                  384 *
C9E7:                  385 * Copy over buffer -> buffer2
C9E7:                  386 *
C9E7:A5 54       (3)   387         lda   buffer
C9E9:85 56       (3)   388         sta   buffer2
C9EB:A5 55       (3)   389         lda   buffer+1
C9ED:85 57       (3)   390         sta   buffer2+1
C9EF:                  391 *
C9EF:                  392 * Set up the indirect pointer for jump to 2nd part of code
C9EF:                  393 *
C9EF:20 7D CA    (6)   394         jsr   enablechain   ;Set X register to $N0
C9F2:                  395 *
C9F2:BD 8D C0    (4)   396         lda   16set,x       ;Prep for sense mode
C9F5:                  397 *
C9F5:                  398 * Now wait for BSY to go hi, signalling 'ready w/ status'
C9F5:                  399 *
C9F5:BD 8E C0    (4)   400 rdh1    lda   17clr,x       ;Read sense
C9F8:10 FB   C9F5(3)   401         bpl   rdh1          ;Wait til a high
C9FA:                  402 *
C9FA:                  403 * Signal Liron we're ready to recieve
C9FA:                  404 *
C9FA:BD 81 C0    (4)   405         lda   reqset,x      ;Raise /REQ
C9FD:                  406 *
```



```
C9FD:                  407 * Wait for a byte from Liron or timeout
C9FD:                  408 *
C9FD:A0 1E       (2)   409         ldy   #statmto      ;Max bytes 'til stat mark
C9FF:BD 8C C0    (4)   410 rdh2    lda   16clr,x
CA02:10 FB   C9FF(3)   411         bpl   rdh2          ;*** No Page Cross ***
CA04:88          (2)   412         dey
CA05:30 D9   C9E0(3)   413         bmi   markerr       ;Didn't find a packet in time
CA07:                  414 *
CA07:                  415 * Is it the beginning of the packet?
CA07:                  416 *
CA07:C9 C3       (2)   417         cmp   #packetbeg    ;Find the packet begin mark
CA09:D0 F4   C9FF(3)   418         bne   rdh2          ;Back again - no timeout for now
CA0B:                  419 *
CA0B:                  420 * Okay load up the table with this stuff
CA0B:                  421 *
CA0B:          CA0B    422 rdh5    equ   *
CA0B:                  423 *
CA0B:A0 06       (2)   424         ldy   #6            ;Seven bytes of overhead
CA0D:BD 8C C0    (4)   425 rdh3    lda   16clr,x       ;If byte ready, grab it
CA10:10 FB   CA0D(3)   426         bpl   rdh3          ;*** No Page Cross ***
CA12:29 7F       (2)   427         and   #%01111111    ;Strip start bit
CA14:99 4B 00    (5)   428         sta   rcvbuf,y
CA17:49 80       (2)   429         eor   #$80          ;Pop MSB back on for checksum
CA19:45 40       (3)   430         eor   checksum
CA1B:85 40       (3)   431         sta   checksum
CA1D:88          (2)   432         dey
CA1E:10 ED   CA0D(3)   433         bpl   rdh3
CA20:                  434 *
CA20:                  435 * Set groups of seven buffer pointer buffer2
CA20:                  436 *
CA20:A5 4C       (3)   437         lda   oddbytes
CA22:F0 27   CA4B(3)   438         beq   start2        ;Skip alteration if no oddbytes
CA24:18          (2)   439         clc
CA25:65 54       (3)   440         adc   buffer
CA27:85 56       (3)   441         sta   buffer2
CA29:A5 55       (3)   442         lda   buffer+1
CA2B:69 00       (2)   443         adc   #0
CA2D:85 57       (3)   444         sta   buffer2+1
CA2F:                  445 *
CA2F:A0 00       (2)   446         ldy   #0
CA31:                  447 *
CA31:                  448 * Now receive the odd bytes
CA31:                  449 *
CA31:BD 8C C0    (4)   450 start0  lda   16clr,x       ;Read in the odd bytes topbits
CA34:10 FB   CA31(3)   451         bpl   start0
CA36:0A          (2)   452         asl   a             ;Pop off the start bit
CA37:85 41       (3)   453         sta   topbits
CA39:          CA39    454 start1  equ   *
CA39:BD 8C C0    (4)   455         lda   16clr,x       ;Get an odd byte
CA3C:10 FB   CA39(3)   456         bpl   start1
CA3E:06 41       (5)   457         asl   topbits       ;Get an MSB
CA40:B0 02   CA44(3)   458         bcs   gob1          ;If MSB set, leave start bit
CA42:49 80       (2)   459         eor   #$80          ;MSB clear- flip start bit
CA44:91 54       (6)   460 gob1    sta   (buffer),y    ;Squirrel it away
CA46:C8          (2)   461         iny                 ;Next spot
CA47:C4 4C       (3)   462         cpy   oddbytes      ;Are we done?
CA49:90 EE   CA39(3)   463         blt   start1        ;If more, branch
CA4B:                  464 *
```

```
CA4B:          CA4B     465 start2 equ   *
CA4B:4C 73 CC    (3)    466        jmp   SlotDepRd
CA4E:                   467 *
CA4E:          CA4E     468 Send80 equ   *
CA4E:A9 80       (2)    469        lda   #$80
CA50:          CA50     470 SendByte equ *
CA50:BC 8C C0    (4)    471        ldy   16clr,x
CA53:10 FB   CA50(3)    472        bpl   SendByte
CA55:9D 8D C0    (5)    473        sta   16set,x
CA58:45 40       (3)    474        eor   checksum
CA5A:85 40       (3)    475        sta   checksum
CA5C:60          (6)    476        rts
CA5D:                   477 *
CA5D:                   478 *
CA5D:                   479 *
CA5D:                   480 *
CA5D:          CA5D     481 resetchain equ *
CA5D:20 87 CA    (6)    482        jsr   ClrPhases
CA60:BD 81 C0    (4)    483        lda   reqset,x
CA63:BD 85 C0    (4)    484        lda   ca2set,x
CA66:A0 50       (2)    485        ldy   #80          ;Hard reset for 80 ms
CA68:20 70 CA    (6)    486        jsr   YMSWait
CA6B:                   487 *
CA6B:20 87 CA    (6)    488        jsr   ClrPhases
CA6E:                   489 *
CA6E:A0 0A       (2)    490        ldy   #10          ;About 10 mS reset time!
CA70:                   491 *
CA70:          CA70     492 YMSWait equ  *
CA70:20 77 CA    (6)    493        jsr   OneMS
CA73:88          (2)    494        dey
CA74:D0 FA   CA70(3)    495        bne   YMSWait
CA76:60          (6)    496        rts
CA77:                   497 *
CA77:          CA77     498 OneMS  equ   *
CA77:A2 C8       (2)    499        ldx   #200
CA79:CA          (2)    500 onems1 dex
CA7A:D0 FD   CA79(3)    501        bne   onems1
CA7C:60          (6)    502        rts
CA7D:                   503 *
CA7D:                   504 *
CA7D:          CA7D     505 enablechain equ *
CA7D:20 97 CA    (6)    506        jsr   SetXN0
CA80:BD 83 C0    (4)    507        lda   calset,x
CA83:BD 87 C0    (4)    508        lda   lstrbset,x
CA86:60          (6)    509        rts
CA87:                   510 *
CA87:                   511 *
CA87:          CA87     512 ClrPhases equ *
CA87:20 97 CA    (6)    513        jsr   SetXN0
CA8A:BD 80 C0    (4)    514        lda   reqclr,x
CA8D:BD 82 C0    (4)    515        lda   calclr,x
CA90:BD 84 C0    (4)    516        lda   ca2clr,x
CA93:BD 86 C0    (4)    517        lda   lstrbclr,x
CA96:60          (6)    518        rts
CA97:                   519 *
CA97:                   520 *
CA97:          CA97     521 SetXN0 equ   *
CA97:A2 60       (2)    522        ldx   #$60
```



```
CA99:                   523 *
CA99:60          (6)    524        rts
CA9A:                   525 *
CA9A:                   526 * Shift tables for use when reading.  Each table should not
CA9A:                   527 *  straddle pages.
CA9A:                   528 *
CA9A:80 80 80 80        529 shift1 dfb   $80,$80,$80,$80,$80,$80,$80,$80
CAA2:00 00 00 00        530        dfb   0,0,0,0,0,0,0,0
CAAA:80 80 80 80        531 shift2 dfb   $80,$80,$80,$80,0,0,0,0
CAB2:80 80 80 80        532        dfb   $80,$80,$80,$80,0,0,0,0
CABA:80 80 00 00        533 shift3 dfb   $80,$80,0,0,$80,$80,0,0
CAC2:80 80 00 00        534        dfb   $80,$80,0,0,$80,$80,0,0
CACA:80 00 80 00        535 shift4 dfb   $80,0,$80,0,$80,0,$80,0
CAD2:80 00 80 00        536        dfb   $80,0,$80,0,$80,0,$80,0
CADA:                   537 *
CADA:                   538 *
```

```
CADA:                   540 *
CADA:           CADA    541 SendData equ *
CADA:A9 05        (2)   542         lda   #>RC2
CADC:A0 00        (2)   543         ldy   #<RC2
CADE:20 FD CA     (6)   544         jsr   SendPile
CAE1:90 05    CAE8(3)   545         bcc   sdoubt
CAE3:A9 80        (2)   546         lda   #CommReset
CAE5:20 98 CF     (6)   547         jsr   AssignID
CAE8:           CAE8    548 sdoubt equ   *
CAE8:60           (6)   549         rts
CAE9:                   550 *
CAE9:                   551 *
CAE9:           CAE9    552 SendPack equ *
CAE9:20 FD CA     (6)   553         jsr   SendPile      ;Try to send a pack
CAEC:90 FA    CAE8(3)   554         bcc   sdoubt
CAEE:A9 80        (2)   555         lda   #CommReset    ;This is a communications failure
CAF0:20 98 CF     (6)   556         jsr   AssignID      ;Reset to try again
CAF3:                   557 *
CAF3:AD F8 06     (4)   558         lda   SvBcL         ;Get back the packetlength
CAF6:85 4D        (3)   559         sta   bytecount1
CAF8:AD 78 07     (4)   560         lda   SvBcH
CAFB:85 4E        (3)   561         sta   bytecounth
CAFD:                   562 *
CAFD:           CAFD    563 SendPile equ *
CAFD:A9 B8        (2)   564         lda   #>RC1         ;Retry count (big!)
CAFF:A0 0B        (2)   565         ldy   #<RC1
CB01:                   566 *
CB01:           CB01    567 AltSendPile equ *
CB01:A6 58        (3)   568         ldx   slot
CB03:9D F3 04     (5)   569         sta   Retry,x
CB06:98           (2)   570         tya
CB07:9D 73 05     (5)   571         sta   Retry2,x
CB0A:                   572 *
CB0A:                   573 * SendPack destroys the bytecount
CB0A:                   574 *
CB0A:           CB0A    575 spile1 equ   *
CB0A:A5 4D        (3)   576         lda   bytecount1
CB0C:8D F8 06     (4)   577         sta   SvBcL
CB0F:A5 4E        (3)   578         lda   bytecounth
CB11:8D 78 07     (4)   579         sta   SvBcH
CB14:                   580 *
CB14:20 83 C8     (6)   581         jsr   SendOnePack   ;Send the packet
CB17:                   582 *
CB17:AD F8 06     (4)   583         lda   SvBcL
CB1A:85 4D        (3)   584         sta   bytecount1
CB1C:AD 78 07     (4)   585         lda   SvBcH
CB1F:85 4E        (3)   586         sta   bytecounth
CB21:                   587 *
CB21:90 0C    CB2F(3)   588         bcc   spilout
CB23:A6 58        (3)   589         ldx   slot
CB25:DE F3 04     (7)   590         dec   Retry,x
CB28:D0 E0    CB0A(3)   591         bne   spile1
CB2A:DE 73 05     (7)   592         dec   Retry2,x
CB2D:10 DB    CB0A(3)   593         bpl   spile1        ;If all fails, carry is set
CB2F:60           (6)   594 spilout rts
CB30:                   595 *
CB30:           CB30    596 RecPack equ  *
CB30:A4 58        (3)   597         ldy   Slot
```



```
CB32:A9 05        (2)   598         lda   #>RC2
CB34:99 F3 04     (5)   599         sta   Retry,y
CB37:           CB37    600 rpk1    equ   *
CB37:20 E3 C9     (6)   601         jsr   ReceivePack
CB3A:90 0F    CB4B(3)   602         bcc   rpout
CB3C:A0 01        (2)   603         ldy   #1
CB3E:20 70 CA     (6)   604         jsr   YMSWait
CB41:20 C0 C9     (6)   605         jsr   dberror       ;Recycle handshake and set carry
CB44:A6 58        (3)   606         ldx   Slot
CB46:DE F3 04     (7)   607         dec   Retry,x
CB49:D0 EC    CB37(3)   608         bne   rpk1          ;Carry set still
CB4B:           CB4B    609 rpout   equ   *
CB4B:60           (6)   610         rts
CB4C:                   611 *
CB4C:                   612 *
```

```
CB4C:                  614 ************************************************************
CB4C:                  615 *                                                          *
CB4C:                  616 *  Divide7                Do DIV and MOD 7 and set auxptr  *
CB4C:                  617 *                                                          *
CB4C:                  618 *   This routine divides the bytecount by seven.  The      *
CB4C:                  619 *  quotient gives the number of groups of seven bytes to   *
CB4C:                  620 *  be sent, and the remainder gives the number of "odd"    *
CB4C:                  621 *  bytes.                                                  *
CB4C:                  622 *                                                          *
CB4C:                  623 *  Input:   bytecountl,h <- # of bytes to write            *
CB4C:                  624 *           buffer       <- pointer to data                *
CB4C:                  625 *  Output:  auxptr       <- pointer to speed up csumming   *
CB4C:                  626 *           oddbytes     <- bytecount MOD 7                *
CB4C:                  627 *           grp7ctr      <- bytecount DIV 7                *
CB4C:                  628 *                                                          *
CB4C:                  629 ************************************************************
CB4C:                  630 *
CB4C:00 24 49          631 pdiv7tab dfb 0,36,73
CB4F:00 04 01          632 pmod7tab dfb 0,4,1
CB52:00 01 02 04       633 div7tab dfb  0,1,2,4,9,18
CB58:00 01 02 04       634 mod7tab dfb  0,1,2,4,1,2
CB5E:                  635 *
CB5E:00 7F FF          636 auxptrinc dfb 0,$7F,$FF
CB61:                  637 *
CB61:        CB61      638 WritePrep equ *
CB61:        CB61      639 Divide7 equ  *
CB61:                  640 *
CB61:                  641 * Set up auxptr <- buffer+$80   if $0FF < bytecount < $200
CB61:                  642 *     or auxptr <- buffer+$100  if $1FF < bytecount
CB61:                  643 *
CB61:A6 4E       (3)   644         ldx   bytecounth    ;0, 1 or 2
CB63:F0 17   CB7C(3)   645         beq   noauxptr      ;Auxptr used only for full pages
CB65:                  646 *
CB65:A5 55       (3)   647         lda   buffer+1
CB67:85 57       (3)   648         sta   auxptr+1      ;Copy over hi order part
CB69:                  649 *
CB69:A9 80       (2)   650         lda   #$80          ;Anticipate smaller bytecount
CB6B:E0 01       (2)   651         cpx   #1            ;Check bytecount
CB6D:F0 04   CB73(3)   652         beq   sap1          ;=> $0FF < bytecount < $200
CB6F:                  653 *
CB6F:E6 57       (5)   654         inc   auxptr+1      ;Add $100 to bytecount instead
CB71:A9 00       (2)   655         lda   #0            ;Make sure lo order unaltered
CB73:18          (2)   656 sap1    clc
CB74:65 54       (3)   657         adc   buffer
CB76:85 56       (3)   658         sta   auxptr
CB78:90 02   CB7C(3)   659         bcc   noauxptr      ;skip if no carry
CB7A:E6 57       (5)   660         inc   auxptr+1      ;don't forget me
CB7C:                  661 *
CB7C:                  662 * Now look up the first order guess for DIV and MOD.  X still has
CB7C:                  663 *  bytecount DIV 256.
CB7C:                  664 *
CB7C:        CB7C      665 noauxptr equ *
CB7C:BD 4C CB    (4)   666         lda   pdiv7tab,x
CB7F:85 4B       (3)   667         sta   grp7ctr
CB81:BD 4F CB    (4)   668         lda   pmod7tab,x
CB84:85 4C       (3)   669         sta   oddbytes
CB86:                  670 *
CB86:                  671 * Now add in the mods and divs for each of the five hi order
```



```
CB86:                  672 *  bits in the lo order bytecount, correcting each time MOD becomes
CB86:                  673 *  bigger than 6.
CB86:                  674 *
CB86:A2 05       (2)   675         ldx   #5            ;Do for five bits
CB88:A5 4D       (3)   676         lda   bytecountl
CB8A:85 59       (3)   677         sta   temp          ;Store lo order for shifting
CB8C:29 07       (2)   678         and   #%00000111    ;Save lo three for later
CB8E:A8          (2)   679         tay
CB8F:                  680 *
CB8F:        CB8F      681 divide3 equ  *
CB8F:06 59       (5)   682         asl   temp          ;C <- next from bytecountl
CB91:90 15   CBA8(3)   683         bcc   divide2       ;If clear, no effect on DIV,MOD
CB93:BD 58 CB    (4)   684         lda   mod7tab,x     ;Get MOD7 for 2^n
CB96:        CB96      685 divide4 equ  *
CB96:18          (2)   686         clc
CB97:65 4C       (3)   687         adc   oddbytes      ;Got new MOD value
CB99:C9 07       (2)   688         cmp   #7            ;Is it too big?
CB9B:90 02   CB9F(3)   689         blt   divide1       ;=> NO leave MOD - 0->C
CB9D:E9 07       (2)   690         sbc   #7            ;Bring MOD under 7 - C still set
CB9F:        CB9F      691 divide1 equ  *
CB9F:85 4C       (3)   692         sta   oddbytes
CBA1:BD 52 CB    (4)   693         lda   div7tab,x     ;Get DIV for this 2^n
CBA4:65 4B       (3)   694         adc   grp7ctr       ;Add to DIV along with correction (C)
CBA6:85 4B       (3)   695         sta   grp7ctr       ;Update the DIV
CBA8:        CBA8      696 divide2 equ  *
CBA8:CA          (2)   697         dex                 ;One less bit to deal with
CBA9:30 06   CBB1(3)   698         bmi   divide5       ;Escape after 6 times through loop
CBAB:D0 E2   CB8F(3)   699         bne   divide3       ;Take brnch 1st 5 loops
CBAD:                  700 *
CBAD:98          (2)   701         tya                 ;Get back the last three bits
CBAE:4C 96 CB    (3)   702         jmp   divide4       ;Sixth pass add in remains
CBB1:                  703 *
CBB1:        CBB1      704 divide5 equ  *
CBB1:                  705 *
CBB1:                  706 *
```

```
                        708 ************************************************************
                        709 *                                                          *
                        710 *  PreCheck                    Does the checksumming prepass *
                        711 *                                                          *
                        712 *   Input:   bytecount   <- bytes in buffer                *
                        713 *            buffer      <- pointer to data to send        *
                        714 *            auxptr      <- extra pointer to speed process *
                        715 *   Output:  checksum    <- 8 bit XOR of data to be sent   *
                        716 *                                                          *
                        717 ************************************************************
                        718 *
CBB1:           CBB1    719 PreCheck equ *
                        720 *
                        721 * Checksum any full pages
                        722 *
CBB1:A5 55      (3)     723         lda   buffer+1
CBB3:48         (3)     724         pha                   ;Preserve buffer pointer
CBB4:A9 00      (2)     725         lda   #0
CBB6:A6 4E      (3)     726         ldx   bytecounth
CBB8:F0 16      CBD0(3) 727         beq   lastpass        ;If no complete pages, skip this
CBBA:           CBBA    728 xor2    equ   *
CBBA:BC 5E CB   (4)     729         ldy   auxptrinc,x     ;Get number of bytes each ptr
CBBD:           CBBD    730 xor1    equ   *
CBBD:51 54      (5)     731         eor   (buffer),y
CBBF:51 56      (5)     732         eor   (auxptr),y
CBC1:88         (2)     733         dey                   ;One less
CBC2:D0 F9      CBBD(3) 734         bne   xor1
CBC4:51 54      (5)     735         eor   (buffer),y
CBC6:51 56      (5)     736         eor   (auxptr),y      ;Have to deal with 0 case
CBC8:                   737 *
CBC8:                   738 * Now move the buffer up for next section
CBC8:                   739 *
CBC8:E0 01      (2)     740         cpx   #1
CBCA:F0 02      CBCE(3) 741         beq   xor5            ;If 256 and up bytes, bump x1
CBCC:E6 55      (5)     742         inc   buffer+1        ; otherwise x2
CBCE:E6 55      (5)     743 xor5    inc   buffer+1
CBD0:                   744 *
CBD0:           CBD0    745 lastpass equ *
CBD0:                   746 *
CBD0:                   747 * Do the remaining less than a page with a single pointer
CBD0:                   748 *
CBD0:A4 4D      (3)     749         ldy   bytecount
CBD2:F0 09      CBDD(3) 750         beq   xor4
CBD4:51 54      (5)     751         eor   (buffer),y      ;Compensate for nth byte
CBD6:51 54      (5)     752 xor3    eor   (buffer),y
CBD8:88         (2)     753         dey
CBD9:D0 FB      CBD6(3) 754         bne   xor3
CBDB:51 54      (5)     755         eor   (buffer),y      ;Last damn (0th) byte
CBDD:                   756 *
CBDD:                   757 * Store result away.  Retrieve old buffer value.
CBDD:                   758 *
CBDD:           CBDD    759 xor4    equ   *
CBDD:85 40      (3)     760         sta   checksum
CBDF:68         (4)     761         pla
CBE0:85 55      (3)     762         sta   buffer+1
CBE2:                   763 *
CBE2:                   764 *
```



```
CBE2:                   766 ************************************************************
CBE2:                   767 *                                                          *
CBE2:                   768 * DetTopBits                 Get topbits for odd bytes    *
CBE2:                   769 *                                                          *
CBE2:                   770 *  Also sets buffer2 pointer to pointer at groups of     *
CBE2:                   771 *  seven bytes.                                            *
CBE2:                   772 *                                                          *
CBE2:                   773 *  Input:   oddbytes <- # of "odd" bytes                   *
CBE2:                   774 *           buffer   <- pointer to data                    *
CBE2:                   775 *  Output:  tbodd    <- topbits for odd bytes             *
CBE2:                   776 *           buffer2  <- buffer+oddbytes                    *
CBE2:                   777 *                                                          *
CBE2:                   778 ************************************************************
CBE2:                   779 *
CBE2:           CBE2    780 DetTopBits equ *
CBE2:                   781 *
CBE2:A4 4C      (3)     782         ldy   oddbytes
CBE4:88         (2)     783         dey
CBE5:A9 00      (2)     784         lda   #0
CBE7:85 59      (3)     785         sta   tbodd
CBE9:                   786 *
CBE9:B1 54      (5)     787 gtbob   lda   (buffer),y
CBEB:0A         (2)     788         asl   a
CBEC:66 59      (5)     789         ror   tbodd
CBEE:88         (2)     790         dey
CBEF:10 F8      CBE9(3) 791         bpl   gtbob
CBF1:38         (2)     792         sec
CBF2:66 59      (5)     793         ror   tbodd
CBF4:                   794 *
CBF4:A5 4C      (3)     795         lda   oddbytes
CBF6:18         (2)     796         clc
CBF7:65 54      (3)     797         adc   buffer
CBF9:85 56      (3)     798         sta   buffer2
CBFB:A5 55      (3)     799         lda   buffer+1
CBFD:69 00      (2)     800         adc   #0
CBFF:85 57      (3)     801         sta   buffer2+1
CC01:                   802 *
CC01:                   803 *
```

```
CC01:                 ***********************************************************
CC01:                 *                                                         *
CC01:                 *  Sun                    Set up next buffer and topbits  *
CC01:                 *                                                         *
CC01:                 *   Primes the pipe for the group of seven bytes routine *
CC01:                 *  setting the topbits byte and the "next" buffer.        *
CC01:                 *  The routine also advances the buffer pointer by 7 to  *
CC01:                 *  prepare for the groups of seven transfer.              *
CC01:                 *                                                         *
CC01:                 *    Input:   buffer2   <- points to groups of 7 data     *
CC01:                 *    Output:  next1,7   <- first 7 bytes in buffer        *
CC01:                 *             topbits   <- MSBs of first 7 bytes          *
CC01:                 *                                                         *
CC01:                 ***********************************************************
CC01:                 *
CC01:        CC01     Sun    equ   *
CC01:                 *
CC01:                 * Copy first seven bytes into the pipeline
CC01:                 *
CC01:A0 06       (2)        ldy   #6
CC03:38          (2)  sun2  sec
CC04:B1 56       (5)        lda   (buffer2),y
CC06:99 4D 00    (5)        sta   next,y
CC09:30 01   CC0C(3)        bmi   sun1
CC0B:18          (2)        clc
CC0C:66 41       (5)  sun1  ror   topbits
CC0E:88          (2)        dey
CC0F:10 F2   CC03(3)        bpl   sun2
CC11:38          (2)        sec
CC12:66 41       (5)        ror   topbits
CC14:                 *
CC14:                 * Advance the pointer
CC14:                 *
CC14:A5 56       (3)        lda   buffer2
CC16:18          (2)        clc
CC17:69 07       (2)        adc   #7
CC19:85 56       (3)        sta   buffer2
CC1B:90 02   CC1F(3)        bcc   sun3
CC1D:E6 57       (5)        inc   buffer2+1
CC1F:        CC1F     sun3  equ   *
CC1F:60          (6)        rts
CC20:                 *
CC20:                 *
```



```
CC20:                 *
CC20:                 * X is slot*16, Y is the desired mode
CC20:                 *
CC20:                 * Set up the IWM mode register.  Extreme care should be taken
CC20:                 *  here. Setting the mode byte with indexed stores causes a
CC20:                 *  false byte to be written a cycle before the real value is
CC20:                 *  written.  This false value, if it enables the timer, causes
CC20:                 *  the IWM Rev A to pop the motor on, inhibiting the setting
CC20:                 *  of the mode until the motor times out! We avoid this by
CC20:                 *  setting the mode byte only when it is not what we want, and if
CC20:                 *  it's not we stay here until we see that it is what we want.
CC20:                 *
CC20:        CC20     SetIWMode equ *
CC20:BD 88 C0    (4)        lda   monclr,x      ;Motor must be off
CC23:BD 8D C0    (4)        lda   l6set,x       ;Set up to access mode register
CC26:4C 2D CC    (3)        jmp   careful       ;Don't mess unless we gotta
CC29:98          (2)  biz   tya
CC2A:9D 8F C0    (5)        sta   l7set,x       ;Try storing the mode value
CC2D:        CC2D     careful equ  *
CC2D:98          (2)        tya                 ;Get back the target value
CC2E:5D 8E C0    (4)        eor   l7clr,x       ;Compare with observed value
CC31:29 1F       (2)        and   #$1F          ;Can only read low 5 bits
CC33:D0 F4   CC29(3)        bne   biz           ;If not right, back to try again
CC35:60          (6)        rts
CC36:                 *
CC36:                 *
CC36:        CC36     WaitIWMOff equ *
CC36:                 *
CC36:                 * Make sure you're in read mode and wait 'til Disk // motor is off
CC36:                 *
CC36:20 97 CA    (6)        jsr   SetXN0        ;Set X
CC39:BD 8E C0    (4)        lda   l7clr,x
CC3C:BD 8D C0    (4)        lda   l6set,x
CC3F:        CC3F     wiwm1 equ   *
CC3F:BD 8E C0    (4)        lda   l7clr,x
CC42:29 20       (2)        and   #%00100000
CC44:D0 F9   CC3F(3)        bne   wiwm1
CC46:BD 8C C0    (4)        lda   l6clr,x
CC49:                 *
CC49:                 *Wait an additional 700 usec to allow 12V on Disk // to decay
CC49:                 *
CC49:5A          (3)        phy
CC4A:A0 8C       (2)        ldy   #140
CC4C:88          (2)  wiwm2 dey
CC4D:D0 FD   CC4C(3)        bne   wiwm2
CC4F:7A          (4)        ply
CC50:                 *
CC50:60          (6)        rts
CC51:                 *
CC51:                 *
CC51:                 * This takes grp7ctr and oddbytes and calculates 7*grp7ctr+oddbytes.
CC51:                 *  The results are in Y(hi) and A(lo).  This is the number of bytes
CC51:                 *  that were received in the last ReceivePack.
CC51:                 *
CC51:        CC51     Rcvcount equ *
CC51:A5 4B       (3)        lda   grp7ctr
CC53:A8          (2)        tay
CC54:A2 00       (2)        ldx   #0
```

```
CC56:86 4B      (3)  907        stx   grp7ctr
CC58:A2 03      (2)  908        ldx   #3
CC5A:0A         (2)  909 times7 asl   a
CC5B:26 4B      (5)  910        rol   grp7ctr
CC5D:CA         (2)  911        dex
CC5E:D0 FA   CC5A(3) 912        bne   times7
CC60:18         (2)  913        clc
CC61:65 4C      (3)  914        adc   oddbytes
CC63:90 02   CC67(3) 915        bcc   t71
CC65:E6 4B      (5)  916        inc   grp7ctr
CC67:84 4C      (3)  917 t71    sty   oddbytes
CC69:38         (2)  918        sec
CC6A:E5 4C      (3)  919        sbc   oddbytes
CC6C:B0 02   CC70(3) 920        bcs   t72
CC6E:C6 4B      (5)  921        dec   grp7ctr
CC70:A4 4B      (3)  922 T72    ldy   grp7ctr
CC72:60         (6)  923        rts
CC73:                924 *
CC73:                925 *
CC73:                115        include pc.cread
CC73:        CC73      1 SlotDepRd equ *
CC73:        CC73      2 start25 equ  *
CC73:A0 00      (2)    3        ldy   #0
CC75:A5 4B      (3)    4        lda   grp7ctr
CC77:48         (3)    5        pha                  ;Save groups of seven counter
CC78:D0 03   CC7D(3)   6        bne   start35
CC7A:4C 0A CD   (3)    7        jmp   done5          ;Go get the checksum
CC7D:                  8 *
CC7D:                  9 * Okay, get the groups of seven
CC7D:                 10 *  Start by getting the topbits for this group of seven
CC7D:                 11 *
CC7D:        CC7D     12 start35 equ  *
CC7D:AD EC C0   (4)   13        lda   16clr+TheOff   ;Get topbits
CC80:10 FB   CC7D(3)  14        bpl   start35
CC82:85 59      (3)   15        sta   temp           ;Just a second
CC84:                 16 *
CC84:                 17 * Split up the seven bits into two indices for topbit tables
CC84:                 18 *
CC84:4A         (2)   19        lsr   a              ;0  1  d1  d2  d3  d4  d5  d6
CC85:4A         (2)   20        lsr   a              ;0  0   1  d1  d2  d3  d4  d5
CC86:4A         (2)   21        lsr   a              ;0  0   0   1  d1  d2  d3  d4
CC87:29 0F      (2)   22        and   #%00001111     ;0  0   0   0  d1  d2  d3  d4
CC89:AA         (2)   23        tax                  ;First index into the tables
CC8A:A5 59      (3)   24        lda   temp           ;1  d1  d2  d3  d4  d5  d6  d7
CC8C:29 07      (2)   25        and   #%00000111     ;0   0   0   0   0  d5  d6  d7
CC8E:85 59      (3)   26        sta   temp           ;Keep for last three bytes
CC90:                 27 *
CC90:                 28 * Read the 1st byte, reunite its msb, store and checksum it
CC90:                 29 *
CC90:AD EC C0   (4)   30        lda   16clr+TheOff
CC93:10 FB   CC90(3)  31        bpl   *-3            ;Back 1 instruction
CC95:5D 9A CA   (4)   32        eor   shift1,x       ;Recombine the MSB with data
CC98:91 56      (6)   33        sta   (buffer2),y    ;Store it away
CC9A:45 40      (3)   34        eor   checksum       ;Add it to the checksum
CC9C:85 40      (3)   35        sta   checksum
CC9E:C8         (2)   36        iny
CC9F:                 37 *
CC9F:                 38 * Now, the second Y turn over occurs at this point in the
```



```
CC9F:                 39 *  loop.  Update the buffer pointer if it occurred.
CC9F:                 40 *
CC9F:D0 02   CCA3(3)  41        bne   *+4
CCA1:E6 57      (5)   42        inc   buffer2+1
CCA3:                 43 *
CCA3:                 44 * Now the second byte
CCA3:                 45 *
CCA3:AD EC C0   (4)   46        lda   16clr+TheOff
CCA6:10 FB   CCA3(3)  47        bpl   *-3            ;Back 1 instruction
CCA8:5D AA CA   (4)   48        eor   shift2,x       ;Recombine the MSB with data
CCAB:91 56      (6)   49        sta   (buffer2),y    ;Store it away
CCAD:45 40      (3)   50        eor   checksum       ;Add it to the checksum
CCAF:85 40      (3)   51        sta   checksum
CCB1:C8         (2)   52        iny
CCB2:                 53 *
CCB2:                 54 * Now the third byte
CCB2:                 55 *
CCB2:AD EC C0   (4)   56        lda   16clr+TheOff
CCB5:10 FB   CCB2(3)  57        bpl   *-3            ;Back 1 instruction
CCB7:5D BA CA   (4)   58        eor   shift3,x       ;Recombine the MSB with data
CCBA:91 56      (6)   59        sta   (buffer2),y    ;Store it away
CCBC:45 40      (3)   60        eor   checksum       ;Add it to the checksum
CCBE:85 40      (3)   61        sta   checksum
CCC0:C8         (2)   62        iny
CCC1:                 63 *
CCC1:                 64 * Now the fourth byte
CCC1:                 65 *
CCC1:AD EC C0   (4)   66        lda   16clr+TheOff
CCC4:10 FB   CCC1(3)  67        bpl   *-3            ;Back 1 instruction
CCC6:5D CA CA   (4)   68        eor   shift4,x       ;Recombine the MSB with data
CCC9:91 56      (6)   69        sta   (buffer2),y    ;Store it away
CCCB:45 40      (3)   70        eor   checksum       ;Add it to the checksum
CCCD:85 40      (3)   71        sta   checksum
CCCF:C8         (2)   72        iny
CCD0:                 73 *
CCD0:                 74 * The first Y turn over occurs at this point in the loop.  Update
CCD0:                 75 *  the buffer pointer if it occurred.
CCD0:                 76 *
CCD0:D0 02   CCD4(3)  77        bne   *+4
CCD2:E6 57      (5)   78        inc   buffer2+1
CCD4:                 79 *
CCD4:A6 59      (3)   80        ldx   temp           ;Now we need the other index
CCD6:                 81 *
CCD6:                 82 * Now the fifth byte
CCD6:                 83 *
CCD6:AD EC C0   (4)   84        lda   16clr+TheOff
CCD9:10 FB   CCD6(3)  85        bpl   *-3            ;Back 1 instruction
CCDB:5D AA CA   (4)   86        eor   shift2,x       ;Recombine the MSB with data
CCDE:91 56      (6)   87        sta   (buffer2),y    ;Store it away
CCE0:45 40      (3)   88        eor   checksum       ;Add it to the checksum
CCE2:85 40      (3)   89        sta   checksum
CCE4:C8         (2)   90        iny
CCE5:                 91 *
CCE5:                 92 * Now the sixth byte
CCE5:                 93 *
CCE5:AD EC C0   (4)   94        lda   16clr+TheOff
CCE8:10 FB   CCE5(3)  95        bpl   *-3            ;Back 1 instruction
CCEA:5D BA CA   (4)   96        eor   shift3,x       ;Recombine the MSB with data
```

```
CCED:91 56      (6)   97        sta   (buffer2),y   ;Store it away
CCEF:45 40      (3)   98        eor   checksum      ;Add it to the checksum
CCF1:85 40      (3)   99        sta   checksum
CCF3:C8         (2)  100        iny
CCF4:                101 *
CCF4:                102 * And, finally, the seventh byte
CCF4:                103 *
CCF4:AD EC C0   (4)  104        lda   16clr+TheOff
CCF7:10 FB   CCF4(3) 105        bpl   *-3           ;Back 1 instruction
CCF9:5D CA CA   (4)  106        eor   shift4,x      ;Recombine the MSB with data
CCFC:91 56      (6)  107        sta   (buffer2),y   ;Store it away
CCFE:45 40      (3)  108        eor   checksum      ;Add it to the checksum
CD00:85 40      (3)  109        sta   checksum
CD02:C8         (2)  110        iny
CD03:                111 *
CD03:                112 * Now see if this is the last group of seven to receive
CD03:                113 *
CD03:C6 4B      (5)  114        dec   grp7ctr
CD05:F0 03   CD0A(3) 115        beq   done5         ;Go to get the checksum etc
CD07:4C 7D CC   (3)  116        jmp   start35       ;Another topbits ...
CD0A:                117 *
CD0A:                118 * Get and reconstruct the checksum
CD0A:                119 *
CD0A:       CD0A     120 done5  equ   *
CD0A:AD EC C0   (4)  121        lda   16clr+TheOff
CD0D:10 FB   CD0A(3) 122        bpl   *-3
CD0F:85 59      (3)  123        sta   temp          ;1 c6 1 c4 1 c2 1 c0
CD11:                124 *
CD11:68         (4)  125        pla                 ;Restore groups of 7 counter
CD12:85 4B      (3)  126        sta   grp7ctr
CD14:AD EC C0   (4)  127        lda   16clr+TheOff  ;1 c7 1 c5 1 c3 1 c1
CD17:10 FB   CD14(3) 128        bpl   *-3
CD19:38         (2)  129        sec
CD1A:2A         (2)  130        rol   a             ;c7 1 c5 1 c3 1 c1 1
CD1B:25 59      (3)  131        and   temp          ;c7 c6 c5 c4 c3 c2 c1 c0
CD1D:45 40      (3)  132        eor   checksum      ;When we're done, should be zero
CD1F:                133 *
CD1F:                134 * Get the packet end mark. Is it correct?
CD1F:                135 *
CD1F:AC EC C0   (4)  136 rdha5  ldy   16clr+TheOff  ;Preserve A
CD22:10 FB   CD1F(3) 137        bpl   rdha5
CD24:                138 *
CD24:C0 C8      (2)  139        cpy   #packetend
CD26:D0 1C   CD44(3) 140        bne   npenderr5
CD28:                141 *
CD28:                142 * Didn't have time before to checksum oddbytes.  Do it now
CD28:                143 *  A still has the partial checksum
CD28:                144 *
CD28:A6 4C      (3)  145        ldx   oddbytes
CD2A:F0 08   CD34(3) 146        beq   icbt15
CD2C:A0 00      (2)  147        ldy   #0
CD2E:51 54      (5)  148 icbt5  eor   (buffer),y
CD30:C8         (2)  149        iny
CD31:CA         (2)  150        dex
CD32:D0 FA   CD2E(3) 151        bne   icbt5
CD34:                152 *
CD34:                153 * Okay, checksum oughta be zero.  If not, checksum error.
CD34:                154 *
```



```
CD34:       CD34     155 icbt15 equ   *
CD34:AA         (2)  156        tax
CD35:D0 11   CD48(3) 157        bne   cserror5
CD37:                158 *
CD37:                159 * Wait for /BSY to go low
CD37:                160 *
CD37:       CD37     161 lstbsywait5 equ *
CD37:AD ED C0   (4)  162        lda   16set+TheOff
CD3A:AD EE C0   (4)  163 rdh45  lda   17clr+TheOff
CD3D:30 FB   CD3A(3) 164        bmi   rdh45
CD3F:                165 *
CD3F:                166 * Got the bytes, now acknowledge their receipt
CD3F:                167 *
CD3F:AD E0 C0   (4)  168        lda   reqclr+TheOff ;Lower REQ
CD42:                169 *
CD42:18         (2)  170        clc
CD43:60         (6)  171        rts
CD44:                172 *
CD44:A9 20      (2)  173 npenderr5 lda #nopackend
CD46:D0 02   CD4A(3) 174        bne   gserror5
CD48:A9 10      (2)  175 cserror5 lda #csumerr
CD4A:38         (2)  176 gserror5 sec
CD4B:60         (6)  177        rts
CD4C:                178 *
CD4C:                116        include pc.main
```

```
CD4C:                    2 *
CD4C:                    3 *
CD4C:          CD4C      4 Entry  equ   *
CD4C:90 03     CD51(3)   5        bcc   bentry       ;If non-boot, skip jump to boot
CD4E:4C 23 C5     (3)    6        jmp   bootcode
CD51:                    7 *
CD51:                    8 * X is still set to slot number.
CD51:                    9 *
CD51:          CD51      10 bentry equ  *
CD51:                    11 *
CD51:A9 40        (2)    12       lda   #%01000000
CD53:1C 78 04     (6)    13       trb   ProFlag+5     ;ProFlag is fixed in //c
CD56:                    14 *
CD56:          CD56      15 atentry equ *
CD56:                    16 *
CD56:D8           (2)    17       cld                 ;Don't want decimal mode!!
CD57:8A           (2)    18       txa
CD58:A8           (2)    19       tay                 ;Really want it in Y... no ROR ABS,Y!
CD59:                    20 *
CD59:                    21 * If this is a PC call, then get the address of the parm table
CD59:                    22 *
CD59:B9 73 04     (4)    23       lda   ProFlag,y
CD5C:30 11     CD6F(3)   24       bmi   noplay
CD5E:                    25 *
CD5E:68           (4)    26       pla                 ;Get lo order
CD5F:99 F3 05     (5)    27       sta   SHTempX,y     ;Keep lo parm address-1
CD62:18           (2)    28       clc
CD63:69 03        (2)    29       adc   #3
CD65:AA           (2)    30       tax                 ;Lo order new return address
CD66:68           (4)    31       pla                 ;Get hi order address
CD67:99 73 06     (5)    32       sta   SHTempY,y     ;Keep hi parm addr-1
CD6A:69 00        (2)    33       adc   #0
CD6C:48           (3)    34       pha                 ;Push back new return address hi
CD6D:8A           (2)    35       txa
CD6E:48           (3)    36       pha                 ;Push new return address lo
CD6F:                    37 *
CD6F:          CD6F      38 noplay equ  *
CD6F:                    39 *
CD6F:                    40 * On the //c, it is important to have the Disk // enable lines
CD6F:                    41 *  off for as long as possible before using the IWM (phases,
CD6F:                    42 *  /WRREQ lines).  Wait here 'til the Disk // motors are off.
CD6F:                    43 *
CD6F:20 36 CC     (6)    44       jsr   WaitIWMOff    ;Must preserve Y!!
CD72:                    45 *
CD72:                    46 * We can't tolerate ints in most of the code, so disable
CD72:                    47 *
CD72:08           (3)    48       php                 ;Save interrupt status
CD73:78           (2)    49       sei                 ;No interrupts please
CD74:                    50 *
CD74:                    51 * Preserve the zero page work area
CD74:                    52 *
CD74:A2 1B        (2)    53       ldx   #ZPSize-1
CD76:B5 40        (4)    54 pzp   lda   ZeroPage,x
CD78:48           (3)    55       pha
CD79:CA           (2)    56       dex
CD7A:10 FA     CD76(3)   57       bpl   pzp
CD7C:                    58 *
CD7C:                    59 * Okay, we're safe... now it's all right to store in zero page
```



```
CD7C:                    60 *
CD7C:84 58        (3)    61       sty   Slot
CD7E:                    62 *
CD7E:                    63 *
CD7E:                    64 * Now map any ProDOS unit references to our sequential ones.
CD7E:                    65 *  The method is bizzare and magicians never reveal their secrets.
CD7E:                    66 *
CD7E:          CD7E      67 allset equ  *
CD7E:A5 43        (3)    68       lda   CMDUnit       ;76543210 7&6 specify unit
CD80:2A           (2)    69       rol   a             ;6543210X C<-7
CD81:08           (3)    70       php                 ;Save drive num
CD82:2A           (2)    71       rol   a             ;543210X7 C<-6
CD83:2A           (2)    72       rol   a             ;43210X76 (6 is grp of 2)
CD84:28           (4)    73       plp                 ;C<-7
CD85:2A           (2)    74       rol   a             ;3210X767
CD86:29 03        (2)    75       and   #%00000011    ;ProDOS only installs up to 4
CD88:49 02        (2)    76       eor   #%00000010    ;000000/67; 6 was /grpoftwo
CD8A:C0 04        (2)    77       cpy   #4            ;If in slot 1,2,or3 reverse grps of two
CD8C:B0 02     CD90(3)   78       bge   allset1
CD8E:49 02        (2)    79       eor   #%00000010
CD90:AA           (2)    80 allset1 tax
CD91:E8           (2)    81       inx
CD92:86 43        (3)    82       stx   CMDUnit       ;You got it
CD94:                    83 *
CD94:                    84 * Now if this is through the MLI xface, gotta copy stuff into the
CD94:                    85 *  send buffer from the parameter list.
CD94:                    86 *
CD94:B9 73 04     (4)    87       lda   ProFlag,y
CD97:10 03     CD9C(3)   88       bpl   darnit
CD99:4C 40 CE     (3)    89       jmp   skipcopy
CD9C:                    90 *
CD9C:                    91 * Get the address of the in-line parameter table
CD9C:                    92 *
CD9C:          CD9C      93 darnit equ  *
CD9C:B9 F3 05     (4)    94       lda   SHTempX,y     ;Get back the low part buff addr
CD9F:85 54        (3)    95       sta   buffer
CDA1:B9 73 06     (4)    96       lda   SHTempY,y     ; and the hi part
CDA4:85 55        (3)    97       sta   buffer+1
CDA6:                    98 *
CDA6:                    99 * Now pull out the command code, and the address of the parameters.
CDA6:                   100 *
CDA6:A0 01        (2)   101       ldy   #1            ;Stacked address is EA-1
CDA8:B1 54        (5)   102       lda   (buffer),y
CDAA:85 42        (3)   103       sta   cmdcode       ;Nice
CDAC:C8           (2)   104       iny
CDAD:B1 54        (5)   105       lda   (buffer),y    ;Get lo part of parmlist address
CDAF:AA           (2)   106       tax                 ;Save it
CDB0:C8           (2)   107       iny
CDB1:B1 54        (5)   108       lda   (buffer),y    ;Get hi part
CDB3:85 55        (3)   109       sta   buffer+1
CDB5:86 54        (3)   110       stx   buffer
CDB7:                   111 *
CDB7:                   112 * Now buffer points to parmlist
CDB7:                   113 *  Check command type, and pidgeonhole the parmlist length
CDB7:                   114 *
CDB7:A9 01        (2)   115       lda   #BadCmd
CDB9:A6 42        (3)   116       ldx   cmdcode
CDBB:E0 0A        (2)   117       cpx   #$A           ;Only valid codes are 0-9
```

```
CDBD:90 03   CDC2(3)  118        blt   noeh          ;=> at least he got that right
CDBF:4C 17 CF    (3)  119 Errorhitch jmp Error       ;Gee, maybe we should promote this guy...
CDC2:        CDC2     120 noeh    equ   *
CDC2:A0 00       (2)  121        ldy   #0            ;Set for indct compare
CDC4:B1 54       (5)  122        lda   (buffer),y    ;Get # of parms?
CDC6:85 5A       (3)  123        sta   Unit
CDC8:                 124 *
CDC8:                 125 * Now copy the bytes
CDC8:                 126 *
CDC8:        CDC8     127 okaycnt equ  *
CDC8:A0 08       (2)  128        ldy   #>cmdlength-1 ;Always copy the maximum
CDCA:        CDCA     129 copyloop equ *
CDCA:B1 54       (5)  130        lda   (buffer),y    ;Pull it out of their hat
CDCC:99 42 00    (5)  131        sta   cmdcode,y     ;Stuff it into mine
CDCF:88          (2)  132        dey
CDD0:D0 F8   CDCA(3)  133        bne   copyloop      ;Copy 'em all
CDD2:                 134 *
CDD2:                 135 * Okay.  The caller of the PC could be making one of three calls
CDD2:                 136 *  with a unit number of $00, Control, Init or Status.  Check for
CDD2:                 137 *  these and do what is appropriate.
CDD2:                 138 *
CDD2:A5 43       (3)  139        lda   CMDUnit
CDD4:D0 6A   CE40(3)  140        bne   skipcopy      ;Never mind
CDD6:                 141 *
CDD6:                 142 * Check the parameter count for this call to unit#0
CDD6:                 143 *
CDD6:A6 42       (3)  144        ldx   CMDCode
CDD8:BD 8E CF    (4)  145        lda   parmctab,x    ;Get the length this command
CDDB:29 7F       (2)  146        and   #$7F          ;Force 0 -> MSB
CDDD:A8          (2)  147        tay                 ;Hang on
CDDE:A9 04       (2)  148        lda   #BadPCnt      ;Antic bad count
CDE0:C4 5A       (3)  149        cpy   Unit          ;User's pcount is currently here
CDE2:D0 DB   CDBF(3)  150        bne   ErrorHitch    ;What a baby!
CDE4:                 151 *
CDE4:                 152 * Now service one of the three commands
CDE4:                 153 *
CDE4:E0 05       (2)  154        cpx   #InitCMD
CDE6:D0 0A   CDF2(3)  155        bne   notinit       ;Not an Init call
CDE8:A9 00       (2)  156        lda   #PowerReset   ;Just like powerup or reset key(//c)
CDEA:20 98 CF    (6)  157        jsr   AssignID      ;Do a reset cycle
CDED:A9 00       (2)  158 Aokay  lda   #0            ;No error allowed
CDEF:4C 39 CF    (3)  159        jmp   sa2
CDF2:                 160 *
CDF2:8A          (2)  161 notinit txa                ;Equiv to 'cmp #StatusCMD'
CDF3:D0 24   CE19(3)  162        bne   maybectrl
CDF5:                 163 *
CDF5:A9 21       (2)  164        lda   #BadCtl       ;Antic a non zero stat code
CDF7:A6 46       (3)  165        ldx   CMDSCode      ;Stat unit#0 can only be code=0
CDF9:D0 C4   CDBF(3)  166        bne   ErrorHitch
CDFB:                 167 *
CDFB:8A          (2)  168        txa                 ;Equiv to 'lda #0'
CDFC:A6 58       (3)  169        ldx   Slot
CDFE:A0 07       (2)  170        ldy   #7
CE00:91 44       (6)  171 nin1   sta   (CmdBufferl),y ;Clear some space
CE02:88          (2)  172        dey
CE03:D0 FB   CE00(3)  173        bne   nin1
CE05:                 174 *
CE05:BD F9 06    (4)  175        lda   NumDevices,x
```



```
CE08:91 44       (6)  176        sta   (CMDBufferl),y ;Stick it where they want it
CE0A:C8          (2)  177        iny
CE0B:                 178 *
CE0B:AD F9 04    (4)  179        lda   $4F9          ;//c Port 1 interrupt status
CE0E:                 180 *
CE0E:91 44       (6)  181        sta   (CMDBufferl),y ;Store PC interrupt status
CE10:                 182 *
CE10:A9 08       (2)  183        lda   #8
CE12:88          (2)  184        dey                 ;A,Y has 0008; # bytes status
CE13:20 F0 CF    (6)  185        jsr   squirrel
CE16:                 186 *
CE16:4C ED CD    (3)  187        jmp   Aokay         ;Skip down (up) with no error
CE19:        CE19     188 maybectrl equ *
CE19:C9 04       (2)  189        cmp   #ControlCMD
CE1B:D0 0B   CE28(3)  190        bne   BUnit         ;Unit #0 was a bad one
CE1D:                 191 *
CE1D:A6 46       (3)  192        ldx   CMDSCode      ;We allow two control calls for Unit#0
CE1F:F0 0B   CE2C(3)  193        beq   enabint       ;0 means enable interrupts
CE21:CA          (2)  194        dex
CE22:F0 14   CE38(3)  195        beq   disabint      ;1 means disable interrupts
CE24:A9 21       (2)  196        lda   #badctl
CE26:        CE26     197 ErrorHitch2 equ *
CE26:D0 97   CDBF(3)  198        bne   ErrorHitch    ;No other codes allowed
CE28:                 199 *
CE28:        CE28     200 BUnit  equ   *
CE28:A9 11       (2)  201        lda   #badUnit      ;Only certain calls can have Unit#0
CE2A:D0 93   CDBF(3)  202        bne   ErrorHitch    ;Branch always
CE2C:                 203 *
CE2C:        CE2C     204 enabint equ  *
CE2C:A9 C0       (2)  205        lda   #$C0
CE2E:8D F9 05    (4)  206        sta   $5F9
CE31:A9 0F       (2)  207        lda   #$0F
CE33:0C 9A C0    (6)  208        tsb   $C09A
CE36:D0 05   CE3D(3)  209        bne   aokayhitch
CE38:                 210 *
CE38:        CE38     211 disabint equ *
CE38:A9 01       (2)  212        lda   #$01
CE3A:1C 9A C0    (6)  213        trb   $C09A
CE3D:4C ED CD    (3)  214 aokayhitch jmp AOkay
CE40:                 215 *
CE40:                 216 *
CE40:                 217 * Okay, everything's all groovy.  ProDOS re-enters here.
CE40:                 218 *  Check Unit number to be sure there is a corresponding device
CE40:                 219 *
CE40:        CE40     220 skipcopy equ *
CE40:A9 28       (2)  221        lda   #NoDrive      ;Anticpate bad unit number
CE42:A4 58       (3)  222        ldy   slot
CE44:BE F9 06    (4)  223        ldx   NumDevices,y
CE47:E4 43       (3)  224        cpx   CMDUnit
CE49:90 DB   CE26(3)  225        blt   ErrorHitch2   ;Safe- If C clr then Z is clr
CE4B:                 226 *
CE4B:                 227 * Set buffer and bytecount in anticpation of the inevitable SendPack.
CE4B:                 228 *
CE4B:A9 09       (2)  229        lda   #>cmdlength
CE4D:85 4D       (3)  230        sta   bytecountl
CE4F:A9 00       (2)  231        lda   #<cmdlength
CE51:85 4E       (3)  232        sta   bytecounth
CE53:85 55       (3)  233        sta   buffer+1
```

```
CE55:A9 42      (2)  234        lda   #>cmdcode
CE57:85 54      (3)  235        sta   buffer
CE59:                236 *
CE59:                237 * If it's a PC call, omit the next two steps
CE59:                238 *
CE59:A6 58      (3)  239        ldx   Slot
CE5B:BD 73 04   (4)  240        lda   ProFlag,x     ;Is it a call from ProDOS?
CE5E:10 13   CE73(3) 241        bpl   notstat       ;=> Statcode already set...
CE60:                242 *
CE60:                243 * Need to generate a parameter count for a ProDOS call
CE60:                244 *
CE60:A6 42      (3)  245        ldx   CMDCode
CE62:BD 8E CF   (4)  246        lda   ParmCTab,x
CE65:29 7F      (2)  247        and   #$7F
CE67:85 5A      (3)  248        sta   Unit
CE69:                249 *
CE69:                250 * ProDOS always needs the highest blockno byte zeroed
CE69:                251 *
CE69:A9 00      (2)  252        lda   #0
CE6B:85 48      (3)  253        sta   CMDBlockS
CE6D:                254 *
CE6D:                255 * If this is a ProDOS status call, set stat code to zero
CE6D:                256 *
CE6D:A5 42      (3)  257        lda   CMDCode
CE6F:D0 02   CE73(3) 258        bne   notstat       ;=> Not status so forget it
CE71:                259 *lda #SCDeviceStat ;A is already zero
CE71:85 46      (3)  260        sta   CMDSCode      ;Store in command table
CE73:                261 *
CE73:                262 * Okay, finally send over the damn command
CE73:                263 *
CE73:        CE73    264 notstat equ  *
CE73:A5 5A      (3)  265        lda   Unit
CE75:A6 43      (3)  266        ldx   CmdPCount     ;Swap the Parmcount & unit#
CE77:86 5A      (3)  267        stx   Unit
CE79:85 43      (3)  268        sta   CMDPCount     ;Now they're correct
CE7B:                269 *
CE7B:A9 80      (2)  270        lda   #cmdmark
CE7D:85 5B      (3)  271        sta   WPacketType
CE7F:                272 *
CE7F:20 87 CA   (6)  273        jsr   ClrPhases     ;Bring all phases off for Quark
CE82:                274 *
CE82:20 E9 CA   (6)  275        jsr   SendPack
CE85:B0 46   CECD(3) 276        bcs   behitch       ;If not okay, skip to bus error
CE87:                277 *
CE87:                278 * Now copy over the buffer address for any data xfer.
CE87:                279 *
CE87:A5 44      (3)  280        lda   CMDBuffer
CE89:85 54      (3)  281        sta   buffer
CE8B:A5 45      (3)  282        lda   CMDBuffer+1
CE8D:85 55      (3)  283        sta   buffer+1
CE8F:                284 *
CE8F:                285 * Now for some commands, we have to send over a packet of data, too.
CE8F:                286 *  See if this command is one of THOSE.
CE8F:                287 *
CE8F:A6 42      (3)  288        ldx   cmdcode
CE91:BD 8E CF   (4)  289        lda   parmctab,x
CE94:10 3B   CED1(3) 290        bpl   noxtrasend    ;Encoded in top bit
CE96:                291 *
```



```
CE96:                292 * The buffer address and bytecount depend on the call type.
CE96:                293 *
CE96:E0 04      (2)  294        cpx   #ControlCmd
CE98:D0 18   CEB2(3) 295        bne   NOControl
CE9A:                296 *
CE9A:                297 * In the case of control, bytecount:=(buffer)
CE9A:                298 * and buffer := buffer+2
CE9A:                299 *
CE9A:A0 01      (2)  300        ldy   #1
CE9C:B1 54      (5)  301        lda   (buffer),y    ;Get Hi order bytecount
CE9E:AA         (2)  302        tax
CE9F:88         (2)  303        dey
CEA0:B1 54      (5)  304        lda   (buffer),y
CEA2:48         (3)  305        pha                 ;Keep for later
CEA3:18         (2)  306        clc
CEA4:A9 02      (2)  307        lda   #2
CEA6:65 54      (3)  308        adc   buffer
CEA8:85 54      (3)  309        sta   buffer
CEAA:68         (4)  310        pla                 ;Get back Lo order bytecount
CEAB:90 13   CEC0(3) 311        bcc   secondsend    ;Skip hi ord increment
CEAD:E6 55      (5)  312        inc   buffer+1
CEAF:4C C0 CE   (3)  313        jmp   secondsend    ;Skip to store bytecount
CEB2:                314 *
CEB2:        CEB2    315 NOControl equ *
CEB2:E0 02      (2)  316        cpx   #WriteCMD     ;Check for a writeblock
CEB4:D0 06   CEBC(3) 317        bne   NOWBlock      ;Must be control or write
CEB6:                318 *
CEB6:                319 * In the case of WriteBlock, the length is 512 and the buffer
CEB6:                320 *  address is at buffer in the command table
CEB6:                321 *
CEB6:A9 00      (2)  322        lda   #0
CEB8:A2 02      (2)  323        ldx   #2
CEBA:D0 04   CEC0(3) 324        bne   secondsend
CEBC:                325 *
CEBC:                326 * For FileWrite, the buffer address is at CMDbuffer
CEBC:                327 *  and the length is at CMDblock.
CEBC:                328 *
CEBC:        CEBC    329 NOWBlock equ *
CEBC:A6 47      (3)  330        ldx   CMDBlockh
CEBE:A5 46      (3)  331        lda   CMDBlockl
CEC0:                332 *
CEC0:        CEC0    333 secondsend equ *
CEC0:86 4E      (3)  334        stx   bytecounth
CEC2:85 4D      (3)  335        sta   bytecountl
CEC4:                336 *
CEC4:A9 82      (2)  337        lda   #datamark
CEC6:85 5B      (3)  338        sta   WPacketType   ;Identify this as a data packet
CEC8:                339 *
CEC8:20 DA CA   (6)  340        jsr   SendData
CECB:90 04   CED1(3) 341        bcc   noxtrasend
CECD:        CECD    342 behitch equ  *
CECD:A9 06      (2)  343        lda   #BusErr       ;This is the bus error hitch
CECF:D0 46   CF17(3) 344        bne   Error
CED1:                345 *
CED1:                346 * On ProDOS status call, we've got to point the buffer pointer
CED1:                347 *  correctly to zero page... it's the only case special case
CED1:                348 *  (on Write, Format and Control no data comes back).
CED1:                349 *
```

```
CED1:          CED1     noxtrasend equ *
CED1:A4 58       (3)            ldy   Slot
CED3:B9 73 04    (4)            lda   ProFlag,y
CED6:10 0C   CEE4(3)            bpl   getresults
CED8:A5 42       (3)            lda   cmdcode
CEDA:D0 08   CEE4(3)            bne   getresults
CEDC:                           *
CEDC:A9 45       (2)            lda   #>CMDBufferh  ;Want status in these four
CEDE:A2 00       (2)            ldx   #<CMDBufferh
CEE0:85 54       (3)            sta   buffer
CEE2:86 55       (3)            stx   buffer+1
CEE4:                           *
CEE4:                           * Please to be calling ReceivePack
CEE4:                           *
CEE4:          CEE4             getresults equ *
CEE4:20 30 CB    (6)            jsr   RecPack       ;Get status byte (maybe read data too)
CEE7:B0 E4   CECD(3)            bcs   behitch
CEE9:                           *
CEE9:                           * Figure how many bytes were sent and put that in X,Y temps
CEE9:                           *
CEE9:20 51 CC    (6)            jsr   Rcvcount      ;Do the times 7...
CEEC:20 F0 CF    (6)            jsr   squirrel      ;Store away count in SHTEMPs
CEEF:                           *
CEEF:                           * For the ProDOS status call, we've got to look at the status byte
CEEF:                           *  returned and return a DIP error if appropriate.  Also overwrite
CEEF:                           *  the X,Y temps with # blocks if this is a ProDOS Stat call.
CEEF:                           *
CEEF:A5 42       (3)            lda   CMDCode       ;Is it a ProDOS status call
CEF1:D0 22   CF15(3)            bne   noerror
CEF3:A6 58       (3)            ldx   Slot
CEF5:BD 73 04    (4)            lda   ProFlag,x
CEF8:10 1B   CF15(3)            bpl   noerror
CEFA:                           *
CEFA:A5 46       (3)            lda   CMDBlockl     ;This'll get loaded into the XY regs later
CEFC:9D F3 05    (5)            sta   SHTempX,x
CEFF:A5 47       (3)            lda   CMDBlockh
CF01:9D 73 06    (5)            sta   SHTempY,x
CF04:                           *
CF04:A5 45       (3)            lda   CMDBufferh    ;Check status byte
CF06:4A          (2)            lsr   a
CF07:4A          (2)            lsr   a
CF08:4A          (2)            lsr   a
CF09:90 04   CF0F(3)            bcc   ChkOffLn      ;no error, go check off line
CF0B:A9 2B       (2)            lda   #WriteProt    ;else set WPROT error
CF0D:80 08   CF17(3)            bra   error
CF0F:          CF0F             ChkOffLn equ *
CF0F:4A          (2)            lsr   a
CF10:4A          (2)            lsr   a
CF11:A9 2F       (2)            lda   #OffLine      ;Assume error
CF13:90 02   CF17(3)            bcc   error
CF15:                           *
CF15:                           * Now it's time to think about returning to the caller
CF15:                           *  Remember that ProDOS doesn't want to know about soft errors,
CF15:                           *  only fatal ones.  If this is a ProDOS call, and the soft error
CF15:                           *  bit in the statbyte is set, there IS NO error (statbyte is
CF15:                           *  cleared).  Also, ProDOS wants only I/O, Write Protect, No Device,
CF15:                           *  Offline.  If any other hard error comes from the device
CF15:                           *  on a ProDOS call, map it to an I/O Error.  (Gross me out.)
```



```
CF15:                           *
CF15:          CF15             noerror equ *
CF15:A5 4D       (3)            lda   statbyte
CF17:          CF17             Error  equ *
CF17:A4 58       (3)            ldy   Slot          ;Need access to screenholes
CF19:99 F3 04    (5)            sta   Retry,Y       ;Keep unadulterated error in shole
CF1C:AA          (2)            tax                 ;Set the Z flag
CF1D:F0 1A   CF39(3)            beq   sa2           ;Special case the zero
CF1F:                           *
CF1F:BE 73 04    (4)            ldx   ProFlag,y     ;Set N to ProDOS call or not
CF22:10 15   CF39(3)            bpl   sa2           ;If PC call, no mapping occurs
CF24:                           *
CF24:A2 00       (2)            ldx   #0            ;Assume a soft error
CF26:C9 40       (2)            cmp   #%01000000    ;Soft error check
CF28:B0 0E   CF38(3)            bge   storeaway     ;If $40 or bigger, map to zero
CF2A:                           *
CF2A:A2 27       (2)            ldx   #IOError      ;Now anticipate ProDOS I/O error
CF2C:C9 2B       (2)            cmp   #WriteProt
CF2E:F0 09   CF39(3)            beq   sa2           ;OK to return Write Protect
CF30:C9 28       (2)            cmp   #NoDrive
CF32:F0 05   CF39(3)            beq   sa2           ;OK to return Drive disconnected
CF34:C9 2F       (2)            cmp   #OffLine
CF36:F0 01   CF39(3)            beq   SA2
CF38:                           *
CF38:          CF38             storeaway equ *
CF38:8A          (2)            txa                 ;Use the default value
CF39:          CF39             sa2    equ *
CF39:A4 58       (3)            ldy   Slot
CF3B:99 73 05    (5)            sta   SHTemp1,y     ;Keep in screenhole
CF3E:                           *
CF3E:                           * If this is the //c version, we need to reset the IWM to its
CF3E:                           *  former disk // state.  This is done by setting the mode register
CF3E:                           *  to a little known (and less documented) mode which speeds up the
CF3E:                           *  internal motor timeout.  When the motor enable has timed out, the
CF3E:                           *  mode can be set back to zero.  This method is necessary because
CF3E:                           *  if the timer is enabled within the timeout period, the motor on a
CF3E:                           *  Rev A IWM pops on for the full timeout period (since mode changes
CF3E:                           *  are disabled when the motor is on.  It's bizzarre. Blame Mac.
CF3E:AD E8 C0    (4)            lda   monclr+$60    ;Motor off
CF41:2C ED C0    (4)            bit   16set+$60     ;Into mode reg access mode
CF44:A9 2B       (2)            lda   #$2B          ;This is the magic "speed up" value
CF46:8D EF C0    (4)            sta   17set+$60     ;Throw into mode register
CF49:EA          (2)            nop                 ;You're supposed to wait a while
CF4A:EA          (2)            nop
CF4B:EA          (2)            nop
CF4C:EA          (2)            nop
CF4D:          CF4D             waitoff equ *
CF4D:AD EE C0    (4)            lda   17clr+$60     ;Wait 'til motor off
CF50:29 20       (2)            and   #$20
CF52:D0 F9   CF4D(3)            bne   waitoff
CF54:A0 00       (2)            ldy   #0            ;Now set the reg back to $00
CF56:A2 60       (2)            ldx   #$60          ;IWM's in slot 6
CF58:20 20 CC    (6)            jsr   SetIWMode
CF5B:AD EC C0    (4)            lda   16clr+$60
CF5E:AD E2 C0    (4)            lda   calclr+$60
CF61:AD E6 C0    (4)            lda   1strbclr+$60
CF64:A4 58       (3)            ldy   Slot          ;Need Slot in Y
CF66:                           *
```

```
07 PC.MAIN        Protocol Converter / CBus Driver   20-OCT-86  06:29 PAGE 43

CF66:                 466 * Now, restore our zero page area.
CF66:                 467 *
CF66:A2 00       (2)  468         ldx   #0
CF68:68          (4)  469 rzp     pla
CF69:95 40       (4)  470         sta   zeropage,x
CF6B:E8          (2)  471         inx
CF6C:E0 1C       (2)  472         cpx   #ZPSize
CF6E:90 F8   CF68(3)  473         blt   rzp
CF70:                 474 *
CF70:                 475 * We're into the stretch!  Restore interrupt mask, load X, Y,
CF70:                 476 *  and A and set the carry if the error byte is non-zero.
CF70:                 477 *
CF70:28          (4)  478         plp                 ;Restore interrupt flag
CF71:B9 F3 05    (4)  479         lda   SHTempx,y     ;Get X value
CF74:AA          (2)  480         tax
CF75:B9 73 05    (4)  481         lda   SHTemp1,y     ;Grab the error result code
CF78:48          (3)  482         pha
CF79:B9 73 06    (4)  483         lda   SHTempy,y     ;Pull out the Y value
CF7C:A8          (2)  484         tay                 ;No more access to screenholes
CF7D:18          (2)  485         clc                 ;Anticipate zero result code
CF7E:68          (4)  486         pla                 ;Pull back result code
CF7F:F0 01   CF82(3)  487         beq   finalskip     ;Return with carry clear
CF81:38          (2)  488         sec                 ;Some type of error
CF82:          CF82   489 finalskip equ *
CF82:                 490 *
CF82:08          (3)  491         php                 ;Save carry and Z flag
CF83:2C 78 04    (4)  492         bit   ProFlag+5     ;Ick - ProFlag is fixed in //c
CF86:70 04   CF8C(3)  493         bvs   ick1          ;If bit 6=1, then return to alt ROM
CF88:28          (4)  494         plp                 ;Vclr so return across ROM bank bdy
CF89:4C 84 C7    (3)  495         jmp   SWRTS2
CF8C:          CF8C   496 ick1    equ   *
CF8C:28          (4)  497         plp
CF8D:60          (6)  498         rts                 ;Flags set correctly again
CF8E:                 499 *
CF8E:                 500 *
CF8E:          CF8E   501 parmctab equ *
CF8E:03               502         dfb   %00000011     ;Status:  3 parms/no data send
CF8F:03               503         dfb   %00000011     ;Read:    3 parms/no data send
CF90:83               504         dfb   %10000011     ;Write:   3 parms/data send
CF91:01               505         dfb   %00000001     ;Format:  1 parm /no data send
CF92:83               506         dfb   %10000011     ;Control: 3 parms/data send
CF93:01               507         dfb   %00000001     ;Init:    1 parm /no data send
CF94:01               508         dfb   %00000001     ;Open:    1 parm /no data send
CF95:01               509         dfb   %00000001     ;Close:   1 parm /no data send
CF96:03               510         dfb   %00000011     ;CharRead: 3 parms/data send
CF97:83               511         dfb   %10000011     ;CharWrite: 3 parms/data send
CF98:                 512 *
CF98:                 513 *
```

```
07 PC.MAIN        ID Assignment Cycle                20-OCT-86  06:29 PAGE 44

CF98:                 515 *
CF98:          CF98   516 AssignID equ *
CF98:48          (3)  517         pha                 ;Save the init code
CF99:20 5D CA    (6)  518         jsr   resetchain    ;Reset all of those things
CF9C:68          (4)  519         pla
CF9D:AA          (2)  520         tax                 ;Save InitCode
CF9E:                 521 *
CF9E:                 522 * Save the command code, unit, and init code
CF9E:                 523 * 'cause we'll trample 'em.
CF9E:                 524 *
CF9E:A5 42       (3)  525         lda   CMDCode
CFA0:48          (3)  526         pha
CFA1:A5 43       (3)  527         lda   CMDPCount
CFA3:48          (3)  528         pha
CFA4:A5 46       (3)  529         lda   CMDSCode
CFA6:48          (3)  530         pha
CFA7:86 46       (3)  531         stx   CMDSCode      ;Store away the type of INIT
CFA9:                 532 *
CFA9:                 533 * Set up to send DefID command packets
CFA9:                 534 *
CFA9:A9 05       (2)  535         lda   #InitCmd
CFAB:85 42       (3)  536         sta   CMDCode
CFAD:A9 00       (2)  537         lda   #0
CFAF:85 5A       (3)  538         sta   Unit
CFB1:A9 02       (2)  539         lda   #2            ;# parms in Init call
CFB3:85 43       (3)  540         sta   CMDPCount
CFB5:                 541 *
CFB5:                 542 * Point the buffer pointer
CFB5:                 543 *
CFB5:A9 42       (2)  544         lda   #>CMDCode
CFB7:85 54       (3)  545         sta   buffer
CFB9:A9 00       (2)  546         lda   #<CMDCode
CFBB:85 55       (3)  547         sta   buffer+1
CFBD:A9 80       (2)  548         lda   #cmdmark
CFBF:85 5B       (3)  549         sta   WPacketType
CFC1:                 550 *
CFC1:20 87 CA    (6)  551         jsr   ClrPhases     ;Make sure phases are off for Quark
CFC4:                 552 *
CFC4:                 553 * Send an ID for the next device in the chain
CFC4:                 554 *
CFC4:          CFC4   555 mordevices equ *
CFC4:E6 5A       (5)  556         inc   Unit
CFC6:A9 09       (2)  557         lda   #>cmdlength
CFC8:85 4D       (3)  558         sta   bytecount1    ;ReceivePack scrambles count
CFCA:A9 00       (2)  559         lda   #<cmdlength
CFCC:85 4E       (3)  560         sta   bytecounth
CFCE:                 561 *
CFCE:20 83 C8    (6)  562         jsr   SendOnePack   ;Send the command
CFD1:90 05   CFD8(3)  563         bcc   mdev2         ;If okay, skip to get response
CFD3:                 564 *
CFD3:C6 5A       (5)  565         dec   Unit
CFD5:4C DF CF    (3)  566         jmp   mdev1
CFD8:                 567 *
CFD8:20 E3 C9    (6)  568 mdev2   jsr   ReceivePack   ;Get the response
CFDB:A5 4D       (3)  569         lda   statbyte
CFDD:F0 E5   CFC4(3)  570         beq   mordevices
CFDF:                 571 *
CFDF:                 572 * Okay, we done last device.  Squirrel away the number of devices.
```

```
CFDF:                   573 *
CFDF:A5 5A       (3)    574 mdev1  lda   Unit
CFE1:A4 58       (3)    575        ldy   slot
CFE3:99 F9 06    (5)    576        sta   NumDevices,y  ;Devices out there
CFE6:                   577 *
CFE6:                   578 * Recover the scrambled ProDOS parms
CFE6:                   579 *
CFE6:68          (4)    580        pla
CFE7:85 46       (3)    581        sta   CMDSCode
CFE9:68          (4)    582        pla
CFEA:85 43       (3)    583        sta   CMDPCount
CFEC:68          (4)    584        pla
CFED:85 42       (3)    585        sta   CMDCode
CFEF:                   586 *
CFEF:60          (6)    587        rts
CFF0:                   588 *
CFF0:                   589 *
CFF0:       CFF0        590 squirrel equ *
CFF0:A6 58       (3)    591        ldx   Slot
CFF2:9D F3 05    (5)    592        sta   SHTempX,x
CFF5:98          (2)    593        tya
CFF6:9D 73 06    (5)    594        sta   SHTempY,x
CFF9:60          (6)    595        rts
CFFA:                   596 *
CFFA:                   597 *
```



```
 C90B ACHE1          ?CD7E ALLSET          CD90 ALLSET1        ?CB01 ALTSENDPILE
 CDED AOKAY          CE3D AOKAYHITCH       CF98 ASSIGNID       ?CD56 ATENTRY
?FABA AUTOSCAN         56 AUXPTR          CB5E AUXPTRINC      ?  4E AUXTYPE
?  2D BADBLOCK         01 BADCMD        ?  22 BADCTLPARM        21 BADCTL
   04 BADPCNT          11 BADUNIT       ?E000 BASIC           C52F BC1
 CECD BEHITCH        CD51 BENTRY         CC29 BIT              0011 BMSGLEN
 C521 BOOTC         ?C514 BOOTCASE5      C523 BOOTCODE         C552 BOOTFAIL
 C55F BOOTMSG        07DB BOOTSCRN       C570 BOOTTAB            32 BSYTO1
   0A BSYTO2           54 BUFFER           56 BUFFER2          CE28 BUNIT
   06 BUSERR        ?  40 BUSHOG     ?  08 BYTECMP             4D BYTECOUNT
   4E BYTECOUNTH       4D BYTECOUNTL  ?C500 C500ORG           C082 CA1CLR
 C083 CA1SET         C084 CA2CLR         C085 CA2SET         CC2D CAREFUL
 CF0F CHKOFFLN    ?  24 CH               C8A2 CHAINUNBSY         40 CHECKSUM
 CFFF CLEARIOROMS    CA87 CLRPHASES          47 CMDBLOCKH    ?  46 CMDBLOCK
   46 CMDBLOCKL          48 CMDBLOCKS      45 CMDBUFFERH         44 CMDBUFFERL
   44 CMDBUFFER          42 CMDCODE        09 CMDLENGTH    ?C58A CMDLIST
   80 CMDMARK            43 CMDPCOUNT      46 CMDSCODE     ?  49 CMDSPARE1
?  4A CMDSPARE2          43 CMDUNIT        C55D COMA             80 COMMRESET
   04 CONTROLCMD       CDCA COPYLOOP       ?FDED COUT          CD48 CSERROR5
   10 CSUMERR       ?  25 CV               CD9C DARNIT             82 DATAMARK
 C996 DATDONE        C9C0 DBERROR        ?CBE2 DETTOPBITS   ?  50 DEVICEID
 CE38 DISABINT       CB52 DIV7TAB        CB9F DIVIDE1        CBA8 DIVIDE2
 CB8F DIVIDE3        CB96 DIVIDE4        CBB1 DIVIDE5       ?CB61 DIVIDE7
 CD0A DONE5          CE2C ENABINT        ?C08A ENABLE1         C08B ENABLE2
 CA7D ENABLECHAIN    CD4C ENTRY          CDBF ERRORHITCH     CF17 ERROR
 CE26 ERRORHITCH2    CF82 FINALSKIP   ?  03 FORMATCMD        CEE4 GETRESULTS
 CA44 GOB1          ?C9E3 GRABSTATUS       4B GRP7CTR          CD4A GSERROR5
 CBE9 GTBOB       ?  51 HOSTID          CD34 ICBT15          CD2E ICBT5
 CF8C ICK1             05 INITCMD          27 IOERROR          C080 IWM
   07 IWMMODE          C08C L6CLR          C08D L6SET          C08E L7CLR
 C08F L7SET       ?  68 LASTONE          CBD0 LASTPASS           00 LOC0
?  01 LOC1         ?CD37 LSTBSYWAIT5      C086 LSTRBCLR       C087 LSTRBSET
 C9E0 MARKERR        CE19 MAYBECTRL      CFDF MDEV1          CFD8 MDEV2
 C50D MLIENTRY       CB58 MOD7TAB        C088 MONCLR         C089 MONSET
 C554 MORCHRS        CFC4 MORDEVICES     07F8 MSLOT            4D NEXT1
   4E NEXT2              4F NEXT3            50 NEXT4            51 NEXT5
   52 NEXT6              53 NEXT7            4D NEXT         CE00 NIN1
   01 NOANSWER         CB7C NOAUXPTR       CEB2 NOCONTROL        28 NODRIVE
 CDC2 NOEH           CF15 NOERROR     ?  1F NOINT        ?  02 NOMARK
   20 NOPACKEND        CD6F NOPLAY         CDF2 NOTINIT        CE73 NOTSTAT
 CEBC NOWBLOCK       CED1 NOXTRASEND     CD44 NPENDERR5      06F9 NUMDEVICES
   4C ODDBYTES           2F OFFLINE        ?CDC8 OKAYCNT         CA79 ONEMS1
 CA77 ONEMS              C3 PACKETBEG        C8 PACKETEND        CF8E PARMCTAB
 C9BB PATCH1      ?  A5 PBBVALUE      ?  FF PBCVALUE            00 PCID2
   BF PDIDBYTE         CB4C PDIV7TAB       CB4F PMOD7TAB    ?  52 POINTER
   00 POWERRESET       C9D3 PREAMBLE      ?CBB1 PRECHECK       C50A PRODOSENTRY
 0473 PROFLAG        CD76 PZP            0BB8 RC1              05 RC2
 C583 RCODE            4B RCVBUF           CC51 RCVCOUNT       C9F5 RDH1
 C9FF RDH2           CA0D RDH3           CD3A RDH45         ?CA0B RDH5
 CD1F RDHA5            01 READCMD          C9E3 RECEIVEPACK    CB30 RECPACK
 C080 REQCLR         C081 REQSET         C576 RESET          CA5D RESETCHAIN
 04F3 RETRY          0573 RETRY2      ?  4F RPACKETTYPE      CB37 RPK1
 CB4B RPOUT          C578 RST1           CF68 RZP            CF39 SA2
 CB73 SAP1        ?  00 SCDEVICESTAT  ?  01 SCGETDCB     ?  03 SCGETDEVINFO
 0473 SCHOLES        C99A SCM1        ?  02 SCRETNLSTAT      C9CC SD10
 C9AF SD7            C9C6 SD9            CAE8 SDOUBT         CEC0 SECONDSEND
 CA4E SEND80         CA50 SENDBYTE       CADA SENDDATA       C883 SENDONEPACK
 CAE9 SENDPACK       CAFD SENDPILE       CC20 SETIWMODE     ?FE89 SETKBD
```

```
?FE93 SETVID      CA97 SETXN0     CA9A SHIFT1     CAAA SHIFT2
 CABA SHIFT3      CACA SHIFT4     0573 SHTEMP1    05F3 SHTEMPX
 0673 SHTEMPY     C924 SKIP1      C926 SKIP2      C960 SKIP3
 C962 SKIP4       CE40 SKIPCOPY   CC73 SLOTDEPRD    58 SLOT
 C8E5 SOB1        C8F6 SOB2       C8FD SOB3         40 SOFT
?  67 SOFTERROR   CB0A SPILE1     CB2F SPILOUT    CFF0 SQUIRREL
 C8AC SSB         C8AF SSD        ?0100 STACK     CA31 START0
 CA39 START1      CA4B START2     ?CC73 START25   CC7D START35
 C900 START         4D STATBYTE   ?  81 STATMARK     1E STATMTO
?  00 STATUSCMD   CF38 STOREAWAY  CC03 SUN2       ?CC01 SUN
 CC0C SUN1        CC1F SUN3       0778 SVBCH      06F8 SVBCL
?  10 SVMASK1     C797 SWPROTO    C784 SWRTS2     ?C9D4 SYNCTAB
 CC67 T71         CC70 T72          59 TBODD        59 TEMP
   60 THEOFF      CC5A TIMES7       41 TOPBITS    C896 UBSY1
   5A UNIT        ?0101 VERSION   ?FC22 VTAB       CC36 WAITIWMOFF
 CF4D WAITOFF     ?  04 WASRESET  ?C9DF WASTE12   C9DE WASTE14
?C9DD WASTE16     ?C9DC WASTE18   ?C9D9 WASTE32   CC3F WIWM1
 CC4C WIWM2         5B WPACKETTYPE   02 WRITECMD  CB61 WRITEPREP
   2B WRITEPROT   CBBD XOR1       ?CBBA XOR2      CBD6 XOR3
 CBDD XOR4        CBCE XOR5       CA70 YMSWAIT      40 ZEROPAGE
   1C ZPSIZE
```



```
?  00 STATUSCMD  ?  00 SCDEVICESTAT    00 PCID2          00 LOC0
   00 POWERRESET ?  01 LOC1            01 BADCMD      ?  01 SCGETDCB
   01 READCMD       01 NOANSWER     ?  02 NOMARK         02 WRITECMD
?  02 SCRETNLSTAT ?  03 FORMATCMD  ?  03 SCGETDEVINFO ?  04 WASRESET
   04 CONTROLCMD    04 BADPCNT         05 RC2            05 INITCMD
   06 BUSERR        07 IWMMODE     ?  08 BYTECMP        09 CMDLENGTH
   0A BSYTO2        10 CSUMERR     ?  10 SVMASK1      0011 BMSGLEN
   11 BADUNIT       1C ZPSIZE          1E STATMTO     ?  1F NOINT
   20 NOPACKEND     21 BADCTL      ?  22 BADCTLPARM  ?  24 CH
?  25 CV            27 IOERROR         28 NODRIVE        2B WRITEPROT
?  2D BADBLOCK      2F OFFLINE         32 BSYTO1         40 ZEROPAGE
?  40 BUSHOG        40 CHECKSUM        40 SOFT           41 TOPBITS
   42 CMDCODE       43 CMDPCOUNT       43 CMDUNIT        44 CMDBUFFER
   44 CMDBUFFERL    45 CMDBUFFERH      46 CMDSCODE    ?  46 CMDBLOCK
   46 CMDBLOCKL     47 CMDBLOCKH       48 CMDBLOCKS   ?  49 CMDSPARE1
?  4A CMDSPARE2     4B RCVBUF          4B GRP7CTR        4C ODDBYTES
   4D NEXT1         4D BYTECOUNTL      4D NEXT           4D STATBYTE
   4D BYTECOUNT     4E NEXT2           4E BYTECOUNTH  ?  4E AUXTYPE
?  4F RPACKETTYPE   4F NEXT3           50 NEXT4       ?  50 DEVICEID
?  51 HOSTID        51 NEXT5           52 NEXT6       ?  52 POINTER
   53 NEXT7         54 BUFFER          56 AUXPTR         56 BUFFER2
   58 SLOT          59 TBODD           59 TEMP           5A UNIT
   5B WPACKETTYPE   60 THEOFF      ?  67 SOFTERROR   ?  68 LASTONE
   80 COMMRESET     80 CMDMARK     ?  81 STATMARK       82 DATAMARK
?  A5 PBBVALUE      BF PDIDBYTE        C3 PACKETBEG      C8 PACKETEND
?  FF PBCVALUE    ?0100 STACK      ?0101 VERSION     0473 PROFLAG
 0473 SCHOLES     04F3 RETRY       0573 RETRY2       0573 SHTEMP1
 05F3 SHTEMPX     0673 SHTEMPY     06F8 SVBCL        06F9 NUMDEVICES
 0778 SVBCH       07DB BOOTSCRN    07F8 MSLOT        0BB8 RC1
 C080 REQCLR      C080 IWM         C081 REQSET       C082 CA1CLR
 C083 CA1SET      C084 CA2CLR      C085 CA2SET       C086 LSTRBCLR
 C087 LSTRBSET    C088 MONCLR      C089 MONSET      ?C08A ENABLE1
 C08B ENABLE2     C08C L6CLR       C08D L6SET        C08E L7CLR
 C08F L7SET      ?C500 C500ORG     C50A PRODOSENTRY  C50D MLIENTRY
?C514 BOOTCASE5   C521 BOOTC       C523 BOOTCODE     C52F BC1
 C552 BOOTFAIL    C554 MORCHRS     C55D COMA         C55F BOOTMSG
 C570 BOOTTAB     C576 RESET       C578 RST1         C583 RCODE
?C58A CMDLIST     C784 SWRTS2      C797 SWPROTO      C883 SENDONEPACK
 C896 UBSY1       C8A2 CHAINUNBSY  C8AC SSB          C8AF SSD
 C8E5 SOB1        C8F6 SOB2        C8FD SOB3         C900 START
 C90B ACHE1       C924 SKIP1       C926 SKIP2        C960 SKIP3
 C962 SKIP4       C996 DATDONE     C99A SCM1         C9AF SD7
 C9BB PATCH1      C9C0 DBERROR     C9C6 SD9          C9CC SD10
 C9D3 PREAMBLE   ?C9D4 SYNCTAB    ?C9D9 WASTE32     ?C9DC WASTE18
?C9DD WASTE16     C9DE WASTE14     ?C9DF WASTE12     C9E0 MARKERR
 C9E3 RECEIVEPACK ?C9E3 GRABSTATUS C9F5 RDH1        C9FF RDH2
?CA0B RDH5        CA0D RDH3        CA31 START0       CA39 START1
 CA44 GOB1        CA4B START2      CA4E SEND80       CA50 SENDBYTE
 CA5D RESETCHAIN  CA70 YMSWAIT     CA77 ONEMS        CA79 ONEMS1
 CA7D ENABLECHAIN CA87 CLRPHASES   CA97 SETXN0       CA9A SHIFT1
 CAAA SHIFT2      CABA SHIFT3      CACA SHIFT4       CADA SENDDATA
 CAE8 SDOUBT      CAE9 SENDPACK    CAFD SENDPILE    ?CB01 ALTSENDPILE
 CB0A SPILE1      CB2F SPILOUT     CB30 RECPACK      CB37 RPK1
 CB4B RPOUT       CB4C PDIV7TAB    CB4F PMOD7TAB     CB52 DIV7TAB
 CB58 MOD7TAB     CB5E AUXPTRINC   CB61 WRITEPREP   ?CB61 DIVIDE7
 CB73 SAP1        CB7C NOAUXPTR    CB8F DIVIDE3      CB96 DIVIDE4
 CB9F DIVIDE1     CBA8 DIVIDE2    ?CBB1 PRECHECK     CBB1 DIVIDE5
?CBBA XOR2        CBBD XOR1        CBCE XOR5         CBD0 LASTPASS
```

```
 CBD6 XOR3          CBDD XOR4          ?CBE2 DETTOPBITS    CBE9 GTBOB
?CC01 SUN           CC03 SUN2           CC0C SUN1          CC1F SUN3
 CC20 SETIWMODE     CC29 BIZ            CC2D CAREFUL       CC36 WAITIWMOFF
 CC3F WIWM1         CC4C WIWM2          CC51 RCVCOUNT      CC5A TIMES7
 CC67 T71           CC70 T72            CC73 SLOTDEPRD    ?CC73 START25
 CC7D START35       CD0A DONE5          CD1F RDHA5         CD2E ICBT5
 CD34 ICBT15       ?CD37 LSTBSYWAIT5    CD3A RDH45         CD44 NPENDERR5
 CD48 CSERROR5      CD4A GSERROR5       CD4C ENTRY         CD51 BENTRY
?CD56 ATENTRY       CD6F NOPLAY         CD76 PZP          ?CD7E ALLSET
 CD90 ALLSET1       CD9C DARNIT         CDBF ERRORHITCH    CDC2 NOEH
?CDC8 OKAYCNT       CDCA COPYLOOP       CDED AOKAY         CDF2 NOTINIT
 CE00 NIN1          CE19 MAYBECTRL      CE26 ERRORHITCH2   CE28 BUNIT
 CE2C ENABINT       CE38 DISABINT       CE3D AOKAYHITCH    CE40 SKIPCOPY
 CE73 NOTSTAT       CEB2 NOCONTROL      CEBC NOWBLOCK      CEC0 SECONDSEND
 CECD BEHITCH       CED1 NOXTRASEND     CEE4 GETRESULTS    CF0F CHKOFFLN
 CF15 NOERROR       CF17 ERROR          CF38 STOREAWAY     CF39 SA2
 CF4D WAITOFF       CF68 RZP            CF82 FINALSKIP     CF8C ICK1
 CF8E PARMCTAB      CF98 ASSIGNID       CFC4 MORDEVICES    CFD8 MDEV2
 CFDF MDEV1         CFF0 SQUIRREL       CFFF CLEARIOROMS  ?E000 BASIC
?FABA AUTOSCAN     ?FC22 VTAB          ?FDED COUT         ?FE89 SETKBD
?FE93 SETVID
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 15-JAN-84 21:28
** TOTAL LINES ASSEMBLED  2157
** FREE SPACE PAGE COUNT   70
```

```
SOURCE   FILE #01 =>ASM.S
 INCLUDE FILE #02 =>S.DIAG1.SRC

0000:                1           lst    on,vsym,asym
0000:                2           include s.diag1.src
```

```
02 S.DIAG1.SRC     slinky diagnostics                20-OCT-86  06:36 PAGE 2

0000:                2 ************************************************************
0000:                3 *
0000:                4 *                  Internal Slinky Diagnostics
0000:                5 *
0000:                6 ************************************************************

0000:                8 ************************************************************
0000:                9 * written by Eric Larson                    19 April 1985
0000:               10 * modified by Rich Williams                 09 May 1985
0000:               11 * put into //c rom by Ray Chiang            20 Feb 1986
0000:               12 ************************************************************

0000:               14 ************************************************************
0000:               15 * on entry:  y-reg has the value of sl.mslot (screen hole offset)
0000:               16 *           x-reg has the value of sl.devno (hardware offset)
0000:               17 *           card size is in numbanks,y
0000:               18 ************************************************************

0000:               20 ************************************************************
0000:               21 *                                 equates
0000:               22 ************************************************************
0000:       0042    23 mptr      equ   $42          ;indirect pointer to messages
0000:       0000    24 testnum   equ   $00
0000:       0045    25 compdata  equ   $45
0000:       0046    26 limit     equ   $46
0000:       0047    27 value     equ   $47
0000:       0049    28 loopcount equ   $49
0000:       0025    29 cv        equ   $25
0000:       00AE    30 dot       equ   $AE          ;ascii period
0000:       0007    31 bell      equ   $07          ;ascii bell
0000:       000D    32 cr        equ   $0D          ;ascii cr
0000:       001B    33 esc       equ   $1B          ;ascii esc

0000:       03B8    35 numbanks  equ   $478-$C0     ;number of 64K banks on card
0000:       0438    36 powerup   equ   $4F8-$C0     ;powerup byte
0000:       04B8    37 power2    equ   $578-$C0

0000:               39 * hardware equates, MUST be in $BF00 to avoid double access

0000:       BFF8    41 addrl     equ   $BFF8        ;address pointers
0000:       BFF9    42 addrm     equ   $BFF9        ;auto incs every data access
0000:       BFFA    43 addrh     equ   $BFFA        ;
0000:       BFFB    44 data      equ   $BFFB        ;data pointed to

0000:       C000    46 kbd       equ   $C000
0000:       C010    47 kstrobe   equ   $C010
0000:       FC42    48 clreop    equ   $FC42
0000:       FC58    49 home      equ   $FC58
0000:       FC9C    50 clreol    equ   $FC9C
0000:       FD8E    51 crout     equ   $FD8E
0000:       FDDA    52 prbyte    equ   $FDDA
0000:       FDED    53 cout      equ   $FDED
```

```
----- NEXT OBJECT FILE NAME IS ASM.S.0
2000:          2000   55           org   $2000
2000:                 56           MSB   OFF
2000:          2000   57 StartTest equ   *             ;entry point for self diagnostics
2000:A9 00            58           lda   #0
2002:85 49            59           sta   LoopCount     ;clear counter
2004:85 4A            60           sta   LoopCount+1
2006:99 38 04         61           sta   PowerUp,Y     ;marks card as having no directory
2009:99 B8 04         62           sta   Power2,Y
200C:B9 B8 03         63           lda   numbanks,Y    ;get result
200F:29 0F            64           and   #$0F
2011:85 46            65           sta   Limit
2013:20 58 FC         66           jsr   Home
2016:A9 08            67           lda   #8

2018:                 69 * "MEMORY CARD TEST<CR>ESC TO EXIT<CR>TEST WILL TAKE "

2018:20 1D 22         71           jsr   print
201B:A5 46            72           lda   Limit
201D:4A               73           lsr   A
201E:4A               74           lsr   A             ;divide by 4 (0-3)
201F:48               75           pha                  ;save size index
2020:09 04            76           ora   #4            ;0-3 --> 4-7
2022:20 1D 22         77           jsr   Print         ;45, 90, 135, or 180
2025:A9 09            78           lda   #9
2027:20 1D 22         79           jsr   Print         ;" SECONDS<CR>CARD SIZE = "
202A:68               80           pla                  ;size index
202B:20 1D 22         81           jsr   Print         ;256K, 512K, 768K, 1 MEG
202E:20 8E FD         82           jsr   Crout
```



```
2031:          2031   84 AddressTest equ *          ;read & write to address register
2031:A9 05            85           lda   #5
2033:85 25            86           sta   CV          ;cursor vertical position
2035:20 8E FD         87           jsr   Crout
2038:A9 10            88           lda   #$10
203A:20 1D 22         89           jsr   Print       ;"PASSES = "
203D:A5 4A            90           lda   LoopCount+1
203F:20 DA FD         91           jsr   PrByte
2042:A5 49            92           lda   LoopCount
2044:20 DA FD         93           jsr   PrByte
2047:20 0B 22         94           jsr   NxtLine
204A:A9 01            95           lda   #1
204C:85 00            96           sta   TestNum     ;start test number at 1
204E:A0 05            97           ldy   #5          ;index into data patterns
2050:B9 F4 22         98 at1       lda   Patterns,Y
2053:20 13 22         99           jsr   setaddr     ;Set address to pattern
2056:DD F8 BF        100           cmp   addrl,X     ;read register back
2059:D0 11   206C    101           bne   atf         ;they didn't match
205B:DD F9 BF        102           cmp   addrm,X
205E:D0 0C   206C    103           bne   atf
2060:09 F0           104           ora   #$F0        ;fill high 4 bits
2062:DD FA BF        105           cmp   addrh,X
2065:D0 05   206C    106           bne   atf
2067:88              107           dey                ;index to next pattern
2068:10 E6   2050    108           bpl   at1
206A:30 03   206F    109           bmi   RollOverTest
206C:4C 97 21        110 atf       jmp   Fail

206F:          206F  112 RollOverTest equ *          ;Test 2. Do address counters roll over
206F:E6 00           113           inc   TestNum     ;addrl, m, h = $FF from previous test
2071:DE F8 BF        114           dec   addrl,X     ;start with address $FFFFE.
2074:BD FB BF        115           lda   data,X      ;dec ok since $FF -> $FE doesn't carry
2077:9D FB BF        116           sta   data,X      ;address should now be $00000
207A:BD FA BF        117           lda   addrh,X
207D:29 0F           118           and   #$0F        ;mask off upper 4 bits
207F:1D F9 BF        119           ora   addrm,X
2082:1D F8 BF        120           ora   addrl,X
2085:F0 03   208A    121           beq   AddBusTest  ;address was indeed $00000
2087:4C 97 21        122           jmp   Fail
```

```
208A:                  124 ****************************************
208A:                  125 *
208A:                  126 * Walk a 1 through the address registers to test for bus shorts
208A:                  127 * assumes addresses = 0 from previous test
208A:                  128 *
208A:                  129 ****************************************
208A:          208A    130 AddBusTest equ  *           ;check for address buss shorts
208A:E6 00             131          inc   TestNum      ;Test 3
208C:A9 01             132          lda   #1
208E:85 45             133          sta   CompData
2090:8A                134          txa                ;Make pointer to addrl
2091:18                135          clc
2092:69 F8             136          adc   #>addrl
2094:85 42             137          sta   MPtr
2096:A9 C0             138          lda   #$C0
2098:85 43             139          sta   MPtr+1
209A:A5 46             140          lda   Limit        ;How many bits used in high address?
209C:F0 04    20A2     141          beq   ab1          ;If 1M then test D3210
209E:C9 0C             142          cmp   #$0C         ;If 768K then test D3210
20A0:D0 02    20A4     143          bne   ab2
20A2:A9 10             144 ab1      lda   #$10
20A4:4A                145 ab2      lsr   A            ;If 512K then test D210 if 256K then D10
20A5:48                146          pha                ;Save it for later
20A6:A0 02             147          ldy   #2           ;Walk a one thru high med and low address
20A8:48                148 ab3      pha
20A9:20 11 22          149          jsr   clraddr      ;Clear address in case of false carries
20AC:68                150          pla
20AD:91 42             151          sta   (MPtr),y     ;Store pattern in address
20AF:48                152          pha
20B0:A5 45             153          lda   CompData     ;get value to store
20B2:9D FB BF          154          sta   data,x
20B5:E6 45             155          inc   CompData     ;Each address gets a different value
20B7:68                156          pla                ;Get pattern back
20B8:4A                157          lsr   A            ;Move the 1 over. $80 -> $40 etc.
20B9:D0 ED    20A8     158          bne   ab3          ;Until all bits tested
20BB:91 42             159          sta   (MPtr),y     ;Zero out current byte
20BD:6A                160          ror   A            ;0 -> $80
20BE:88                161          dey
20BF:10 E7    20A8     162          bpl   ab3          ;Loop through all 3 address registers
20C1:A9 01             163          lda   #1           ;Now read em all back
20C3:85 45             164          sta   CompData
20C5:68                165          pla                ;Get start value for high byte
20C6:A0 02             166          ldy   #2
20C8:48                167 ab4      pha
20C9:20 11 22          168          jsr   clraddr      ;Clear address in case of false carry
20CC:68                169          pla
20CD:91 42             170          sta   (MPtr),y     ;Set address
20CF:85 47             171          sta   Value        ;Don't pha since we might abort
20D1:BD FB BF          172          lda   data,x
20D4:C5 45             173          cmp   CompData     ;Right data?
20D6:D0 0F    20E7     174          bne   abfail
20D8:E6 45             175          inc   CompData
20DA:A5 47             176          lda   Value
20DC:4A                177          lsr   A
20DD:D0 E9    20C8     178          bne   ab4
20DF:91 42             179          sta   (MPtr),y
20E1:6A                180          ror   A            ;0 -> $80
20E2:88                181          dey
```



```
20E3:10 E3    20C8     182          bpl   ab4
20E5:30 03    20EA     183          bmi   ClearTest
20E7:4C 97 21          184 abFail   jmp   Fail

20EA:                  186 ************************************************
20EA:                  187 *
20EA:                  188 ************************************************

20EA:          20EA    190 ClearTest equ  *           ;see if all locations clear to zero
20EA:20 11 22          191          jsr   clraddr      ;Set address and A to 0
20ED:          20ED    192 FillTest  equ  *           ;Loop & see if all locations fill to ones
20ED:E6 00             193          inc   TestNum      ;Test 4 = 00s.  Test 5 = FFs
20EF:85 45             194          sta   CompData     ;value to fill RAM with
20F1:A5 45             195 f1       lda   CompData
20F3:9D FB BF          196          sta   Data,X       ;write data out
20F6:9D FB BF          197          sta   Data,X
20F9:9D FB BF          198          sta   Data,X
20FC:9D FB BF          199          sta   Data,X
20FF:BD F8 BF          200          lda   addrl,X
2102:D0 ED    20F1     201          bne   f1
2104:1D F9 BF          202          ora   addrm,X       ;are addrl & addrm both zero?
2107:D0 E8    20F1     203          bne   f1           ;no, keep going
2109:20 E4 21          204          jsr   PrDot        ;Z = 1 if done
210C:D0 E3    20F1     205          bne   f1           ;no, keep going
210E:20 0B 22          206          jsr   NxtLine      ;Go to next line and clear address
2111:BD FB BF          207 cp1      lda   Data,X       ;read data back
2114:C5 45             208          cmp   CompData
2116:D0 CF    20E7     209          bne   abFail       ;Failed if ne
2118:BD FB BF          210          lda   Data,X       ;do 2 per loop for speed
211B:C5 45             211          cmp   CompData
211D:D0 C8    20E7     212          bne   abFail
211F:BD F8 BF          213          lda   addrl,X
2122:D0 ED    2111     214          bne   cp1
2124:1D F9 BF          215          ora   addrm,X       ;are addrl & addrm both zero?
2127:D0 E8    2111     216          bne   cp1          ;no, keep going
2129:20 E4 21          217          jsr   PrDot        ;Z = 1 if done
212C:D0 E3    2111     218          bne   cp1          ;no, keep going
212E:20 0B 22          219          jsr   NxtLine      ;Go to next line and clear address
2131:A5 45             220          lda   CompData
2133:49 FF             221          eor   #$FF         ;0 -> FF
2135:D0 B6    20ED     222          bne   FillTest
```

```
2137:        2137  Computed  equ   *            ;each byte gets computed value
2137:E6 00                   inc   TestNum      ;Test 6
2139:A9 55                   lda   #$55         ;Starting data pattern
213B:85 45                   sta   CompData     ;Address left at 0 from last test
213D:20 FD 21        c1      jsr   getvalue     ;Value = addrm + addrh + $55. A = 0
2140:18              c2      clc
2141:65 47                   adc   Value
2143:65 45                   adc   CompData
2145:9D FB BF                sta   data,x
2148:85 45                   sta   CompData     ;Save for next add
214A:BD F8 BF                lda   addrl,x
214D:D0 F1   2140            bne   c2
214F:BD F9 BF                lda   addrm,x      ;Time to print a dot?
2152:D0 E9   213D            bne   c1
2154:20 E4 21                jsr   PrDot        ;Z = 1 if done
2157:D0 E4   213D            bne   c1
2159:20 0B 22                jsr   NxtLine      ;Go to next line and clear address
215C:A9 55                   lda   #$55         ;Starting data pattern
215E:85 45                   sta   CompData
2160:20 FD 21        c3      jsr   getvalue     ;Now read em back
2163:18              c4      clc
2164:65 47                   adc   Value
2166:65 45                   adc   CompData
2168:85 45                   sta   CompData
216A:BD FB BF                lda   data,x
216D:C5 45                   cmp   CompData     ;Is it right?
216F:D0 26   2197            bne   Fail
2171:BD F8 BF                lda   addrl,x
2174:D0 ED   2163            bne   c4
2176:BD F9 BF                lda   addrm,x      ;Time to print a dot?
2179:D0 E5   2160            bne   c3
217B:20 E4 21                jsr   PrDot        ;Z = 1 if done
217E:D0 E0   2160            bne   c3
```



```
2180:        2180  Pass      equ   *            ;passed all the tests
2180:A9 0B                   lda   #$0B
2182:20 1D 22                jsr   Print        ;"CARD OK"
2185:F8                      sed
2186:A5 49                   lda   LoopCount
2188:18                      clc
2189:69 01                   adc   #1
218B:85 49                   sta   LoopCount
218D:A5 4A                   lda   LoopCount+1
218F:69 00                   adc   #0
2191:85 4A                   sta   LoopCount+1
2193:D8                      cld
2194:4C 31 20                jmp   AddressTest  ;loop until first failure
2197:                *
2197:        2197  Fail      equ   *            ;display failure message
2197:48                      pha                 ;save actual data
2198:20 42 FC                jsr   ClrEop
219B:A9 0A                   lda   #$0A
219D:20 1D 22                jsr   Print        ;"CARD FAILED!<CR>"
21A0:A5 00                   lda   TestNum
21A2:C9 03                   cmp   #3
21A4:B0 09   21AF            bcs   DataErr      ;not an addressing problem
21A6:68                      pla                 ;there is no failling data really
21A7:A9 0C                   lda   #$0C
21A9:20 1D 22                jsr   Print        ;"ADDRESS ERROR"
21AC:4C DE 21                jmp   ErrCommon
21AF:A9 0D           DataErr lda   #$0D
21B1:20 1D 22                jsr   Print        ;"DATA ERROR "
21B4:38                      sec
21B5:BD F8 BF                lda   addrl,X
21B8:E9 01                   sbc   #1           ;set back to actual failing value
21BA:48                      pha
21BB:BD F9 BF                lda   addrm,X
21BE:E9 00                   sbc   #0           ;propagate borrows (if any)
21C0:48                      pha
21C1:BD FA BF                lda   addrh,X
21C4:29 0F                   and   #$0F         ;mask off high 4 bits
21C6:E9 00                   sbc   #0
21C8:20 DA FD                jsr   PrByte       ;print as two hex digits
21CB:68                      pla
21CC:20 DA FD                jsr   PrByte       ;print addrm as two hex digits
21CF:68                      pla
21D0:20 DA FD                jsr   PrByte       ;print addrl as two hex digits
21D3:A9 0E                   lda   #$0E
21D5:20 1D 22                jsr   Print        ;" - "
21D8:68                      pla                 ;actual data
21D9:45 45                   eor   CompData
21DB:20 DA FD                jsr   PrByte       ;print failing data as two hex digits
21DE:A9 0F           ErrCommon lda #$0F
21E0:20 1D 22                jsr   Print        ;"<CR>SEE DEALER FOR SERVICE<CR>"
21E3:60                      rts
21E4:                *
21E4:                *
21E4:                *
21E4:        21E4  PrDot     equ   *
21E4:A9 AE                   lda   #Dot
21E6:20 ED FD                jsr   Cout
21E9:AD 00 C0                lda   Kbd          ;Is escape pressed?
```

```
21EC:C9 9B          316           cmp   #ESC+$80
21EE:D0 05   21F5   317           bne   noesc
21F0:68             318           pla                  ;Pop current return address
21F1:68             319           pla
21F2:8D 10 C0       320           sta   KStrobe
21F5:        21F5   321 noesc     equ   *
21F5:BD FA BF       322           lda   addrh,x        ;Test if last dot
21F8:29 0F          323           and   #$0F
21FA:C5 46          324           cmp   Limit          ;Z = 1 if last dot
21FC:60             325           rts
21FD:        21FD   326 getvalue  equ   *
21FD:18             327           clc
21FE:BD F9 BF       328           lda   addrm,x
2201:7D FA BF       329           adc   addrh,x
2204:69 55          330           adc   #$55
2206:85 47          331           sta   Value
2208:A9 00          332           lda   #0
220A:60             333           rts
220B:        220B   334 NxtLine   equ   *              ;Go to next line and clear address
220B:20 8E FD       335           jsr   Crout
220E:20 9C FC       336           jsr   ClrEol
2211:               337 * fall into clraddr
2211:        2211   338 clraddr   equ   *              ;Clears the address registers
2211:A9 00          339           lda   #0
2213:        2213   340 setaddr   equ   *              ;Sets the address registers to A
2213:9D F8 BF       341           sta   addrl,x        ;Must do in this order
2216:9D F9 BF       342           sta   addrm,x        ;to avoid false carry
2219:9D FA BF       343           sta   addrh,x
221C:60             344           rts

221D:               346 ************************************************
221D:               347 *
221D:               348 ************************************************

221D:        221D   350 Print     equ   *              ;print message to the screen
221D:A8             351           tay
221E:B9 30 22       352           lda   Messages,Y
2221:A8             353           tay
2222:B9 41 22       354 pr1       lda   M0,Y
2225:48             355           pha
2226:09 80          356           ora   #$80           ;all characters must have high bit set
2228:20 ED FD       357           jsr   Cout
222B:C8             358           iny                   ;index to next character
222C:68             359           pla
222D:10 F3   2222   360           bpl   pr1            ;last character had high bit set
222F:60             361           rts
2230:        2230   362 Messages  equ   *              ;table of pointers to actual messages
2230:00             363           dfb   M0-M0
2231:05             364           dfb   M1-M0
2232:09             365           dfb   M2-M0
2233:0D             366           dfb   M3-M0
2234:11             367           dfb   M4-M0
2235:14             368           dfb   M5-M0
2236:16             369           dfb   M6-M0
2237:18             370           dfb   M7-M0
2238:1B             371           dfb   M8-M0
2239:47             372           dfb   M9-M0
223A:5C             373           dfb   M0A-M0
```



```
223B:6D             374           dfb   M0B-M0
223C:77             375           dfb   M0C-M0
223D:84             376           dfb   M0D-M0
223E:8F             377           dfb   M0E-M0
223F:92             378           dfb   M0F-M0
2240:AA             379           dfb   M10-M0
2241:31 20 4D 45    380 M0        dci   "1           MEG"
2246:32 35 36 CB    381 M1        dci   "256K"
224A:35 31 32 CB    382 M2        dci   "512K"
224E:37 36 38 CB    383 M3        dci   "768K"
2252:31 38 B0       384 M4        dci   "180"
2255:34 B5          385 M5        dci   "45"
2257:39 B0          386 M6        dci   "90"
2259:31 33 B5       387 M7        dci   "135"
225C:4D 45 4D 4F    388 M8        asc   "MEMORY      CARD TEST"
226C:0D             389           dfb   CR
226D:45 53 43 20    390           asc   "ESC         TO EXIT"
2278:0D             391           dfb   CR
2279:54 45 53 54    392           dci   "TEST        WILL TAKE "
2288:20 53 45 43    393 M9        asc   "           SECONDS"
2290:0D             394           dfb   CR
2291:43 41 52 44    395           dci   "CARD        SIZE = "
229D:0D 0D          396 M0A       dfb   CR,CR
229F:43 41 52 44    397           asc   "CARD        FAILED"
22AA:0D 07 07 87    398           dfb   CR,BELL,BELL,BELL+128
22AE:0D 0D          399 M0B       dfb   CR,CR
22B0:43 41 52 44    400           asc   "CARD        OK"
22B7:8D             401           dfb   CR+128
22B8:41 44 44 52    402 M0C       dci   "ADDRESS     ERROR"
22C5:44 41 54 41    403 M0D       dci   "DATA        ERROR "
22D0:20 2D A0       404 M0E       dci   "           - "
22D3:0D             405 M0F       dfb   CR
22D4:53 45 45 20    406           asc   "SEE         DEALER FOR SERVICE"
22EA:8D             407           dfb   CR+128
22EB:50 41 53 53    408 M10       dci   "PASSES      = "
22F4:FF CC AA 55    409 Patterns  dfb   $FF,$CC,$AA,$55,$33,$00 ;data buss patterns
22FA:52 69 63 68    410           asc   'Rich        Williams'
2307:63 6F 70 79    411           asc   'copyright   1986 Apple Computer Inc.'
2329:61 6C 6C 20    412           asc   'all         rights reserved'
233C:        233C   413 zsld.end  equ   *
233C:        00C3   414           ds    $23FF-*,0      ;make sure not too big
```

```
 20A2 AB1            20A4 AB2            20A8 AB3            20C8 AB4
 20E7 ABFAIL         208A ADDBUSTEST     2031 ADDRESSTEST    BFFA ADDRH
 BFF8 ADDRL          BFF9 ADDRM          2050 AT1            206C ATF
   07 BELL           213D C1             2140 C2             2160 C3
 2163 C4             20EA CLEARTEST      2211 CLRADDR        FC9C CLREOL
 FC42 CLREOP           45 COMPDATA      ?2137 COMPUTED        FDED COUT
 2111 CP1            FD8E CROUT            0D CR               25 CV
 BFFB DATA           21AF DATAERR          AE DOT            21DE ERRCOMMON
   1B ESC            20F1 F1             2197 FAIL           20ED FILLTEST
 21FD GETVALUE       FC58 HOME           C000 KBD            C010 KSTROBE
   46 LIMIT            49 LOOPCOUNT      229D M0A            22AE M0B
 22B8 M0C            22D0 M0E            2241 M0             22C5 M0D
 22D3 M0F            2246 M1             22EB M10            224A M2
 224E M3             2252 M4             2255 M5             2257 M6
 2259 M7             225C M8             2288 M9             2230 MESSAGES
   42 MPTR           21F5 NOESC          03B8 NUMBANKS       220B NXTLINE
?2180 PASS           22F4 PATTERNS       04B8 POWER2         0438 POWERUP
 2222 PR1            FDDA PRBYTE         21E4 PRDOT          221D PRINT
 206F ROLLOVERTEST   2213 SETADDR       ?2000 STARTTEST         00 TESTNUM
   47 VALUE          ?233C ZSLD.END
```



```
   00 TESTNUM          07 BELL             0D CR               1B ESC
   25 CV               42 MPTR             45 COMPDATA         46 LIMIT
   47 VALUE            49 LOOPCOUNT        AE DOT            03B8 NUMBANKS
 0438 POWERUP        04B8 POWER2         ?2000 STARTTEST     2031 ADDRESSTEST
 2050 AT1            206C ATF            206F ROLLOVERTEST   208A ADDBUSTEST
 20A2 AB1            20A4 AB2            20A8 AB3            20C8 AB4
 20E7 ABFAIL         20EA CLEARTEST      20ED FILLTEST       20F1 F1
 2111 CP1           ?2137 COMPUTED       213D C1           ?2180 PASS
 2160 C3             2163 C4             2197 FAIL           21AF DATAERR
 21DE ERRCOMMON      21E4 PRDOT          21F5 NOESC          21FD GETVALUE
 2211 CLRADDR        2213 SETADDR        221D PRINT          2222 PR1
 2230 MESSAGES       2241 M0             224A M2             224E M3
 2252 M4             2255 M5             2257 M6             2259 M7
 225C M8             2288 M9             229D M0A            22AE M0B
 22B8 M0C            22C5 M0D            22D0 M0E            22D3 M0F
 22EB M10            22F4 PATTERNS      ?233C ZSLD.END       BFF8 ADDRL
 BFF9 ADDRM          BFFA ADDRH          BFFB DATA           C000 KBD
 C010 KSTROBE        FC42 CLREOP         FC58 HOME           FC9C CLREOL
 FD8E CROUT          FDDA PRBYTE         FDED COUT
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 15-JAN-84 21:28
** TOTAL LINES ASSEMBLED   417
** FREE SPACE PAGE COUNT   82
```

Glossary

accumulator: The register in the 65C02 microprocessor where most computations are performed.

ACIA: Acronym for *Asynchronous Communications Interface Adapter.* A single **chip** that converts data from parallel to serial form and vice versa. An ACIA handles serial transmission and reception and RS-232-C signals under the control of its internal registers, which can be set and changed by **firmware** or **software.**

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

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

address: A number that specifies the location of a single **byte** of memory. Addresses can be given as decimal integers or as hexadecimal integers. A 64K system has addresses ranging from 0 to 65535 (in decimal) or from $0000 to $FFFF (in hexadecimal).

algorithm: A step-by-step procedure for solving a problem or accomplishing a task.

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

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

analog data: Data in the form of continuously variable quantities. Compare **digital data.**

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

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

AND: A logical operator that produces a true result if both its operands are true, and a false result if either or both its operands are false. Compare **OR, NOT, exclusive OR.**

ANSI: Acronym for *American National Standards Institute,* which sets standards for many technical fields and is the most common standard for computer terminals.

Apple I: The first Apple computer. It was built in a garage in California by Steve Jobs and Steve Wozniak.

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

Apple III: An Apple computer; part of the Apple II family. The Apple III offered a built-in disk drive and built-in RS-232-C (serial) port. Its memory was expandable to 256K.

Apple II: A family of computers, including the original Apple II, the Apple II Plus, the Apple IIe, the Apple IIc, and the Apple IIGS. The original Apple II used Integer BASIC instead of Applesoft BASIC, and it required a keyboard command (PR#6) in order to start up from a disk.

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

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

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

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

Apple IIGS: A powerful new member of the Apple II family. The Apple IIGS uses a 16-bit microprocessor and has 256K of RAM. It has slots like the Apple IIe and ports like the Apple IIc, and contains a 15-voice custom sound chip.

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

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

application program: A program written for some specific purpose, such as word processing, data base management, graphics, or telecommunication. Compare **system program.**

argument: A value on which a function or statement operates; it can be a number or a variable. For example, in the BASIC statement `VTAB 10`, the number 10 is the argument. Compare **operand.**

arithmetic expression: A combination of numbers and arithmetic operators (such as 3 + 5) that indicates some operation to be carried out.

arithmetic operator: An operator, such as +, that combines numeric values to produce a numeric result. Compare **logical operator, relational operator.**

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

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

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

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

asynchronous transmission: A method of data transmission in which the receiving and sending devices don't share a common timer, and no timing data is transmitted. Each information character is individually synchronized, usually by the use of start and stop bits. The time interval between characters isn't necessarily fixed. Compare **synchronous transmission.**

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

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

bandwidth: The range of frequencies a device can handle. Bandwidth and maximum data transfer rate are directly proportional. For example, a video monitor's greater bandwidth allows it to display more information per scan frame than most home television sets can. To display 80 columns of text, a monitor should have a bandwidth of at least 12 MHz.

base address: In *indexed addressing,* the fixed component of an address.

BASIC: Acronym for *Beginners All-purpose Symbolic Instruction Code.* BASIC is a high-level programming language designed to be easy to learn. Two versions of BASIC are available from Apple Computer for use with all Apple II–family systems: Applesoft BASIC (built into the firmware) and Integer BASIC.

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

binary: Characterized by having two different components, or by having only two alternatives or values available; sometimes used synonymously with **binary system.**

binary digit: The smallest unit of information in the binary number system; a 0 or a 1. Also called a **bit.**

binary operator: An operator that combines two operands to produce a result. For example, + is a binary arithmetic operator; < is a binary relational operator; OR is a binary logical operator. Compare **unary operator.**

binary system: The representation of numbers in the base-2 system, using only the two digits 0 and 1. For example, the numbers 0, 1, 2, 3, and 4 become 0, 1, 10, 11, and 100 in binary notation. The binary system is commonly used in computers because the values 0 and 1 can easily be represented in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the display screen. A single binary digit—a 0 or a 1—is called a **bit.** Compare **decimal, hexadecimal.**

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

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

bits per second: See **bit rate.**

board: See **printed-circuit board.**

body: In BASIC, the statements or instructions that make up a part of a program, such as a loop or a subroutine.

boot: Another way to say **start up.** A computer boots by loading a program into memory from an external storage medium such as a disk. Starting up is often accomplished by first loading a small program, which then reads a larger program into memory. The program is said to "pull itself up by its own bootstraps"—hence the term *bootstrapping* or *booting.*

boot disk: See **startup disk.**

bootstrap: See **boot.**

bps: See **bit rate.**

branch: (v) To pass program control to a line or statement other than the next in sequence. (n) A statement that performs a branch. See **conditional branch, unconditional branch.**

BREAK: A SPACE (0) signal, sent over a communication line, of long enough duration to interrupt the sender. This signal is often used to end a session with a time-sharing service. BREAK is also used in BASIC to stop execution of a program; it's generated by pressing Control-C.

BRK: A "software interrupt." An instruction that causes the 6502 or 65C02 microprocessor to halt. Pronounced "break."

buffer: A "holding area" of the computer's memory where information can be stored by one program or device and then read at a different rate by another; for example, a print buffer. In editing functions, an area in memory where deleted (cut) or copied data is held. In some applications, this area is called the *Clipboard.*

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

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

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

cable: An insulated bundle of wires with connectors on the ends; the number of wires varies with the type of connection. Examples are serial cables, disk drive cables, and AppleTalk cables.

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

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

carrier: The background signal on a communication channel that is modified to carry information. Under RS-232-C rules, the carrier signal is equivalent to a continuous MARK (1) signal; a transition to 0 then represents a start bit.

carry flag: A status bit in the 6502 or 65C02 microprocessor, used as a ninth bit with the eight accumulator bits in addition, subtraction, rotation, and shift operations.

cathode-ray tube (CRT): An electronic device, such as a television picture tube, that produces images on a phosphor-coated screen. The phosphor coating emits light when struck by a focused beam of electrons. A common display device used with personal computers.

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

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

character code: A number used to represent a character for processing by a computer system.

character set: The entire set of characters that can be either shown on a monitor or used to code computer instructions. In a printer, the entire set of characters that the printer is capable of printing.

chip: See **integrated circuit.**

Clear To Send: An RS-232-C signal from a DCE to a DTE that is normally kept false until the DCE makes it true, indicating that all circuits are ready to transfer data out. See **Data Communication Equipment, Data Terminal Equipment.**

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

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

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

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

command character: An ASCII character, usually Control-A or Control-I, that causes the serial port firmware to interpret subsequent characters as commands.

command register: An ACIA location (at $C09A for port 1 and $C0AA for port 2) that stores parity type and RS-232-C signal characteristics.

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

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

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

computer language: See **programming language.**

computer system: A computer and its associated hardware, firmware, and software.

conditional branch: A **branch** whose execution depends on the truth of a condition or the value of an expression. Compare **unconditional branch.**

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

connector: A plug, socket, jack, or port.

constant: In a program, a symbol that represents a fixed, unchanging value. Compare **variable.**

control character: A nonprinting character that controls or modifies the way information is printed or displayed. In the Apple II family, control characters have ASCII values between 0 and 31, and are typed from a keyboard by holding down the Control key while pressing some other key. In the Macintosh family, the Command key performs a similar function.

control code: One or more nonprinting characters—included in a text file—whose function is to change the way a printer prints the text. For example, a program may use certain control codes to turn boldface printing on and off. See **control character.**

control key: A general term for a key that controls the operation of other keys; for example, Apple, Caps Lock, Control, Option, and Shift. When you hold down or engage a control key while pressing another key, the combination makes that other key behave differently. Also called a *modifier key.*

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

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

control register: An ACIA location (at $C09B for port 1 and $C0AB for port 2) that stores data format and baud rate selections.

Control-Reset: A combination keystroke on Apple II–family computers that usually causes an Applesoft BASIC program or command to stop immediately. If a program disables the Control-Reset feature, you need to turn the computer off to get the program to stop.

copy protect: To make a disk uncopyable. Software publishers frequently try to copy protect their disks to prevent them from being illegally duplicated by software pirates. Compare **write protect.**

CPU: See **central processing unit.**

CRT: See **cathode-ray tube.**

CTS: See **Clear To Send.**

crash: To cease to operate unexpectedly, possibly destroying information in the process.

current input device: The source, such as the keyboard or a modem, from which a program is currently receiving its input.

current output device: The destination, such as the display screen or a printer, currently receiving a program's output.

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

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

data: Information, especially information used or operated on by a program. The smallest unit of information a computer can understand is a **bit.**

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

Data Carrier Detect (DCD): An RS-232-C signal from a DCE (such as a modem) to a DTE (such as an Apple IIe) indicating that a communication connection has been established. See **Data Communication Equipment, Data Terminal Equipment.**

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

data format: The form in which data is stored manipulated, or transferred.

data set: A device that modulates, demodulates, and controls signals transferred between business machines and communication facilities. A form of **modem.**

Data Set Ready (DSR): An RS-232-C signal from a DCE to a DTE indicating that the DCE has established a connection. See **Data Communication Equipment, Data Terminal Equipment.**

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

Data Terminal Ready (DTR): An RS-232-C signal from a DTE to a DCE indicating a readiness to transmit or receive data. See **Data Communication Equipment, Data Terminal Equipment.**

DCD: See **Data Carrier Detect.**

DCE: See **Data Communication Equipment.**

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

decimal: The common form of number representation used in everyday life, in which numbers are expressed in in the base-10 system, using the ten digits 0 through 9. Compare **binary, hexadecimal.**

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

deferred execution: The execution of a BASIC program instruction that is part of a complete program. The program instruction is executed only when the complete program is run. You defer execution of the instruction by preceding it with a program line number. The complete program executes consecutive instructions in numerical order. Compare **immediate execution.**

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

delimiter: A character that is used for punctuation to mark the beginning or end of a sequence of characters, and which therefore is not considered part of the sequence itself. For example, Applesoft BASIC uses the double quotation mark (") as a delimiter for string constants: the string `"DOG"` consists of the three characters *D, O,* and *G,* and does not include the quotation marks.

demodulate: To recover the information being transmitted by a modulated signal. For example, a conventional radio receiver demodulates an incoming broadcast signal to convert it into the sound emitted by the radio's speaker. Compare **modulate.**

device: Frequently used as a short form of **peripheral device.**

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

device handler: See **device driver.**

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

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

digital data: Data that can be represented by digits—that is, data that are discrete rather than continuously variable. Compare **analog data.**

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

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

DIP: See **dual in-line package.**

DIP switches: A bank of tiny switches, each of which can be moved manually one way or the other to represent one of two values (usually on and off). See **dual in-line package.**

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

disk: An information-storage medium consisting of a flat, circular, magnetic surface on which information can be recorded in the form of small magnetized spots, in a manner similar to the way sounds are recorded on tape. See **floppy disk, hard disk.**

disk-based: See **disk-resident.**

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

disk drive: The device that holds a disk, retrieves information from it, and saves information to it.

disk envelope: A removable, protective paper sleeve used when handling or storing a 5.25-inch disk. It must be removed before you insert the disk in a disk drive. Compare **disk jacket.**

disk jacket: A permanent, protective covering for a disk. 5.25-inch disks have flexible, paper or plastic jackets; 3.5-inch disks have hard plastic jackets. The disk is never removed from the jacket. Compare **disk envelope.**

Disk Operating System (DOS): An optional software system for the Apple II family of computers that enables the computer to control and communicate with one or more disk drives. The acronym *DOS* rhymes with *boss.*

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

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

display: (1) A general term to describe what you see on the screen of your display device when you're using a computer; from the verb form, which means "to place into view." (2) Short for a display device.

display color: The color currently being used to draw high-resolution or low-resolution graphics on the display screen.

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

display screen: The screen of the monitor; the area where you view text and pictures when using the computer.

DOS 3.2: An early Apple II operating system. DOS stands for **Disk Operating System;** 3.2 is the version number. Disks formatted using DOS 3.2 have 13 sectors per track.

DOS 3.3: An operating system used by the Apple II family of computers. DOS stands for **Disk Operating System;** 3.3 is the version number. Disks formatted with DOS 3.3 have 16 sectors per track.

drive: See **disk drive.**

DSR: See **Data Set Ready.**

DTE: See **Data Terminal Equipment.**

DTR: See **Data Terminal Ready.**

dual in-line package (DIP): An integrated circuit packaged in a narrow rectangular box with a row of metal pins along each side. **DIP switches** on the box allow you to change settings. For example, ImageWriter printer DIP switches control functions such as line feed, form length, and **baud** setting.

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

EBCDIC: Acronym for *Extended Binary-Coded Decimal Interchange Code;* pronounced "EB-si-dik." A code used by IBM that represents each letter, number, special character, and control character as an 8-bit binary number. EBCDIC has a character set of 256 8-bit characters. Compare **ASCII.**

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

80-column text card: A peripheral card that allows the Apple II, Apple II Plus, and Apple IIe to display text in either 40 columns or 80 columns.

80/40-column switch: A switch that controls the maximum number of columns or characters across the screen. A television can legibly display a maximum of 40 characters across the screen, whereas a video monitor can display 80 characters.

embedded: Contained within. For example, the string `'HUMPTY DUMPTY'` is said to contain an embedded space.

emulate: To operate in a way identical to a different system. For example, the Apple II 2780/3780 Protocol Emulator and the Apple II 3270 BSC Protocol Emulator, together with the Apple Communications Protocol Card (ACPC), allow the Apple II, Apple II Plus, or Apple IIe to emulate the operations of IBM 3278 and 3277 terminals and 3274 and 3271 control units.

emulation mode: A mode of operation in which the computer is emulating the operation of another computer or interface. See **emulate.**

end-of-command mark: A punctuation mark used to separate commands sent to a peripheral device such as a printer or plotter. Also called a *command terminator.*

end-of-line character: A character which indicates that the preceding text constitutes a full line.

error code: A number or other symbol representing a type of error.

error message: A message displayed or printed to tell you of an error or problem in the execution of a program or in your communication with the system. An error message is often accompanied by a beep.

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

escape code: A sequence of characters that begins with an ESCAPE character and constitutes a complete command. Usually synonymous with **escape sequence.**

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

escape mode: A state of the Apple IIe and IIc entered by pressing the Esc key and certain other keys. The other keys take on special meanings for positioning the cursor and controlling the display of text on the screen.

escape sequence: A sequence of keystrokes, beginning with the Esc key. In **escape mode,** escape sequences are used for positioning the cursor and controlling the display of text on the screen. Escape sequences are also used as codes to control printers.

Esc key: See **Escape key.**

even/odd parity check: In data transmission, a check that tests whether the number of 1 bits in a group of binary digits is even (even parity check) or odd (odd parity check).

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

exclusive OR: A logical operator that produces a true result if one of its operands is true and the other false, and a false result if its operands are both true or both false. Compare **OR, AND,** and **NOT.**

execute: To perform the actions specified by a program command or sequence of commands.

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

expression: A formula in a program that defines a calculation to be performed.

FIFO: Acronym for "first in, first out" order, as in a **queue.**

file: Any named, ordered collection of information stored on a disk. Application programs and operating systems on disks are files. You make a file when you create text or graphics, give the material a name, and save it to disk.

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

fixed-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position within the number. Typically, the point is considered to lie at the right end of the number so that the number is interpreted as an **integer.** Compare **floating-point.**

flag: A variable whose value (usually 1 or 0, standing for *true* or *false*) indicates whether some condition holds or whether some event has occurred. A flag is used to control the program's actions at some later time.

floating-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is permitted to "float" to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Compare **fixed-point.**

floppy disk: A **disk** made of flexible plastic, as compared to a **hard disk,** which is made of metal. The term *floppy* is now usually applied only to disks with thin, flexible **disk jackets,** such as 5.25-inch disks. With 3.5-inch disks, the disk itself is flexible, but the jacket is made of hard plastic; thus, 3.5-inch disks aren't particularly "floppy."

format: (n) (1) The form in which information is organized or presented. (2) The general shape and appearance of a printed page, including page size, character width and spacing, line spacing, and so on. (v) To divide a disk into tracks and sectors where information can be stored. Blank disks must be formatted before you can save information on them for the first time; same as **initialize.**

form feed: An ASCII character (decimal 12) that causes a printer or other paper-handling device to advance to the top of the next page.

Fortran: Short for *Formula Translator.* A high-level programming language especially suitable for applications requiring extensive numerical calculations, such as in mathematics, engineering, and the sciences.

framing error: In serial data transfer, the absence of the expected stop bit(s) at the end of a received character.

frequency: In alternating current (AC) signals, the number of complete cycles transmitted per second. Frequency is usually expressed in hertz (cycles per second), kilohertz (kilocycles per second), or megahertz (megacycles per second). In acoustics, frequency of vibration determines musical pitch. Compare **duration.**

full duplex: A four-wire communication circuit or protocol that allows two-way data transmission between two points at the same time. Compare **half duplex.**

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

game I/O connector: A 16-pin connector inside the Apple II, II Plus, and IIe, originally designed for connecting hand controls to the computer, but also used for connecting some other peripheral devices. Compare **hand control connector.**

graph: A pictorial representation of data.

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

half duplex: A two-wire communication circuit or protocol designed for data transmission in either direction but not both directions simultaneously. Compare **full duplex.**

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

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

hang: To cease operation because either an expected condition is not satisfied or an infinite loop is occurring. A computer that's hanging is called a *hung system.* Compare **crash.**

hard disk: A disk made of metal and sealed into a drive or cartridge. A hard disk can store very large amounts of information compared to a **floppy disk.**

hard disk drive: A device that holds a hard disk, retrieves information from it, and saves information to it. Hard disks made for microprocessors are permanently sealed into the drives.

hardware: In computer terminology, the machinery that makes up a computer system. Compare **firmware, software.**

hertz: The unit of frequency of vibration or oscillation, defined as the number of *cycles per seoond.* Named for the physicist Heinrich Hertz and abbreviated *Hz.* The 6502 microprocessor used in the Apple II systems operates at a clock frequency of about 1 million hertz, or 1 megahertz (MHz). The 68000 microprocessor used in the Macintosh operates at 7.8336 MHz.

hexadecimal: The representation of numbers in the base-16 system, using the ten digits 0 through 9 and the six letters A through F. For example, the decimal numbers 0, 1, 2, 3, 4, ... 8, 9, 10, 11, ... 15, 16, 17 would be shown in hexadecimal notation as 00, 01, 02, 03, 04, ... 08, 09, 0A, 0B, ... 0F, 10, 11. Hexadecimal numbers are easier for people to read and understand than are binary numbers, and they can be converted easily and directly to binary form. Each hexadecimal digit corresponds to a sequence of four **binary digits,** or bits. Hexadecimal numbers are usually preceded by a dollar sign ($).

high ASCII characters: ASCII characters with decimal values of 128 to 255. Called *high ASCII* because their high bit (first binary digit) is set to 1 (for *on*) rather than 0 (for *off*).

high-level language: A programming language that is relatively easy for people to understand. A single statement in a high-level language typically corresponds to several instructions of machine language. High-level languages available from Apple Computer include BASIC, Pascal, Instant Pascal, Logo, Pilot, SuperPILOT, and Fortran. Compare **low-level language.**

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

high-resolution graphics: The display of graphics on a screen as a six-color array of points, 280 columns wide and 192 rows high. When a text window is in use, the visible high-resolution graphics display is 280 by 160 points.

hold time: In computer circuits, the amount of time a signal must remain valid after some related signal has been turned off. Compare **setup time.**

Hz: See **hertz.**

IC: See **integrated circuit.**

immediate execution: The execution of a program statement as soon as it is typed. In BASIC, immediate execution occurs when the line is typed without a line number; immediate execution allows you to try out nearly every statement immediately to see how it works. Compare **deferred execution.**

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

IN#: This command designates the source of subsequent input characters. It can be used to designate a device in a slot or a machine-language routine as the source of input.

index: (1) A number used to identify a member of a list or table by its sequential position. (2) A list or table whose entries are identified by sequential position. (3) In machine-language programming, the variable component of an indexed address, contained in an index register and added to the base address to form the effective address.

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

index register: A register in a computer processor that holds an index for use in indexed addressing. The 6502 microprocessor used in the Apple II family of computers has two index registers, called the **X register** and the **Y register.** The 68000 microprocessor used in Macintosh-family computers has 16 registers that can be used as index registers.

index variable: A variable whose value changes on each pass through a loop. Often called *control variable* or *loop variable.*

infinite loop: A section of a program that will repeat the same sequence of actions indefinitely.

initialize: (1) To set to an initial state or value in preparation for some computation. (2) To prepare a blank disk to receive information by organizing its surface into tracks and sectors; same as **format.**

initialized disk: A disk that has been organized into tracks and sectors by the computer and is therefore ready to store information.

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

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

input routine: A machine-language routine; the standard input routine reads characters from the keyboard. A different input routine might, for example, read them from an external terminal.

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

integer: A whole number in fixed-point form. Compare **real number.**

Integer BASIC: A version of the BASIC programming language used by the Apple II family of computers. Integer BASIC is older than Applesoft BASIC and is capable of processing numbers in integer (fixed-point) form only. Many games are written in Integer BASIC because its instructions can be executed very quickly. Compare **Applesoft BASIC.**

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

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

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

interpreter: A language translator that reads a program instruction by instruction and immediately translates each instruction for the computer to carry out. Compare **compiler.**

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

inverse video: The display of text on the computer's display screen in the form of dark dots on a light (or other single phosphor color) background, instead of the usual light dots on a dark background.

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

I/O device: Input/output device. A device that transfers information into or out of a computer. See **input, output, peripheral device.**

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

IWM: "Integrated Woz Machine"; the custom chip that controls Apple's 3.5-inch disk drives.

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

K: See **kilobyte.**

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

keyboard input connector: The connector inside the Apple II family of computers by which the keyboard is connected to the computer.

keyword: A special word or sequence of characters that identifies a particular type of statement or command, such as *RUN, BRUN,* or *PRINT.*

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

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

language: See **programming language.**

language card: A peripheral card that, when placed in slot 0 of a 48K Apple II or Apple II Plus, gives the computer a total of 64K of memory. If you have an Apple II or Apple II Plus, you need a language card or the equivalent to use ProDOS.

language translator: A system program that reads another program written in a particular programming language and either executes it directly or converts it into some other language (such as machine language) for later execution. See **interpreter, compiler, assembler.**

leading zero: A zero occurring at the beginning of a decimal number, deleted by most computing programs.

least significant bit: The rightmost bit of a binary number. The least significant bit contributes the smallest quantity to the value of the number. Compare **most significant bit.**

LIFO: Acronym for "last in, first out" order, as in a **stack.**

line: See **program line.**

line feed: An ASCII character (decimal 10) that ordinarily causes a printer or video display to advance to the next line.

line number: A number identifying a program line in an Applesoft BASIC program.

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

list: To display on a monitor, or print on a printer, the contents of memory or of a file.

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

local: Connected to or close by the host system.

location: See **memory location.**

logic: (1) In microcomputers, a mathematical treatment of formal logic using a set of symbols to represent quantities and relationships that can be translated into switching circuits, or *gates.* AND, OR, and NOT are examples of logical gates. Each gate has two states, open or closed, allowing the application of **binary** numbers for solving problems. (2) The systematic scheme that defines the interactions of signals in the design of an automatic data processing system.

logical operator: An operator, such as AND, tha combines logical values to produce a logical result, such as true or false; sometimes called a *Boolean operator.* Compare **arithmetic operator, relational operator.**

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

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

loop variable: See **index variable.**

low-level language: A programming language that is relatively close to the form the computer's processor can execute directly. One statement in a low-level language corresponds to a single machine-language instruction. Examples are 6502 machine language, 6502 assembly language, and 68000 machine and assembly languages. Compare **high-level language.**

low-order byte: The less significant half of a memory address or other two-byte quantity. In the 6502 microprocessor used in the Apple II family of computers, the low-order byte of an address is usually stored first, and the **high-order byte** second. The opposite is true for Macintosh computers.

low-power Schottky (LS): A type of **transistor-transistor logic (TTL)** integrated circuit having lower power and higher speed than a conventional TTL integrated circuit; named for Walter Schottky (1886–1956), a semiconductor physicist.

low-resolution graphics: The display of graphics on a display screen as a 16-color array of blocks, 40 columns wide and 48 rows high. For example, on a Macintosh when the text window is in use, the visible low-resolution graphics display is 40 by 40 plotting points—that is, 40 by 40 **pixels.** See **high-resolution graphics.**

LS: See **low-power Schottky.**

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

mainframe computer: A central processing unit or computer that is larger and more powerful than a minicomputer or a personal computer (microcomputer). Frequently called simply a *mainframe* for short.The Apple Access II program and MacTerminal make it possible to communicate with mainframe computers over telecommunications media.

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

main memory: The part of a computer's memory whose contents are directly accessible to the microprocessor; usually synonymous with **random-access memory (RAM).** Programs are loaded into main memory, and that's where the computer keeps information while you're working. Sometimes simply called *memory.* See also **read-only memory, read-write memory.**

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

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

memory: A hardware component of a computer system that can store information for later retrieval. See **main memory, random-access memory, read-only memory, read-write memory.**

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

memory-resident: (1) Stored permanently in memory as firmware (ROM). (2) Held continually in memory even while not in use. DOS is a memory-resident program.

menu: A list of choices presented by a program, from which you can select an action.

MHz: Megahertz; one million hertz. See **hertz.**

microcomputer: A computer, such as any of the Apple II or Macintosh computers, whose processor is a **microprocessor.**

microprocessor: A computer **processor** contained in a single integrated circuit, such as the 6502 or 65C02 microprocessor used in the Apple II family of computers and the 68000 microprocessor used in the Macintosh family. The microprocessor is the **central processing unit (CPU)** of the microcomputer.

microsecond: One millionth of a second. Abbreviated µs.

millisecond: One thousandth of a second. Abbreviated ms.

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

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

modifier key: A key (Apple, Caps Lock, Control, Option, Shift) that generates no keyboard events of its own, but changes the meaning of other keys or mouse actions. Also called a *control key.*

modulate: To modify or alter a signal so as to transmit information. For example, conventional broadcast radio transmits sound by modulating the amplitude (amplitude modulation, or *AM*) or the frequency (frequency modulation, or *FM*) of a carrier signal.

monitor: See **video monitor.**

Monitor program: A system program built into the firmware of some computers, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level. The Monitor program activates the disk drive when you turn on the computer.

most significant bit: The leftmost bit of a binary number. The most significant bit contributes the largest quantity to the value of the number. For example, in the binary number 10110 (decimal value 22), the leftmost bit has the decimal value 16 (2^4). Compare **least significant bit.**

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

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

nanosecond: One billionth of a second. Abbreviated ns.

nested loop: A loop contained within the body of another loop and executed repeatedly during each pass through the outer loop. See **loop.**

nested subroutine call: A call to a subroutine from within the body of another subroutine.

nibble: A unit of data equal to half a byte, or four bits. A nibble can hold any value from 0 to 15.

NOT: A unary logical operator that produces a true result if its operand is false, and a false result if its operand is true. Compare **AND, OR, exclusive OR.**

NTSC: (1) Abbreviation for *National Television Standards Committee.* The committee that defined the standard format used for transmitting broadcast video signals in the United States. (2) The standard video format defined by the NTSC.

object code: See **object program.**

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

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

opcode: See **operation code.**

Open Apple: A **control key** on the Apple II–family keyboards; on later keyboards, simply called the *Apple key.*

operand: A value to which an operator is applied. The value on which an operation code operates. Compare **argument.**

operating system: A program that organizes the actions of the parts of the computer and its peripheral devices. See **disk operating system.**

operation code: The part of a machine-language instruction that specifies the operation to be performed. Often called *opcode.*

operator: A symbol or sequence of characters, such as + or AND, specifying an operation to be performed on one or more values (the operands) to produce a result. See **arithmetic operator, relational operator, logical operator, unary operator, binary operator.**

option: (1) Something chosen or available as a choice; for instance, items in a menu. (2) An **argument** whose provision is optional.

OR: A logical operator that produces a true result if either or both of its operands are true, and a false result if both of its operands are false. Compare **exclusive OR, AND, NOT.**

output: Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem.

output routine: A machine-language routine that performs the sending of characters. The standard output routine sends characters to the screen. A different output routine might, for example, send them to a printer.

overflow: The condition that exists when an attempt is made to put more data into a given memory area than it can hold; for example, a computational result that exceeds the allowed range.

override: To modify or cancel an instruction by issuing another one.

overrun: A condition that occurs when the processor does not retrieve a received character from the receive data register of the Asynchronous Communications Interface Adapter (ACIA) before the subsequent character arrives. The ACIA automatically sets bit 2 (OVR) of its status register; subsequent characters are lost. The receive data register contains the last valid data word received.

page: (1) A screenful of information on a video display. In the Apple II family of computers, a page consists of 24 lines of 40 or 80 characters each. (2) An area of main memory containing text or graphical information being displayed on the screen. (3) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256.

page zero: See **zero page.**

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

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

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

pass: A single execution of a loop.

PC board: See **printed-circuit board.**

peek: To read information directly from a location in the computer's memory.

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

peripheral bus: The **bus** used for transmitting information between the computer and peripheral devices connected to the computer's expansion slots or ports.

peripheral card: A removable printed-circuit board that plugs into one of the computer's expansion slots. Peripheral cards allow the computer to use peripheral devices or to perform some subsidiary or peripheral function.

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

peripheral slot: See **expansion slot.**

phase: (1) A stage in a periodic process. A point in a cycle. For example, the 6502 microprocessor uses a clock cycle consisting of two phases called Φ 0 and Φ 1. (2) The relationship between two periodic signals or processes.

PILOT: Acronym for *Programmed Inquiry, Learning, Or Teaching.* A high-level programming language designed for teachers and used to create computer-aided instruction (CAI) lessons that include color graphics, sound effects, lesson text, and answer checking. SuperPILOT is an enhanced version of the original Apple II PILOT programming language.

pipelining: A feature of a processor that enables it to begin fetching the next instruction before it has finished executing the current instruction. All else being equal, processors with this feature run faster than those without it.

pixel: Short for *picture element.* A point on the graphics screen; the visual representation of a bit on the screen (white if the bit is 0, black if it's 1). Also, a location in video memory that maps to a point on the graphics screen when the viewing window includes that location.

plotting vector: A code representing a single step in drawing a shape on the high-resolution graphics screen. The plotting vector specifies whether to plot a point at the current screen position, and in what direction to move (up, down, left, or right) before processing the next vector. See **shape definition, shape table.**

pointer: An item of information consisting of the memory address of some other item. For example, Applesoft BASIC maintains internal pointers to the most recently stored variable, the most recently typed program line, and the most recently read data item, among other things. The 6502 uses one of its internal registers as a pointer to the top of the stack.

point of call: The point in a program from which a subroutine or function is called.

poke: To store information directly into a location in the computer's memory.

pop: To remove the top entry from a **stack,** moving the stack pointer to the entry below it. Synonymous with *pull.* Compare **push.**

port: In the Apple IIc, slots are called ports.

power supply: A circuit that draws electrical power from a power outlet and converts it to the kind of power the computer can use.

power supply case: The metal case inside most Apple II and Macintosh computers that houses the power supply. The Apple IIc uses an external power supply case.

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

precedence: The order in which operators are applied in evaluating an expression. Precedence varies from language to language, but usually resembles the precedence rules of algebra.

printed-circuit board: A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly Fiberglas, to which integrated circuits and other electronic components are connected.

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

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

ProDOS: An Apple II operating system designed to support hard disk drives like the ProFile, as well as floppy disk storage devices. ProDOS stands for *Professional Disk Operating System*. Compare **Disk Operating System (DOS).**

ProDOS command: Any one of the 28 commands recognized by ProDOS.

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

program line: The basic unit of an Applesoft BASIC program, consisting of one or more statements separated by colons (:).

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

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

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

prompt line: A specific area on the display reserved for **prompts.**

protocol: A formal set of rules for sending and receiving data on a communication line.

Protocol Converter: A set of machine language routines used in the Apple II family for performing block device I/O. See **Smartport.**

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

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

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

radio-frequency (RF) modulator: A device that makes your television set work as a monitor.

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

random-access memory (RAM): Memory in which information can be referred to in an arbitrary or random order. As an analogy, a book is a random-access storage device in that it can be opened and read at any point. RAM usually means the part of memory available for programs from a disk; the programs and other data are lost when the computer is turned off. A computer with 512K RAM has 512 kilobytes available to the user. (Technically, the read-only memory [ROM] is also *random access,* and what's called RAM should correctly be termed *read-write memory.*) Compare **read-only memory, read-write memory.**

random-access text file: A text file that is partitioned into an unlimited number of uniform-length compartments called *records.* When you open a random-access text file for the first time, you must specify its record length. No record is placed in the file until written to. Each record can be individually read from or written to—hence, *random-access.*

raster: The pattern of parallel lines making up the image on a video display screen. The image is produced by controlling the brightness of successive points on the individual lines of the raster.

read: To transfer information into the computer's memory from outside the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

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

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

real number: In computer usage, a number that may include a fractional part; represented inside the computer in **floating-point** form. Because a real number is of infinite precision, this representation is usually approximate. Compare **integer.**

receive data register: A read-only register in the serial port ACIA (at $C098 for port 1 and $C0A8 for port 2) that stores the most recent character successfully received.

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

relational operator: An operator, such as >, that operates on numeric values to produce a logical result. Compare **arithmetic operator, logical operator.**

Request-To-Send: An RS-232-C signal from a DTE to a DCE that serves to prepare the DCE for data transmission.

reserved word: A word or sequence of characters reserved by a programming language for some special use and therefore unavailable as a variable name in a program.

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

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

RF modulator: See **radio-frequency modulator.**

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

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

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

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

RS-232 cable: Any cable that is wired in accordance with the RS-232 standard, which is the common serial data communication interface standard.

RTS: See **Request-To-Send.**

run: (1) To execute a program. When a program *runs,* the computer performs the instructions. (2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

save: To store information by transferring the information from main memory to a disk. Work not saved disappears when you turn off the computer or when the power is interrupted.

screen: See **display screen.**

scroll: To move all the text on the screen upward or downward, and, in some cases, sideways. See **viewport, window.**

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

setup time: The amount of time a signal must be valid in advance of some event. Compare **hold time.** See **valid signal.**

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

simple variable: A variable that is not an element of an array.

68000: The microprocessor used in the Macintosh and Macintosh Plus.

6502: The microprocessor used in the Apple II, in the Apple II Plus, and in early models of the Apple IIe.

65C02: The microprocessor used in the enhanced Apple IIe, the extended keyboard IIe, and the Apple IIc.

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

Smartport: A set of machine language routines used in the Apple II family for performing block device I/O. See **Protocol Converter.**

soft switch: Also called a *software switch;* a means of changing some feature of the computer from within a program. For example, **DIP switch** settings on ImageWriter printers can be overridden with soft switches. Specifically, a soft switch is a location in memory that produces some special effect whenever its contents are read or written.

software: A collective term for **programs,** the instructions that tell the computer what to do. They're usually stored on disks. Compare **hardware, firmware.**

source code: See **source program.**

source program: The form of a program given to a language translator, such as a compiler or assembler, for conversion into another form; sometimes called *source code.* Compare **object program.**

space character: A text character whose printed representation is a blank space, typed from the keyboard by pressing the Space bar.

SPACE parity: A bit value of 0 appended to a binary number for transmission. The receving device can look for this value on each character as a means of error checking.

stack: A list in which entries are added (pushed) or removed (popped) at one end only (the top of the stack), causing them to be removed in last-in, first-out (LIFO) order. Compare **queue.**

standard instruction: An instruction automatically present when no superseding instruction has been received.

start bit: A transition from a MARK signal to a SPACE signal for one bit-time, indicating that next string of bits represents a character.

starting value: The value assigned to the index variable on the first pass through a loop.

start up: To get the system running. Starting up is the process of first reading the operating system program from the disk, and then running an application program.

startup disk: A disk with all the necessary program files—such as the Finder and System files contained in the System folder in Macintosh—to set the computer into operation. In Apple II, sometimes called a *boot disk.*

statement: A unit of a program in a high-level language that specifies an action for the computer to perform. A statement typically corresponds to several instructions of machine language.

status register: A location in the ACIA (at $C099 for port 1 and $C0A9 for port 2) that stores the state of two RS-232-C signals and the state of the transmit and receive data registers, as well as the outcome of the most recent character transfer.

step value: The amount by which the index variable changes on each pass through a loop.

stop bit: A MARK signal following a data string (or the optional parity bit), indicating the end of a character.

string: An item of information consisting of a sequence of text characters.

strobe: A signal whose change is used to trigger some action.

subroutine: A part of a program that can be executed on request from another point in the program and that returns control, on completion, to the point of the request.

synchronous: A mode of data transmission in which a constant time interval exists between transmission of successive bits, characters, or events. Compare **asynchronous.**

synchronous transmission: A transmission process that uses a clocking signal to ensure an integral number of unit (time) intervals between any two characters. Compare **asynchronous transmission.**

syntax: (1) The rules governing the structure of statements or instructions in a programming language. (2) A representation of a command that specifies all the possible forms the command can take.

system: A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose—for example, a computer system comprising a processor, keyboard, monitor, and disk drive.

system configuration: See **configuration.**

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

system software: The component of a computer system that supports application programs by managing system resources such as memory and I/O devices.

tab: An ASCII character that commands a device such as a printer to start printing at a preset location (called a *tab stop*). There are two such characters: horizontal tab (hex 09) and vertical tab (hex 0B). TAB works like the tabs on a typewriter.

television set: A display device capable of receiving broadcast video signals (such as commercial television broadcasts) by means of an antenna. Can be used in combination with a radio-frequency modulator as a display device for the Apple II family of computers. Compare **video monitor.**

text: (1) Information presented in the form of readable characters. (2) The display of characters on a display screen. Compare **graphics.**

text window: An area on the video display screen within which text is displayed and scrolled.

traces: Electrical paths that connect the components on a circuit board.

transistor-transistor logic (TTL): (1) A family of integrated circuits having bipolar circuit logic; TTLs are used in computers and related devices. (2) A standard for interconnecting such circuits, which defines the voltages used to represent logical zeros and ones.

transmit data register: A location in the ACIA (at location $C098 for port 1 and $C0A8 for port 2) that holds the current character to be transmitted.

troubleshoot: To locate and correct the cause of a problem or malfunction, especially in hardware. Compare **debug.**

TTL: See **transistor-transistor logic.**

turnkey disk: See **startup disk.**

unary operator: An operator that applies to a single operand. For example, the minus sign (–) in a negative number such as –6 is a unary arithmetic operator. Compare **binary operator.**

unconditional branch: A branch that does not depend on the truth of any condition. Compare **conditional branch.**

value: An item of information that can be stored in a variable, such as a number or a string.

variable: (1) A location in the computer's memory where a value can be stored. (2) The symbol used in a program to represent such a location. Compare **constant.**

vector: (1) The starting address of a program segment, when used as a common point for transferring control from other programs. (2) A memory location used to hold a vector, or the address of such a location.

video: (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form.

video monitor: A display device that can receive video signals by direct connection only, and that cannot receive broadcast signals such as commercial television. Can be connected directly to the computer as a display device. Compare **television set.**

viewport: All or part of the display screen used by an application program to display a portion of the information (such as a document, picture, or worksheet) on which a program is working. Compare **window.**

volume: A general term referring to a storage device; a source of or a destination for information. A volume has a name and a volume directory with the same name. Its information is organized into files.

warm start: The process of transferring control back to the operating system in response to a failure in an application program. Compare **cold start.**

window: The portion of a collection of information (such as a document, picture, or worksheet) that is visible in a viewport on the display screen. Compare **viewport.**

word: A group of bits that is treated as a unit; the number of bits in a word is a characteristic of each particular computer.

write: To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

write-enable notch: The square cutout on one edge of a 5.25-inch disk's jacket. If there is no write-enable notch, or if it is covered with a write-protect tab, the disk drive can read information from the disk, but cannot write on it.

write protect: To protect the information on a 5.25-inch disk by covering the write-enable notch with a write-protect tab, preventing the disk drive from writing any new information onto the disk. Compare **copy protect.**

write-protect tab: (1) A small adhesive sticker used to write protect a 5.25-inch disk by covering the write-enable notch. (2) The small plastic tab i the corner of a 3.5-inch disk jacket. You lock (writ protect) the disk by sliding the tab toward the edg of the disk; you unlock the disk by sliding the tab back so that it covers the rectangular hole.

X register: One of the two index registers in the 6502 microprocessor.

Y register: One of the two index registers in the 6502 microprocessor.

zero page: The first page (256 bytes) of memory in the Apple II family of computers, also called *page zero*. Since the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; thi makes zero-page locations more efficient to address, in both time and space, than locations i any other page of memory.

Bibliography

Addendum to the Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1984.

Applesoft BASIC Programmer's Reference Manual, Vols. 1 and 2. For the Apple II, IIe, and IIc. Cupertino, Calif.: Apple Computer, Inc., 1982. The version that applies to both the Apple IIe and the Apple IIc has Apple product number A2L0084 (Vol. 1) and A2L0085 (Vol. 2).

Applesoft Tutorial. Cupertino, Calif.: Apple Computer, Inc., 1982.

Apple IIe Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1982.

Apple II Monitors Peeled. Cupertino, Calif.: Apple Computer, Inc., 1978. Currently not updated for Apple IIe and IIc, but a good introduction to Apple II–series input/output procedures; also useful for historical background.

Leventhal, Lance. *6502 Assembly Language Programming.* Berkeley, Calif.: Osborne/McGraw-Hill, 1979.

Synertek Hardware. Santa Clara, Calif.: Synertek Incorporated, 1976. Does not contain instructions new to 65C02, but is the only currently available manufacturer's hardware manual for 6500-series microcomputers.

Synertek Programming. Santa Clara, Calif.: Synertek Incorporated, 1976. The only currently available manufacturer's programming manual for 6500-series microcomputers.

Watson, Allen, III. "A Simplified Theory of Video Graphics, Part I." *Byte,* Vol. 5, No. 11 (November 1980).

———. "A Simplified Theory of Video Graphics, Part II." *Byte,* Vol. 5, No. 12 (December 1980).

———. "More Colors for Your Apple." *Byte, Vol.* 4, No. 6 (June 1979).

———. "True Sixteen-Color Hi-Res." *Apple Orchard,* Vol. 5, No. 1 (January 1984).

Wozniak, Steve. "System Description: The Apple II." *Byte,* Vol. 2, No. 5 (May 1977).

———. "SWEET16: The 6502 Dream Machine." *Byte,* Vol. 2, No. 10 (October 1977).

Index

Cast of Characters

* (asterisk) 59, 104
@ (at sign) 113
\ (backslash) 59, 63
^ (caret) 225
: (colon) 216, 224
$ (dollar) 225
. (period) 206
? (question mark) 59, 169, 179
_ (underscore) 179

A

accumulator 18, 64, 69, 84, 90, 115
ACIA
  block diagram 276
  command register 280
  control register 278–279
  register locations for port 1 159
  register locations for port 2 173
  status register 130, 281
  transmit/receive register 282
acoustic coupler 177
addresses
  Applesoft BASIC interpreter 326
  display 259–261
  firmware 322–327
  hardware 316–321, 353–356
  I/O link 56–58
  memory 20
  mouse port 325
  port 323–325
  RAM 22, 351
  ROM 22–23, 250, 351–352
  serial port 323–324
  video display 101
  video firmware 324
addressing modes 26, 226
  65C02 302, 304
AltChar switch 102, 243, 360
alternate character set 69, 70, 88, 92, 360
AltZP switch 28, 242
analog inputs 200
any-key-down (AKD) 78, 243, 255
Apple Integer BASIC 308, 330, 348, 356, 388
Apple Language Card 351
Apple Logo II 330
Applesoft BASIC 169, 308, 329, 388
  mouse and 195–196
Applesoft BASIC interpreter 14, 23, 36, 51, 52, 59, 63, 204, 214, 250, 352
  addresses 326
Apple IIc
  block diagram 235–236
  schematic diagrams 291–296
Apple II computers, interrupts 332
Apple II series
  differences 348–365
  disk I/O 361
  hardware 365
  I/O 357
  keyboard 357–359
  machine identification 350
  memory structure 351–356
  video display 359–360
A register 18, 43, 84, 113–115, 192, 213
arithmetic, hexadecimal 215
arrow keys 4
ASCII character set 3, 70, 78, 80–81, 381–382, 391–395
ASCII codes 58, 78, 80–81, 89, 164, 368
ASCII input mode 209
assembler 220
assembly language, mouse and, 195–196
assembly-language programs, debugging 221
asterisk (*) 59, 204
asynchronous communications interface adapter 130
at sign (@) 113
AUD 256, 365
audio output jack 8, 256
automatic repeat 3, 358
Autostart ROM 356
auxiliary memory 42–44, 74, 106, 160–161, 175, 269
  screen holes 315
auxiliary RAM 22, 184

B

back panel 9–10
backslash (\) 59, 63
backspace 63, 114
BadBlock $2D error 151
BadCmd $01 error 150
BadCtl $21 error 150
BadCtlParm $22 error 150
BadPCnt $04 error 150
BadUnit $11 error 150
Bank2 switch 241
bank selector switches 27–35
bank-switched memory 24–35
BASIC command 228
baud rate
  serial port 1 163
  serial port 2 177–178

bell 114
bell character 115
Bell routine 84
Bell1 routine 84
bits 384–386
blanking intervals 257
block device I/O firmware, entry points 23
block-type devices 120
BREAK signal 163
BRK ($00) instructions 212, 221, 334
  handling 337–338
buffers
  display 38, 99
  input 36, 38
  serial I/O 343–345, 362
BusErr $06 error 150
button interrupt mode 188
bytes 384–386

C

CALL -151 command 223
Canadian keyboard 375
cancel line 63
Caps Lock 5, 81, 358, 360
caret (^) 225
carriage return (CR) 63, 114, 164, 179
carry bit 43
cassette I/O 364
C command 156, 170
CH (cursor horizontal) 64
character generator 14, 263
  control signals 266
character output switch (CSW) 57, 64, 71, 84, 101, 113, 115
characters
  at sign (@) 113
  command 155, 205
  control 4, 5, 60, 65–67, 70, 114, 165, 392
  flashing 69, 70, 88–89
  inverse 69, 70, 88–89
  lowercase 395
  normal 69, 70, 88–89
  prompt 59
  special 393
  uppercase 394
character sets 358–360
  alternate 69, 70, 88, 92, 360
  ASCII 3, 70, 78, 80–81, 381–382, 391–395
  display 89
  MouseText 91
  primary 69, 70, 88, 359
  screen 6
  text 88–89
ClampMouse routine 194
ClearMouse routine 193
ClEOLZ routine 112, 113
clock rate 237
clock signals 239–241
CLOSE call 127, 141–142
ClrEOL routine 112, 113
ClrEOP routine 112, 113
ClrScr routine 112, 113
ClrTop routine 112, 113
CmdNum 125
CMOS 237
code conversions 391
cold start 51, 121–122
colon (:) 216, 224
colors
  double high-resolution graphics 99
  high-resolution 97, 268
  low-resolution 94, 266
column-address strobe (CAS) 252
command character 155, 205
command number 125, 127
command register, ACIA 280
connectors 9–10
  disk drive 274
  external power 234
  hand controller 199–200, 287
  keyboard 254
  mouse 187, 284
  serial port 154, 278
  video expansion 270
  video output 270
CONTINUE BASIC command 228
Control 5, 81, 255, 358
Control-\ 67
Control-[67
Control-] 67
Control-_ 67
Control-A 169, 171, 172, 179, 362
Control-A I 181, 184
Control-A Q 184
Control-A T 180–182, 184
Control-B 214, 228
Control-C 67, 204, 214, 228, 341
CONTROL call 127, 136–139
control characters 4, 5, 60, 65–67, 70, 114, 165, 392
Control-D 155, 169
Control-E 213, 228
Control-G 65, 66, 84
Control-H 63, 65, 66
Control-I 155, 158, 165, 362
Control-J 65, 66
Control-K 57, 66, 228
Control-L 66
control list 137, 138
Control-M 65, 66, 118
Control-N 66
Control-O 66, 90
Control-P 57, 90, 215, 228
Control-Q 66
Control-R 66, 172, 181, 184
control register ACIA, 278–279
Control-Reset 4, 49, 51, 52, 81, 121, 123, 162, 171, 176, 204, 218
Control-S 66, 67, 341
Control-T 171
Control-U 63, 66, 82
Control-V 66, 158, 172
Control-W 66, 158
Control-X 60, 63, 67, 82
Control-Y 67, 218, 228
Control-Z 67
COut routine 63, 64, 90, 112, 113, 214
COut1 routine 56, 60, 65, 67, 68, 70, 112, 114
CP/M 328
CPU 13
CR *See* carriage return
CROut routine 112, 114
CROut1 routine 112, 114
CSW 57, 64, 71, 84, 101, 113, 155
CSWH 57
CSWL 57
C3COut1 routine 56, 61, 62, 65–67, 68, 70
C3KeyIn routine 56, 58, 61, 62

cursor
  blinking question mark 169, 179
  blinking underscore 179
  mouse 187
  movement keys 4, 62
CV (cursor vertical) 64

D

Data Carrier Detect 281
data format
  serial port 1 163
  serial port 2 177–178
data inputs 23
Data Set Ready (DSR) 281
Data Terminal Ready (DTR) 280
data transfer 42–43
debugging 212, 221
decimal 387–388
  negative 388–389
Delete 4, 358
device control block (DCB) 130, 138
device information block (DIB) 131
DevSpec $30–$3F error 151
DHiRes switch 46, 102, 103
disassembler 220
disk controller unit 15, 247–248, 273, 365
disk drive 8–9
  connector 274
  connector signals 274
  I/O 120–121, 273–274, 361
  I/O port 120–121
disk-use light 3, 7
display addresses
  mapping 259–261
  transformation 260
display bits, high-resolution 96
display character sets 89
display formats
  inverse 214
  normal 214
display memory
  addressing 258, 260
  switches 44–46
display modes 104, 261–269, 360
  switching 101–105
display pages 99–101
  HRP1 38
  HRP1X 38
  HRP2 39, 269
  HRP2X 39, 269
  maps 105–111
  TLP1 36
  TLP1X 38
  TLP2 38
  TLP2X 38
display soft switches 50, 101–105
DisVBl switch 190
DisXY switch 189
DMA transfers 357
dollar sign ($) 225
DOS 51, 57, 58, 155, 169, 204, 214, 312, 328, 332
Down Arrow 4
Dvorak keyboard 6–7, 358, 370

E

editing, GetLn 63
80Col switch 101, 102, 243
80-column display 38, 64, 68, 86, 91, 92, 100, 106, 358
  addressing 263
  dot patterns 269
  map 108
  switching to 5
80/40 switch 3, 5
80Store switch 39, 45, 101, 102, 106, 241
EnbXY switch 189
English keyboard 372
enhanced video firmware 249–250
EnlCRAM switch 242
entry points
  firmware 23
  memory expansion card 122
  Monitor 322, 326–327
  ports 71
  UniDisk 3.5 I/O 122
EnVBl switch 190
environmental specifications 232
Escape 5, 60–62, 358
escape codes 60–62, 82
Escape Control-Q 215, 228
EXAMINE command 213, 228
expansion ROM space 74
expansion slots 357
Extended 80-Column Text Card 351, 354, 360
EXTINT 276, 342

F

F command 156, 170
firmware 13
  addresses 322–327
  disk I/O 120
  display mode 104
  enhanced video 249–250
  entry points 23
  I/O 71–73
  mouse 191–196
  protocols 72–73
  serial port 1 160
  serial port 2 174
  Smartport 124
firmware routines
  Monitor 112–115
  mouse 193–194
  video 116–118
flag inputs 23
flags
  inverse 69, 70
  keyboard 78
flashing characters 69, 70, 88–89
forced cold-start reset 50, 52
FORMAT call 127, 135–136
Fortran 330
40-column display 5, 50, 86, 91, 92, 94, 259, 358
  addressing 263
  map 107
  memory 261
  switching to 5
48K memory 36–39
  switches 39
French keyboard 373–374
full-duplex operation 182–184
function keys 4

G

game input 198–201
  characteristics 199
game paddles 198, 287
GCR 247
general logic unit (GLU) 15, 245–246, 365
German keyboard 376–377
GetLn routine 36, 59–60, 78, 82, 205
  escape codes 60–62
GetLn1 routine 82
GetLnZ routine 82
global storage 36
GO command 219
graphics display
  buffer 38
  double high-resolution 97–98, 111, 269
  high-resolution 38–39, 95–97, 110, 267–268
  low-resolution 94–95, 109, 266–267
  mixed-mode 98–99

H

half-duplex operation 180–181
hand controller 198
  circuits 288
  connector 287
  connector signals 199–200, 287
  input and output 287–290, 363–364
  signals 289, 364
handle 232
hardware 365
  addresses 316–321, 353–356
headphone jack 8, 256
hexadecimal 387–389
hexadecimal arithmetic 215
hexadecimal notation 225
high-resolution graphics 38–39, 95–97, 267–268
  double 97–98, 111, 269
  map 110
HiRes switch 45, 103, 241, 243, 354
HLine routine 112, 114
HomeMouse routine 194
HOME routine 112, 114
horizontal-count bits 259–260
HRP1 38
HRP1X 38
HRP2 39, 269
HRP2X 39, 269

I

I command 156, 170, 228
identification bytes 73
index registers 18
INIT call 127, 139–140
InitMouse routine 194
IN#n command 57
IN#2 command 169, 171, 172, 176, 179, 180–183
IN#4 command 196
input buffer 36
input/output unit (IOU) 15, 243–244, 256, 365
inputs
  analog 200
  data 23
  flag 23
  game 198–201
  hand controller 287–290, 363–364
  keyboard 78–82
  mouse 186–198, 282–286, 363
  switch 200
input subroutines 58–63
  GetLn 59–60
  KeyIn 58–59
  Monitor 82
  RdKey 58, 78, 341
Integer BASIC 308, 330, 348, 356, 388
integrated circuits 13, 241–248
Integrated Woz Machine (IWM) 15, 247–248, 273, 365
internal converter 234–235
  specifications 234
Interrupt Request 281
interrupts 74, 331–347, 357
  bypassing 345–347
  external 342–343
  EXTINT 276
  handling 334–345
  keyboard 341–342
  mouse 50, 187–188, 190–191, 339–340, 345–346, 354
  serial 343–345
  Smartport 130, 138
  sources 338–339
  VBlInt 187–188, 190, 243
  vectors 333–334
  XInt 190
  YInt 190
inverse characters 69, 70, 88–89
INVERSE command 90, 214, 228
inverse display 214
inverse flag 69, 70
I/O 357
  cassette 364
  disk 273–274, 361
  disk drive 120–121
  links 56–58
  memory expansion card 123
  port 71–74
  serial 274–282, 361–362
  Smartport 123–124
IOError $27 error 150
IOUDis switch 45, 103, 189, 355
IRQ 334
ISO keyboard 371
Italian keyboard 378–379

J

jacks
  headphone 8, 256
  output 8, 256
joysticks 198, 199

K

K command 156, 170
keyboard 3–7, 254–255, 357–359
  Canadian 375
  characteristics 79
  circuit diagram 254
  Dvorak 6–7, 358, 370
  encoder 78
  English 372
  features 3
  flag 78
  French 373–374
  German 376–377
  input buffer 36, 38
  inputs 78–82
  interrupts 340–342
  ISO 371
  Italian 378–379
  layouts and codes 3, 6–7, 366–381
  Sholes 6, 358, 367–368
  signals 255
  specifications 3
  strobe 78, 339, 353
  switch 3, 6–7
  Western Spanish 380–381
keyboard input switch (KSW) 57, 71, 101
KeyIn routine 56–57, 58–59, 78
keys
  cursor movement 4, 62
  modifier 5, 81
  special function 4
KSW 57, 71, 101
KSWH 57
KSWL 57

L

languages 329–330
last opened location 205
L command 156, 170
Left Arrow 4, 63
LF *See* line feed
line feed 114, 164, 179
LIST command 220–221, 225
Logo II 330
lowercase characters 395
low-resolution graphics 94–95, 266–267
  map 109

M

Machine Language Interface 125
machine-language programs 219–221
main logic board 12–15
main memory 42–44, 161, 175, 269
  screen holes 313–314
main RAM 22
maps
  display address 259–261
  display page 105–111
  48K memory 37
  memory 20, 308–321
  ROM 397, 398
M command 156, 171
memory
  addressing 248–253
  auxiliary 42–44, 74, 106, 160–161, 175, 269, 315
  bank-switched 24–35
  changing contents 208–210
  comparing data 211–212
  dump 206–208
  examining 206
  48K 36–39
  main 42–44, 161, 175, 269, 313–314
  maps 20, 308–321
  moving data 210–211
  RAM 13
  range 207
  ROM 13
memory addresses 20
  display 258, 260
  hardware page 316–321
  port I/O 72–73
  port screen hole 74
  text window 68–69
memory bus organization 249
memory expansion 14
  mouse and 191, 195
memory expansion card 120, 250, 291
  entry points 122
  I/O 123
  startup routine 122
memory management unit (MMU) 15, 241–242, 365
memory pages
  $00 (zero page) 20, 24, 25, 308–311
  $01 (stack) 20, 24, 25
  $02 (input buffer) 36
  $03 (global storage, vectors) 36, 312
  $04–$07 (TLP1) 36–38
  $08 (communication port buffers) 38
  $08–$0B (TLP2) 38
  $20–$3F (HRP1) 38
  $40–$5F (HRP2) 39
  $D0–$FF 26
microprocessor 13, 18–19
  See also 65C02 microprocessor
Mini-Assembler
  address formats 226
  instruction formats 226
  starting 223–224
  using 224–226
mixed-mode displays 98–99
MIXED switch 102, 243, 354
MLI 125
modem 177–178
modem port, commands 170–172
modifier keys 5, 81

Monitor 5, 23, 36, 57, 59, 63, 161, 176, 258, 312, 356
entry points 322, 326–327
firmware routines 112–115
game support 201
input routines 82
interrupts 74
memory location 204
output 270
speaker routines 84
vectors 326–327
Monitor commands
advanced 216–218
debugging 221
machine-language program 219–221
memory 205–212
register 212–213
repeating 217–218
summary 227–229
syntax 205
mouse
as hand controller 198
assembly-language support 195–196
BASIC support 195–196
button signals 286
circuits 285
connector 284
connector signals 284
defaults 50
firmware entry points 23
firmware routines 193–194
inputs 186–198, 282–286, 363
interrupts 50, 339–340, 345–346, 354
I/O firmware support 191–196
operating modes 187–188
Pascal support 195
port characteristics 186
screen holes 196–197
soft switches 189–191
waveforms 283
mouse port
addresses 325
I/O firmware protocol 195
screen holes 197
MouseText 69, 70, 88, 90, 360
characters 91
MouX1 switch 190
MouY1 switch 190
MoveAux routine 42–43
MOVE command 210–211, 216–217, 227
movement/button interrupt mode 188
movement interrupt mode 187–188
multiplexing 251
RAM address 252

N

N command 228
nD command 156, 170
negative decimal 388–389
next changeable location 205, 208–209
nibbles 94, 384, 386
nnB command 156, 170
nnn command 156, 170
nnnN command 157, 171, 362
NoDrive $28 error 151
NonFatal $50–$7F error 151
normal characters 69, 70, 88–89
NORMAL command 90, 214, 228
normal display 214
NoWrite $2B error 150
nP command 157, 171
NTSC 86, 96, 97, 257, 270
#6 command 155
#7 command 169
#8 command 169

O

O Control-K 215
OffLine $2F error 151
Open Apple 4, 52, 81, 200, 221, 222, 358
Open Apple-Control-Reset 52, 121, 162, 176, 361
OPEN call 127, 140–141
operating systems 214, 328–329
output jack 256
outputs
hand controller 363–364
speaker 82–84
strobe 24
video display 116–118
video signal 270–273
output subroutines 64–70
COut 64
COut1 65
C3Out1 65–67
overflow bit 44

P

paddles 198, 287
Page 1 20, 36–38, 50, 269
high-resolution 95
text 100, 106
Page 1X 38, 269
text 100
Page 2 20, 39
high-resolution 95
text 100
Page2 switch 45, 102, 241, 243, 354
Page 2X 38, 39
pages, display *See* display pages
pages, memory *See* memory pages
PAL 257
parity bit 163
Pascal 56, 68, 72, 124, 169, 330, 332
mouse and 195
video control functions 117–118
Pascal Operating System 329
Pdl0 287
Pdl1 287
PEEK 329
period (.) 206
peripheral-card memory spaces 352–353
peripheral identification numbers (PIN) 389–390
PInit routine 116

pinouts
GLU 246
IOU 243
IWM 247
MMU 242
RAM 251
ROM 249, 250
6551 277
TMG 245
video expansion connector 272
PLOT routine 112, 114
POKE 329
ports
characteristics 71
disk I/O 120–121
entry points 71
I/O 71–74
mouse 186, 325
printer 155–157
ROM space 73
screen hole RAM space 74
serial 275
serial port 1 22, 154–165
serial port 2 22, 38, 167–184
video output 86
PosMouse routine 193
power converter 234–235
specifications 234
power light 3, 7
power-on reset 50
power supply 11–12
specifications 233, 383
PR#n command 57
PR#0 command 198
PR#1 command 155, 159, 162, 181
PR#2 command 169, 171, 172, 179, 181, 183, 184
PR#3 command 90, 196
PR#4 command 196, 198
PR#7 command 328
PRBl2 routine 112, 114
PrByte routine 112, 115
PRead routine 116, 201
P register 18, 213
PrErr routine 112, 115
PrHex routine 112, 115
primary character set 69, 70, 88, 359
PRINTER command 155
printer port, commands 155–157
PrntAX routine 113, 115
processor status register 18, 213
ProDOS 51, 57, 124, 155, 169, 204, 214, 312, 328, 332
Machine Language Interface 125
program counter (PC) 18
prompt characters 59
Protocol Converter 124
PStatus routine 118
PTrig switch 190
PWrite routine 117

Q

Q command 171
question mark (?) 59
blinking 169, 179

R

RAM 13
addressing 251–253
auxiliary 22, 184
main 22
memory expansion card 123
peripheral-card 353
port screen hole 74
refreshing 251–252
Smartport 126
timing 252–253
RAM addresses 22, 351
multiplexing 252
RAMRd switch 39, 44, 242
RAMWrt switch 39, 44, 242
R command 157, 171
Rd63 switch 190
Rd80Col switch 102
Rd80Store switch 45, 102
RdAltChar switch 102
RdAltZP switch 28
RdBnk2 switch 28
RdBtn0 switch 190
RdChar routine 82
RdDHiRes switch 46, 103
RdHiRes switch 45, 103
RdIOUDis switch 46, 103, 189
RdKey routine 58, 78, 341
RdLCRAM switch 28
RdMIXED switch 102
RdPage2 switch 45, 102
RdRAMRd switch 39
RdRAMWrt switch 39
RdTEXT switch 102
RdVBlMsk switch 190
RdX0Edge switch 189
RdXYMsk switch 189
RdY0Edge switch 190
READ BLOCK call 126, 132–133
READ call 127
ReadMouse routine 188, 192, 193
registers 18–19
changing 213
command 280
control 278–279
examining 213
shift 263, 268
Smartport 125
status 18, 130, 213, 281
transmit/receive 282
REMIN command 169
REMOUT command 169
Request to Send (RTS) 280
Reset 3, 4, 81, 358
reset routine 49–50
reset vectors 50–53
Return 4, 60, 169, 204, 206, 208, 209, 213, 215, 224, 227
return from subroutine (RTS) 57, 219
retype 63
RGB 269
Right Arrow 4, 63, 82
ROM 13
addresses 22–23, 351–352
addressing 249–250
Autostart 356
character generator 14, 266, 267
Monitor 356
peripheral-card 352

ROMEN1 250
ROMEN2 250
ROM map
  auxiliary side 398
  main side 397
ROM space
  expansion 74
  port 73
row-address strobe (RAS) 252
RstVBl switch 190
RstXInt switch 189, 242
RstXY switch 189
RstYInt switch 190, 242
RTS
  Request to Send 280
  return from subroutine 57, 219

S

S command 157, 171
screen holes 38, 312–315
  auxiliary memory 315
  main memory 313–314
  mouse 196–197
  serial port 1 160–161
  serial port 2 174–175
SCRN routine 113, 115
serial buffering 343–344
serial cards 361–362
serial I/O 274–282, 361–362
  buffers 362
serial port 361–362
  circuits 275
  connectors 278
  connector signals 278
serial port 1 22, 154–165
  addresses 323
  at startup 159
  carriage return and line feed 164
  characteristics 154, 161–165
  data format and baud rate 163
  displaying output 165
  hardware page locations 159
  I/O firmware support 160
  screen holes 160–161
  sending special characters 165
  using 155–158
serial port 2 22, 38, 167–184
  addresses 324
  at startup 173
  carriage return and line feed 179
  characteristics 168–169, 176–184
  commands 170–172
  data format and baud rate 177–178
  hardware page locations 173–174
  I/O firmware support 174
  routing input and output 179–184
  screen holes 174–175
  using 169–173
ServeMouse routine 187, 188, 193, 339
SetCol routine 113, 115
SetMouse routine 192, 193, 339
SetPWRC routine 53
Shift 5, 81, 255, 358
shift register 263, 268
Sholes keyboard 6, 358, 367–368
signals
  BREAK 163
  character generator 266
  clock 239–241
  disk drive connector 274
  GLU 246
  hand controller 199–200, 289, 364
  hand controller connector 287
  IOU 243–244
  IWM 247
  keyboard 255
  MMU 242
  mouse button 286
  mouse connector 187, 284
  RAM timing 253
  ROM 250
  serial port connector 278
  65C02 305
  6551 277
  synchronization 257
  timing 264–265
  TMG 245
  video expansion connector 272–273
  video output 270–273
65C02 microprocessor 13, 75, 237–241, 297–307
  addressing modes 26, 302, 304
  block diagram 237–238, 299
  characteristics 300–301
  data sheets 298–307
  enhancements 301
  instruction mnemonics 302
  instruction set 306
  interrupt handling 333
  operational codes 306–307
  pin configuration 299
  pin function 300
  programming model 303
  ratings 300
  registers 18–19
  signal description 305
  specifications 239
  timing 239–241, 302
  versus 6502 297–298
SLOTC3ROM switch 354
SLOTCXROM switch 354
Smartport
  calls 125–148
  commands and parameters 149
  error codes 150–151
  I/O interface 123–124
  locating 124
  soft switches 20, 24
  bank selector 27
  display 50, 101–105
  mouse 189–191
Solid Apple 4, 52, 81, 200, 221, 222, 358
Solid Apple-Control-X 341
speaker 359
  circuit diagram 256
  outputs 82–84
  volume control 8, 256
special characters 393
special function keys 4
specifications
  environmental 232
  power supply 233, 383
  65C02 239
S register 18, 213
stack pointer 18, 213
startup 121–123
STATUS call 126, 128–132

status register, ACIA 130, 281
STEP command 221–223
stop-list function 67
STORE command 216, 227
strobes
  column-address 252
  keyboard 78, 255, 339, 353
  outputs 24
  row-address 252
Super Serial Card 179, 361, 362
Sw0 287
Sw1 287
switch inputs 200
synchronization signals 257

T

Tab 4, 358
T command 171, 180–183
terminal mode 179, 184
text, character sets 88–89
text display 263
  characteristics 93
  40-column 5, 50, 86, 91, 92, 94, 107, 259, 261, 358
  80-column 5, 38, 64, 68, 86, 91, 92, 100, 106, 108, 269, 358
  switching 93
text Page 1 100, 106
text Page 1X 100
text Page 2 100
TEXT switch 102, 243, 354
text window 68–69
timing generator (TMG) 15, 245, 365
TLP1 36
TLP1X 38
TLP2 38
TLP2X 38
toggle switches 24
TRACE command 221–223
transmit/receive register, ACIA 282
transparent mode 187

U

underscore (_), blinking 179
UniDisk 3.5 120, 124, 130, 223
  control list 138
  I/O entry points 122
Up Arrow 4
uppercase characters 394
USER command 218, 228

V

validity-check byte 51, 53
VBlInt switch 243
vectors 36, 56
  interrupt 333–334
  Monitor 326–327
  reset 50–53
VERIFY command 211–212, 217, 227
vertical blanking 338, 354, 360
  active modes 188
vertical-count bits 260
VID 270, 365
video counters 257–258
video display 257–273, 359–360
  addresses 101
  circuits 262
  I/O firmware support 116–118
  modes 261–269
  specifications 87
video expansion
  connector signals 272–273
  output 271–273
video output firmware
  addresses 324
  entry points 23
video output port, characteristics 86
video output signals 257, 270–273
VLine routine 113, 115
voltage converter 11–12
volume control 8, 256
VTabZ routine 113

W

warm start 50, 51, 123
Western Spanish keyboard 380–381
WNDW 257
words 385
WRITE BLOCK call 127, 134–135
WRITE call 127, 143–144

X

X0 243, 284
X0Edge switch 189
X1 284
X command 157, 171
XFer routine 42–44, 312
XInt 190, 354
XON/XOFF protocol 171
X register 18, 43, 73, 84, 114, 115, 118, 128, 192, 201, 213

Y

Y0 243, 284
Y0Edge switch 190
Y1 284
YInt 190, 354
Y register 18, 43, 113, 114, 115, 128, 192, 201, 213

Z

Z command 157, 165, 171

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using the Apple Macintosh™ Plus and Microsoft® Word. Proof and final pages were created on the Apple LaserWriter® Plus. POSTSCRIPT™, the LaserWriter's page-description language, was developed by Adobe Systems Incorporated.

Text type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Display type is ITC Avant Garde Gothic®. Bullets are ITC Zapf Dingbats®. Program listings are set in Apple Courier, a monospaced font.

The Apple Technical Library

The Official Publications from Apple® Computer, Inc.

The Apple Technical Library offers programmers, developers, and enthusiasts the most complete technical information available on Apple computers, peripherals, and software. The Library consists of technical manuals for the Apple II family of computers, the Macintosh family of computers, key peripherals, and programming environments.

Apple Technical Library titles on the Apple II family include technical references to the Apple IIe, Apple IIc, and Apple IIGS computers, with detailed descriptions of the hardware, firmware, ProDOS operating system, and the built-in programming tools that programmers and developers can draw upon. In addition to a technical introduction and programmer's guide to the Apple IIGS, there are tutorials and references for Applesoft BASIC and Instant Pascal programmers.

The Inside Macintosh Library provides complete technical references to the Macintosh 512, Macintosh 512 enhanced, Macintosh Plus, Macintosh SE, and Macintosh II computers. Individual volumes provide technical introductions and programmer's guides to the Macintosh as well as detailed information on hardware, firmware, system software, and programming tools. The Inside Macintosh Library offers the most detailed and complete source of information available for the Macintosh family of computers.

In addition, titles in the Apple Technical Library offer references to the wide range of important printers, communications standards, and programming environments such as the Standard Apple Numerics Environment (SANE) to help programmers and experienced users get the most out their computer systems.



Apple IIc Technical Reference Manual

The Official Publication from Apple Computer, Inc.

Apple's definitive guide to all versions of the Apple® IIc personal computer. Written and produced by the people at Apple Computer, this manual provides a comprehensive, single-source reference for programmers and hardware designers.

The *Apple IIc Technical Reference Manual* describes all aspects of the IIc—including its physical characteristics, the hardware/firmware locations that control memory and I/O, and the electrical and electronic implementation of machine features and capabilities. Summary tables provide quick reference to all of the I/O firmware entry points, including those used by the system's Smartport.

The manual describes:

- Memory organization and control.
- Apple IIc I/O interface, including keyboard and speaker, video display modes (including graphics modes), internal and external disk drives, the Apple IIc Memory Expansion Card, serial ports, and the mouse and game port.
- Use of the system Monitor routines in the IIc firmware to disassemble and debug machine-language programs.
- The hardware, including pinouts of custom ICs and internal/external connectors, plus schematic diagrams.

Appendixes contain quick-reference tables, describe interrupt handling, and include product comparisons of all members of the Apple II family of computers.

This edition of the *Apple IIc Technical Reference Manual* also includes firmware listings for the new Apple IIc Memory Expansion Card. Information on obtaining firmware listings for earlier versions of the IIc, as well as supplementary technical information, is provided.

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576

030-1238-B
Printed in U.S.A.

Addison-Wesley Publishing Company, Inc.

ISBN 0-201-17752-8