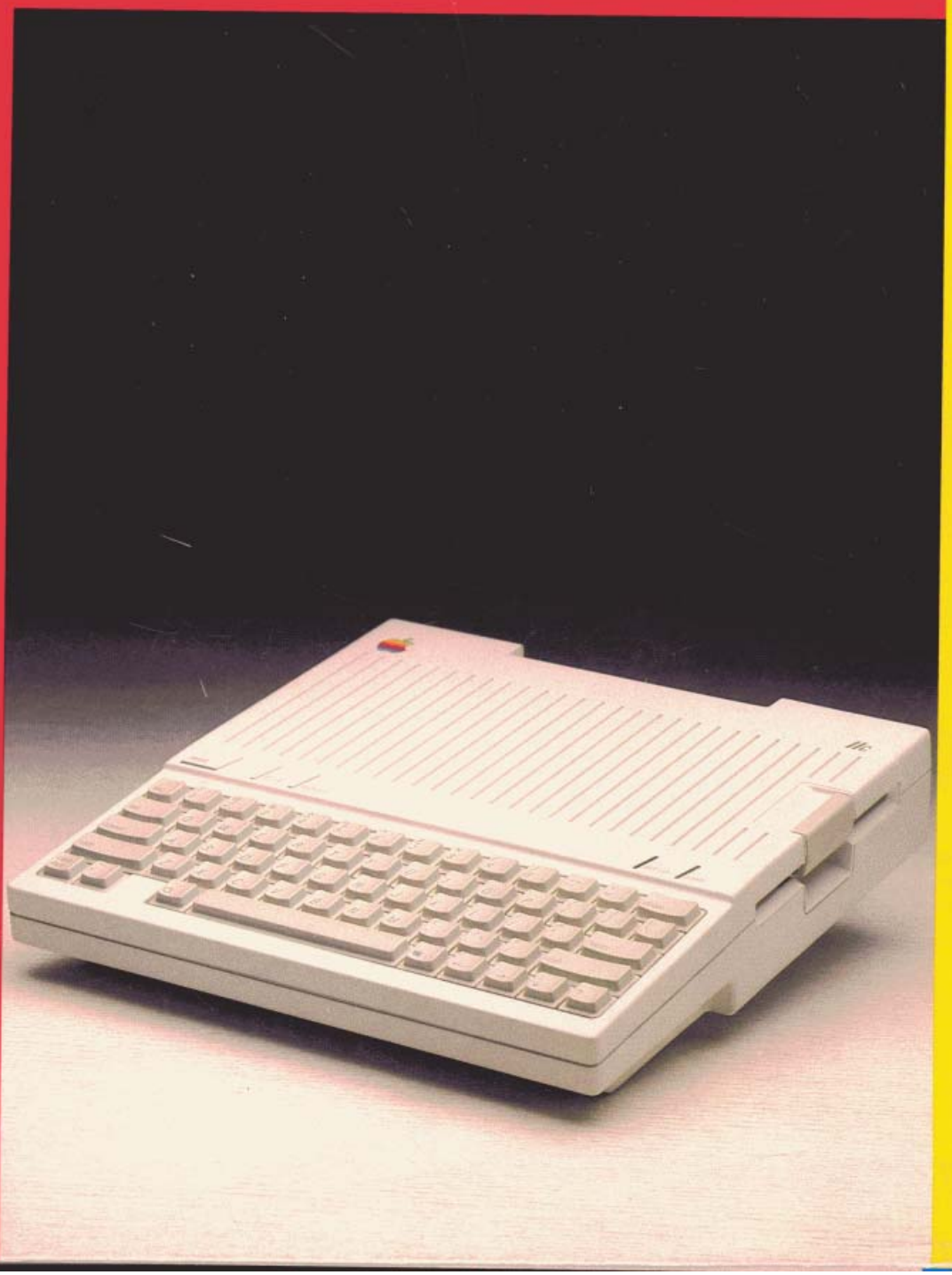
The Apple IIc

The Apple IIc Reference Manual

Volume 1



Customer Satisfaction

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the documentation or media at no charge to you during the 90-day period after you purchased the product.

In addition, if Apple releases a corrective update to a software product during the 90-day period after you purchased the software, Apple will replace the applicable disks and documentation with the revised version at no charge to you during the six months after the date of purchase.

In some countries the replacement period may be different; check with your authorized Apple dealer. Return any item to be replaced with proof of purchase to Apple or an authorized Apple dealer.

Limitation on Warranties and Liability

Even though Apple has tested the software described in the manual and reviewed its contents, neither Apple nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in this manual, their quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is," and you the purchaser are assuming the entire risk as to their quality and performance. In no event will Apple or its software suppliers be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering or reproducing these programs or data. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Copyright

Product Revisions

Apple cannot guarantee that you will receive notice of a revision to the software described in the manual, even if you have returned a registration card received with the product. You should periodically check with your authorized Apple dealer.

20525 Mariani Avenue
Cupertino, California 95014

Warning

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

Reorder Apple Product #A2L4030

The Apple IIc

Apple IIc Reference Manual
Volume 1

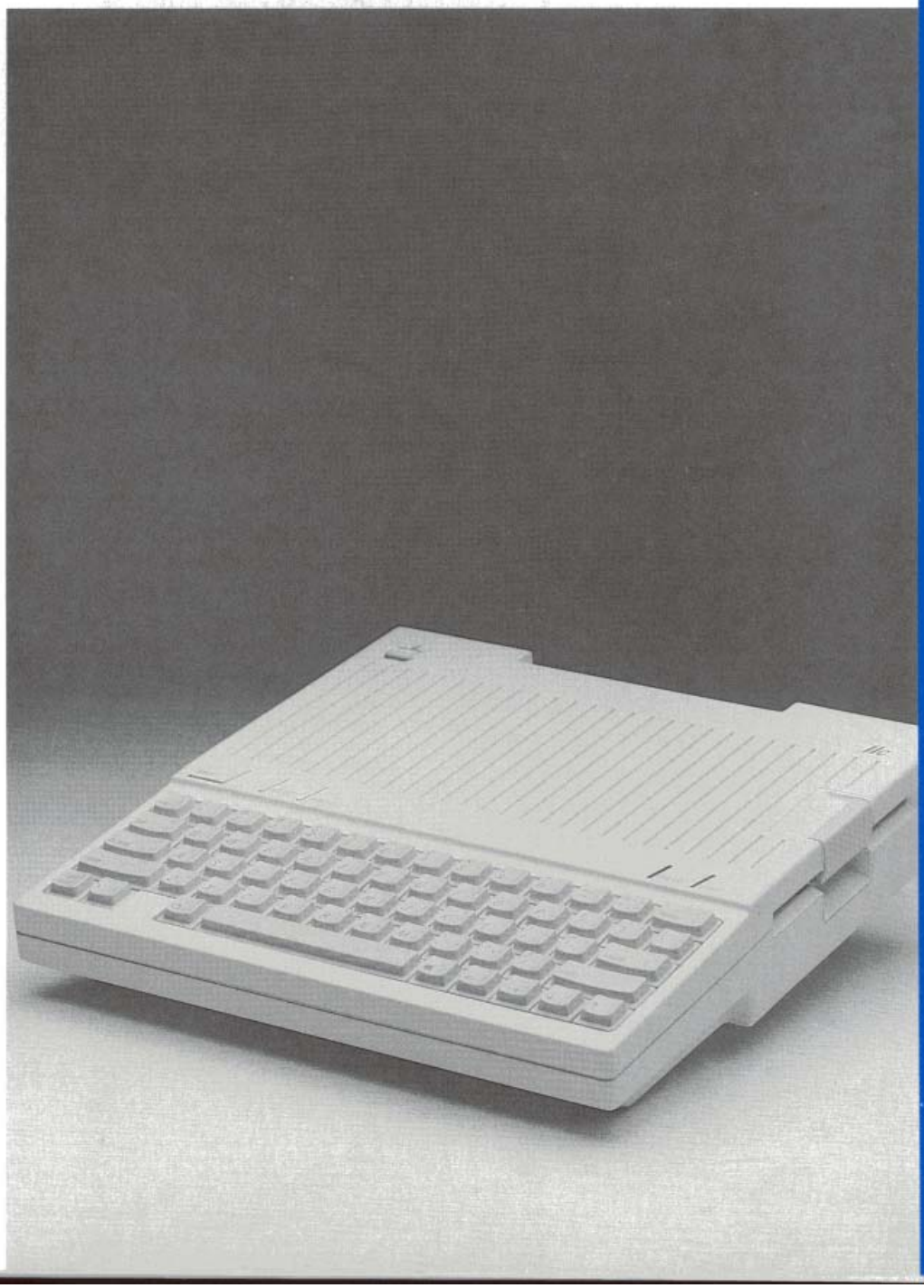


Table of Contents

Volume 1

Chapter 2 Memory Organization and Control 15

Volume 2

Appendix E Interrupts 41

Appendix F Apple II Series Differences 61

List of Figures and Tables

Volume 1

Chapter 1 Introduction

Chapter 2 Memory Organization and Control

Chapter 3 Introduction to Apple IIc I/O

Chapter 4 Keyboard and Speaker

Chapter 5 Video Display Output

Chapter 9 Mouse and Game Input

Chapter 11 Hardware Implementation

Volume 2

Appendix A The 65C02 Microprocessor

Appendix B Memory Map

Appendix C Important Firmware Locations

Appendix E Interrupts

Appendix F Apple II Series Differences

Appendix G USA and International Models

Appendix H Conversion Tables

Radio Frequency Interference Statement

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

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

You can determine whether your computer is causing interference by turning it off. If the interference stops, it was probably caused by the computer or its peripheral devices. To further isolate the problem:

- Disconnect the peripheral devices and their input/output cables one at a time. If the interference stops, it is caused by either the peripheral device or its I/O cable. These devices usually require shielded I/O cables. For Apple peripheral devices, you can obtain the proper shielded cable from your dealer. For non-Apple peripheral devices, contact the manufacturer or dealer for assistance.

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

- Turn the TV or radio antenna until the interference stops.
- Move the computer to one side or the other of the TV or radio.
- Move the computer farther away from the TV or radio.
- Plug the computer into an outlet that is on a different circuit than the TV or radio. (That is, make certain the computer and the radio or television set are on circuits controlled by different circuit breakers or fuses.)
- Consider installing a rooftop television antenna with coaxial cable lead-in between the antenna and TV.

If necessary, you should consult your dealer or an experienced radio/television technician for additional suggestions. You may find helpful the following booklet, prepared by the Federal Communications Commission:

"How to Identify and Resolve Radio-TV Interference Problems."

This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, stock number 004-000-00345-4.

Preface

This is the reference manual for the Apple IIc personal computer. It contains detailed descriptions of all of the hardware and firmware that make up the Apple IIc. The manual is divided into two volumes: Volume I contains all the chapters; Volume II contains the appendixes.

This manual contains a lot of information about the way the Apple IIc works, but it doesn't tell you how to use the Apple IIc. For this, you should read the other Apple IIc manuals, especially the *Apple IIc Interactive Owner's Manual*.

This manual describes the internal operation of the Apple IIc as completely as possible in a single reference work. The criterion for deciding to include an item of information was whether it would help an assembly-language programmer or hardware designer.

Contents of This Manual

This manual presents first the physical characteristics of the Apple IIc (Chapter 1), then the hardware locations and firmware that control memory management and input/output (Chapters 2 through 10), and finally the electrical and electronic inplementation of those capabilities (Chapter 11). The appendixes contain summary tables and information comparing other Apple products to the Apple IIc.

Chapter 1 identifies the main physical features of the Apple IIc.

Chapter 2 presents an overview of the 65C02 microprocessor (whose instruction set appears in Appendix A), and then discusses the processor's address space, what it contains, and how to control it.

Chapter 3 is an introduction to Chapters 4 through 9. It describes the common characteristics of input/output processing. Chapters 4 through 9 then discuss the kinds of devices—both built-in and attachable—that the Apple IIc supports:

- Keyboard and speaker (Chapter 4)
- Video display (Chapter 5)
- Disk drives (Chapter 6)
- Serial port 1 for printers and plotters (Chapter 7)
- Serial port 2 for modems and other communication devices (Chapter 8)
- Mouse and hand controls, including game paddles and joysticks (Chapter 9)

Chapter 10 is a brief tutorial on how to use the Monitor firmware to disassemble and debug machine-language programs, and manipulate memory contents.

Chapter 11 is a detailed description of the hardware that implements the features described in the earlier chapters. This information is included primarily for programmers, but it will also help you if you just want to understand more about the way the Apple IIc works.

Additional reference information appears in the appendixes. Appendix A is the manufacturer's description of the 65C02 instruction set, including the 27 new instructions available on the CMOS version of the 65C02 used in the Apple IIc.

Appendix B is a memory map of the Apple IIc, including detailed tables of page zero, page three, the screen holes, and the hardware page.

Appendix C lists the *published* firmware entry points, arranged by address, and indicates where in the manual they are described. The list includes I/O firmware (pages $C1 through $CF) and Monitor firmware (pages $F0 through $FF). For Applesoft interpreter firmware (pages $D0 through $EF), refer to the *Applesoft BASIC Reference Manual* (in two volumes).

Appendix D discusses the operating systems and languages that run on the Apple IIc.

Appendix E describes how to use the Apple IIc's interrupt handling capabilities.

Appendix F contains an overview of the differences among the Apple II series computers.

Appendix G contains the keyboard layouts, code conversion tables, and external power supply characteristics of USA and international models of the Apple IIc.

Appendix H contains reference tables for code and number base conversion.

Appendix I contains a listing code for the Monitor, enhanced video firmware, and input/output firmware contained in the Apple IIc. The listings do not include the built-in Applesoft interpreter, which is discussed in the *Applesoft BASIC Programmer's Reference Manual.*

The bibliography lists articles and books containing additional information about the Apple IIc and related products.

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

At the back of this manual is a *Tell Apple Card.* Please fill it out and send it in. Your experience with this manual will help us plan new reference materials.

Symbols in This Manual

This manual uses a three-level numbering system to make it easier to cross-reference information. A reference like 2.4.3 means Chapter 2, section 4, subsection 3. G.1.8 refers to Appendix G, section 1, subsection 8.

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

Warning
Important information regarding your safety or protection of data appears in boxes like this.

Note: Information that is useful but not central to the discussion in a given part of the text appears in gray boxes like this.

Captions, cross-references and incidental definitions appear in marginal glosses like this.

Introduction

chapter 1

This chapter introduces you to the Apple IIc. It identifies the major components of the machine, both outside and inside, and tells you where in the rest of the manual to find information about each one.

1.1 Outside of Machine

Appendix G illustrates and discusses several international keyboard layouts.

Figure 1-1 shows an Apple IIc with a Standard USA keyboard. This chapter discusses both the Standard (Sholes) and Simplified (Dvorak) USA keyboards, as well as the lights and switches on the front of the machine.

Figure 1-2 is a diagram of the parts of the computer that you can see, hear, or access from the outside. The Apple IIc comes equipped with keyboard, speaker (with headphone jack and volume control), disk drive, attachable power supply, and internal voltage converter. It also has built-in interfaces and connectors for a serial printer, video display, special video display adapters, modem, mouse, and game controls.

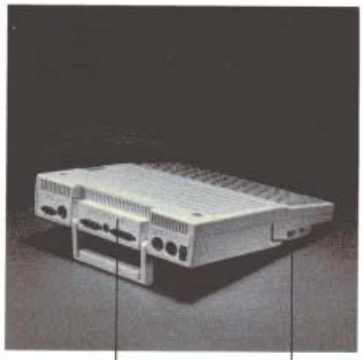
There are no user connections inside the Apple IIc, but expansion is possible and easy with the connectors on the Apple IIc back panel.

Figure 1-1. *Apple IIc With Standard USA Keyboard*

Keyboard (See Figs. 1-4 and 1-5)

Disk Drive (See Fig. 1-7)

Figure 1-2. *Block Diagram of External Features*



Back Panel (See Fig. 1-8)

Speaker (See Fig. 1-6)

1.1.1 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 front of the Apple IIc includes the keyboard (Figure 1-3), the computer's primary input device. It has a typewriter layout, uppercase and lowercase, and can generate all 128 characters—including control characters—in the ASCII character set. The front of the computer also has a reset key, 80/40 switch, keyboard switch, disk-use light, and power light.

Figure 1-3. Keyboard and Front of Apple IIc

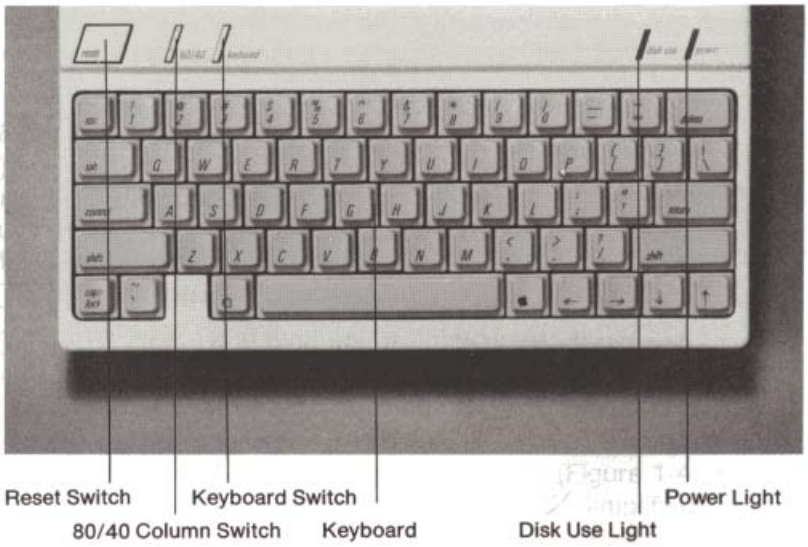


Table 1-1 lists the characteristics of all Apple IIc keyboards and front panels.

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, Ó, Apple
Cursor movement keys:	←, →, ↓, ↑, RETURN, DELETE, TAB
Modifier keys:	CONTROL, SHIFT, CAPS-LOCK, ESC
Front-panel switches:	80/40 switch, keyboard switch
Front-panel lights:	Power light, disk-use light

Features

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

Special Function Keys

The Apple keys are connected to one-bit addresses in memory, described in Chapter 9.

The Apple IIc has three special function keys: RESET, and two keys marked with apples: one outlined, ⌘, and one filled in, .

RESET has a direct line to the 65C02 microprocessor: holding down CONTROL while pressing RESET causes the Apple IIc to restart processing with a program that puts the machine in a known state (Chapter 2). So you don't accidentally destroy current work, the reset takes effect only if you hold down CONTROL while pressing RESET.

Chapter 2 describes the results of the various reset procedures.

If you hold down both CONTROL and ⌘ while pressing RESET, the computer starts up as if you just turned it on.

Cursor Movement Keys

Four of the cursor movement keys have arrows on them: left, right, down and up. The other three keys are RETURN, DELETE and TAB. All generate ASCII control characters (Table 4-2). However, 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, change the codes generated by the other keys. None of these keys generates a code when pressed by itself. A fourth key, ESC, generates a control code that the Monitor responds to by interpreting 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.

The other keys to use with CONTROL are: @ [\ ] ^ _ and their international equivalents (see Appendix G).

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.

ESC 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 ESC and it generates the ESC control character (key code $1B—see Table 4-2). Many programs, including the built-in Monitor program, then interpret other specific keys as designating an **escape sequence**.

The 80/40 Switch

This switch takes effect only if the program or operating system you are using actually checks it. See Table 4-1.

The 80/40 switch selects whether information is to be displayed in 40 columns to the line or 80 columns. This switch indicates 40-column display in its down position, and 80-column display in its up position.

Note: Not all programs check this switch. Even programs that do check the switch may rely on the user setting it before running the program.

The Keyboard Switch

The keyboard switch selects which of the two keyboard layouts and character sets the computer is to get from the keyboard and display on the screen. On USA versions of the Apple IIc, 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.

Figure 1-4. *The USA Standard or* ***Sholes*** *Keyboard (Keyboard Switch Up)*

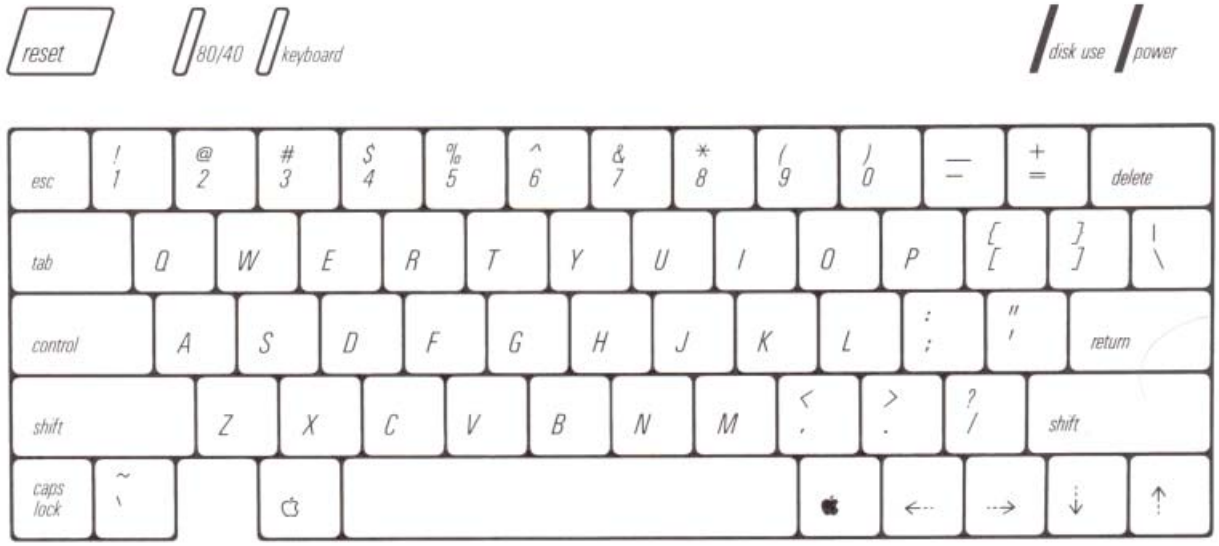
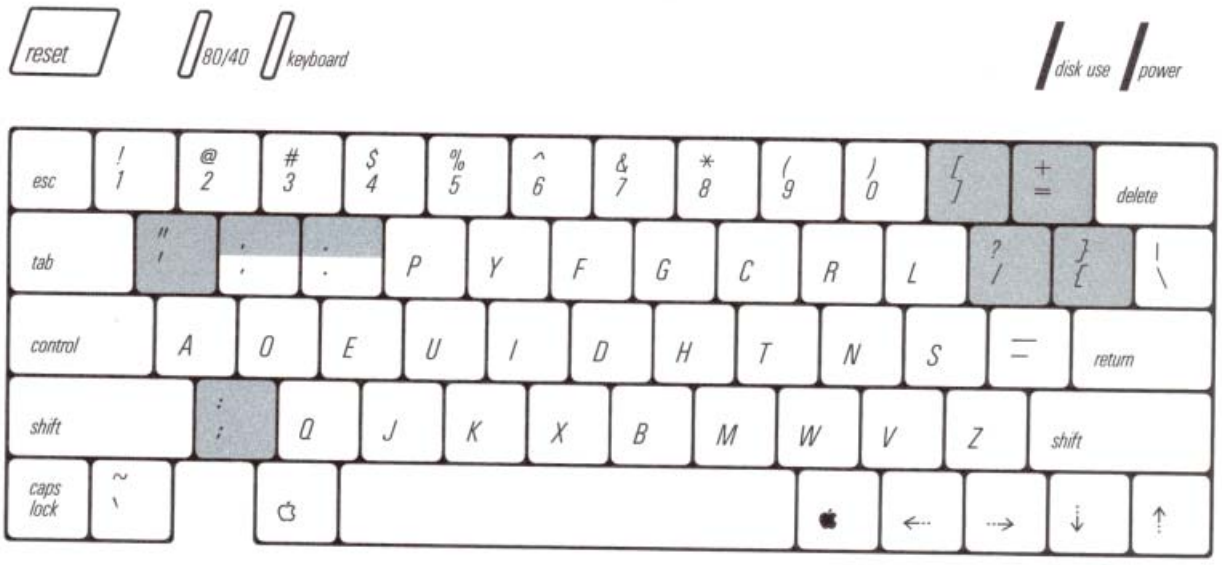


Figure 1-5. *Simplified or* ***Dvorak*** *Keyboard (Keyboard Switch Down). Note: Shaded characters may be in different positions on some models.*



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 on.

The green power light glows when 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) and remedy it 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.

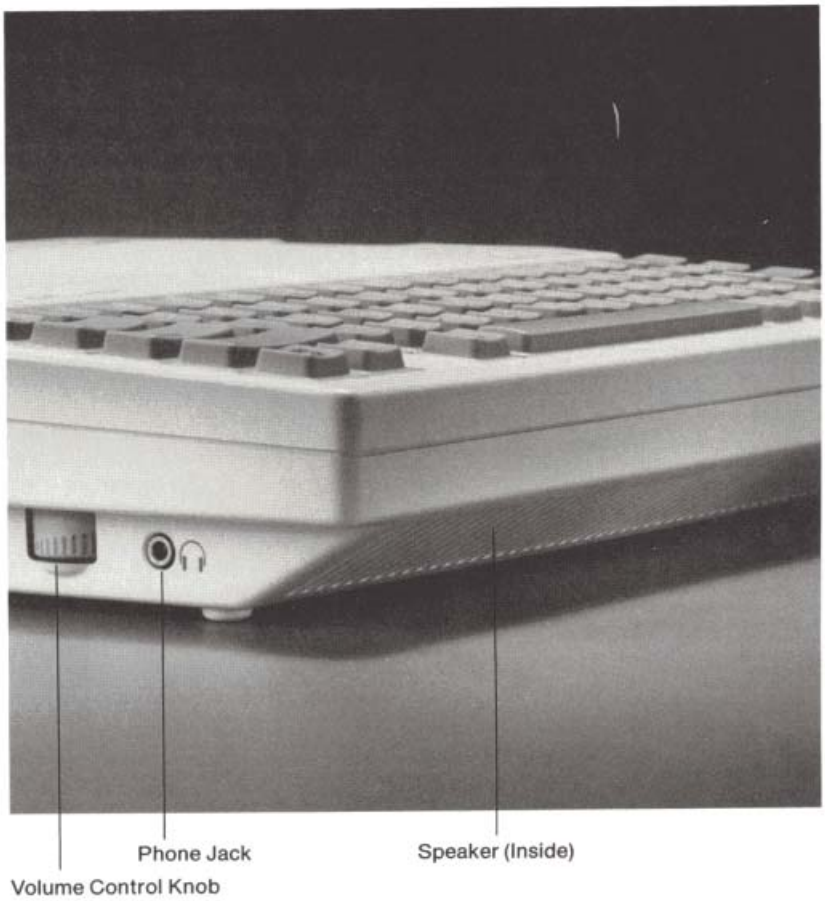
1.1.2 The Speaker

The Apple IIc has a loudspeaker in the bottom of the case, as shown in Figure 1-6. The speaker enables Apple IIc programs to produce a variety of sounds that make programs more useful and interesting. There is also a volume control on the left side of the Apple IIc case, and a mini-phone jack for connecting headphones or an external speaker.

The way programs control the speaker is described in section 4.2.

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 Phone Jack*



1.1.3 The Built-in Disk Drive

I/O means input (information coming into the computer) and output (information going out of the computer). Chapter 6 describes how to use the Apple IIc's disk I/O hardware and firmware.

The Apple IIc has a built-in disk drive (Figure 1-7) that is fully compatible with Apple Disk II—that is, it reads and writes 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*

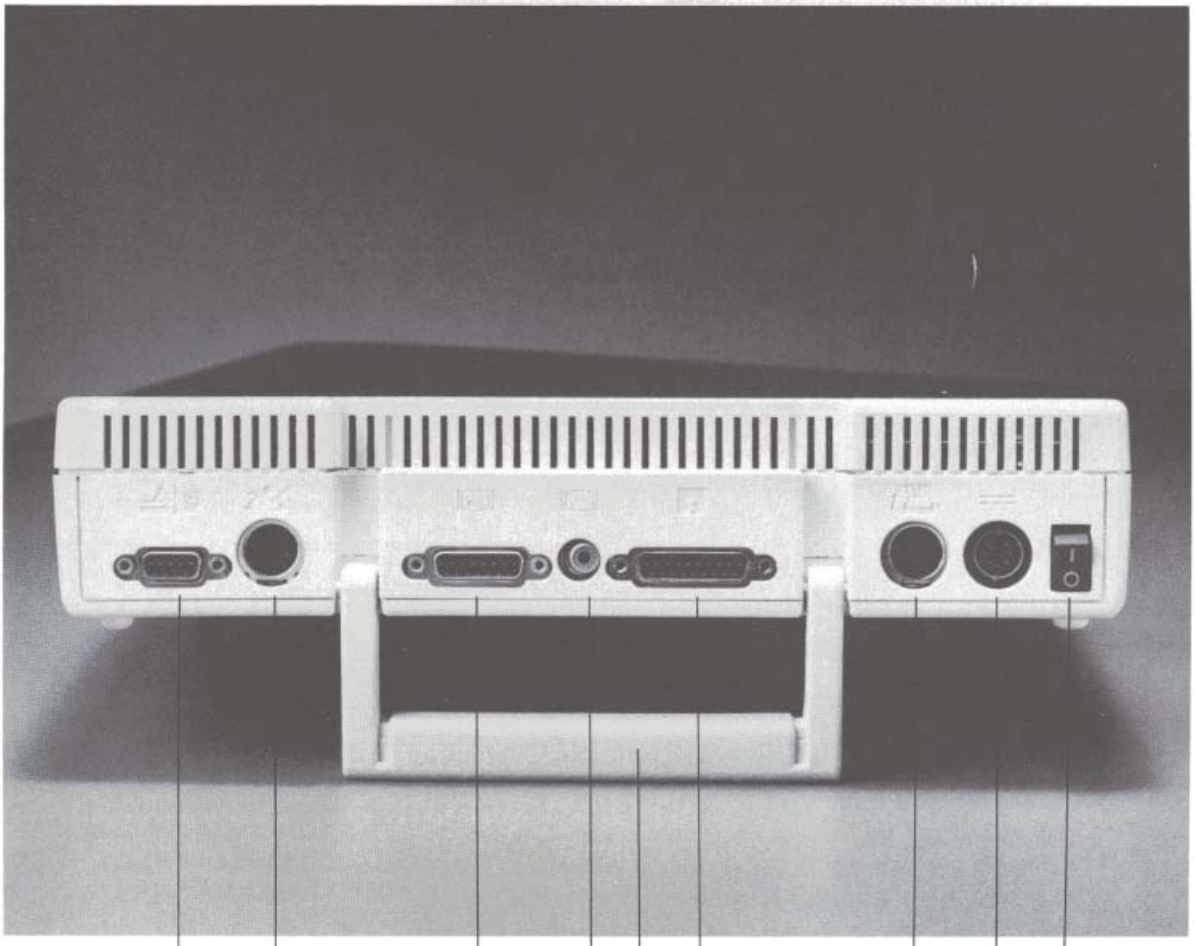
1.1.4 The Back Panel

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 controls, a mouse, a joystick or some other pointing device
- a 5-pin DIN connector for serial input and output (port 2; normally for a modem)
- a 15-pin D-type connector for video expansion
- an RCA-type jack for a video monitor
- a 19-pin D-type connector for connecting a second disk drive
- another 5-pin DIN connector for serial input and output (port 1; normally for a printer or plotter)
- a special 7-pin DIN connector for power input

The installation manuals for the external devices contain instructions for connecting them. Be sure to move the handle until it clicks into position for propping up the computer before attaching cables to the back panel.

Figure 1-8. *Back Panel Connectors*

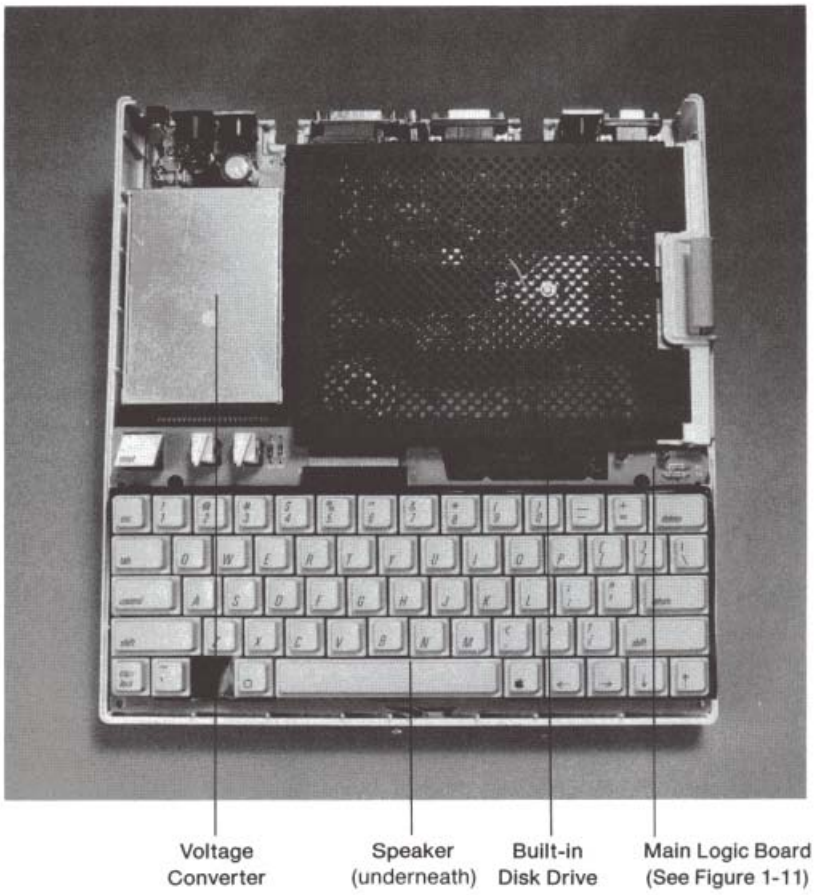


1.2 Inside of Machine

Chapter 11 discusses in further detail these components and how they work.

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

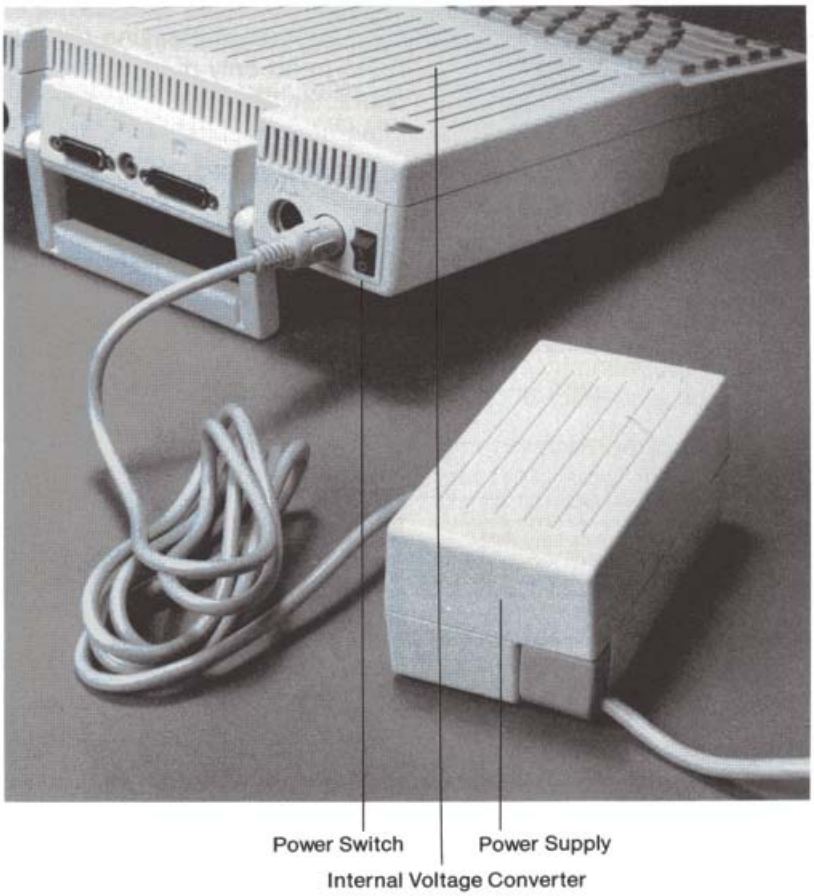
Figure 1-9. *Block Diagram of Inside of Machine*



1.2.1 The Internal Voltage Converter

The built-in voltage converter operates from a 15V DC 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



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

The voltage converter produces three different voltages: +5V, +12V, and -12V. (Minus 5V 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 mishaps.

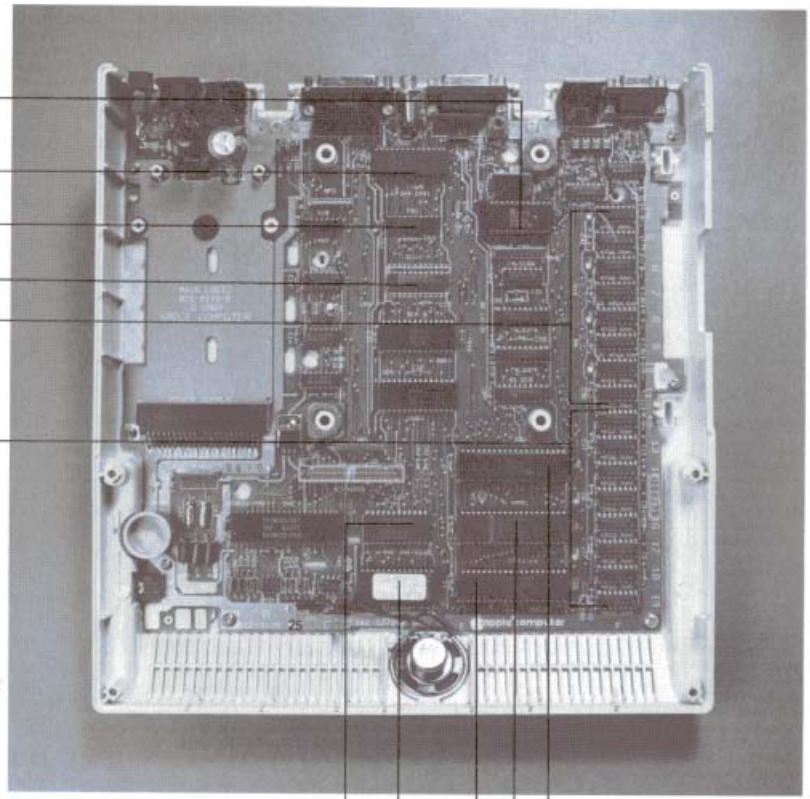
1.2.2 The Main Logic Board

Almost all the electronic parts of the Apple IIc are attached to the main logic board, which is mounted flat in the bottom of the case.

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

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 integrated circuits.

Figure 1-11. *Main Logic Board*



The specifications of the 65C02 are given in Chapter 11; the 65C02 instruction set is given in Appendix A.

The CPU is a 65C02 microprocessor. The 65C02 is a CMOS version of the 6502, which is an eight-bit microprocessor with a sixteen-bit address bus. In the Apple IIc, the 65C02 runs at 1 MHz and performs up to 500,000 eight-bit operations per second.

Chapter 11 describes how RAM works; Appendix B lists important RAM locations.

The keyboard is scanned by an IC that generates matrix values for a read-only memory (ROM). The value of this ASCII code is latched and readable by programs.

ROMs: see Chapter 11.

The character generator ROM converts display information to a form that display devices 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 IC's are custom-made for the Apple IIc:

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 logic functions required.

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

- The Integrated *Woz* Machine (IWM) is a single-chip version of the Apple Disk II controller card.

1.2.3 The Other Circuit Boards

The Apple IIc contains other circuit boards that serve special purposes: a motor-speed control board and a 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.

Memory Organization and Control

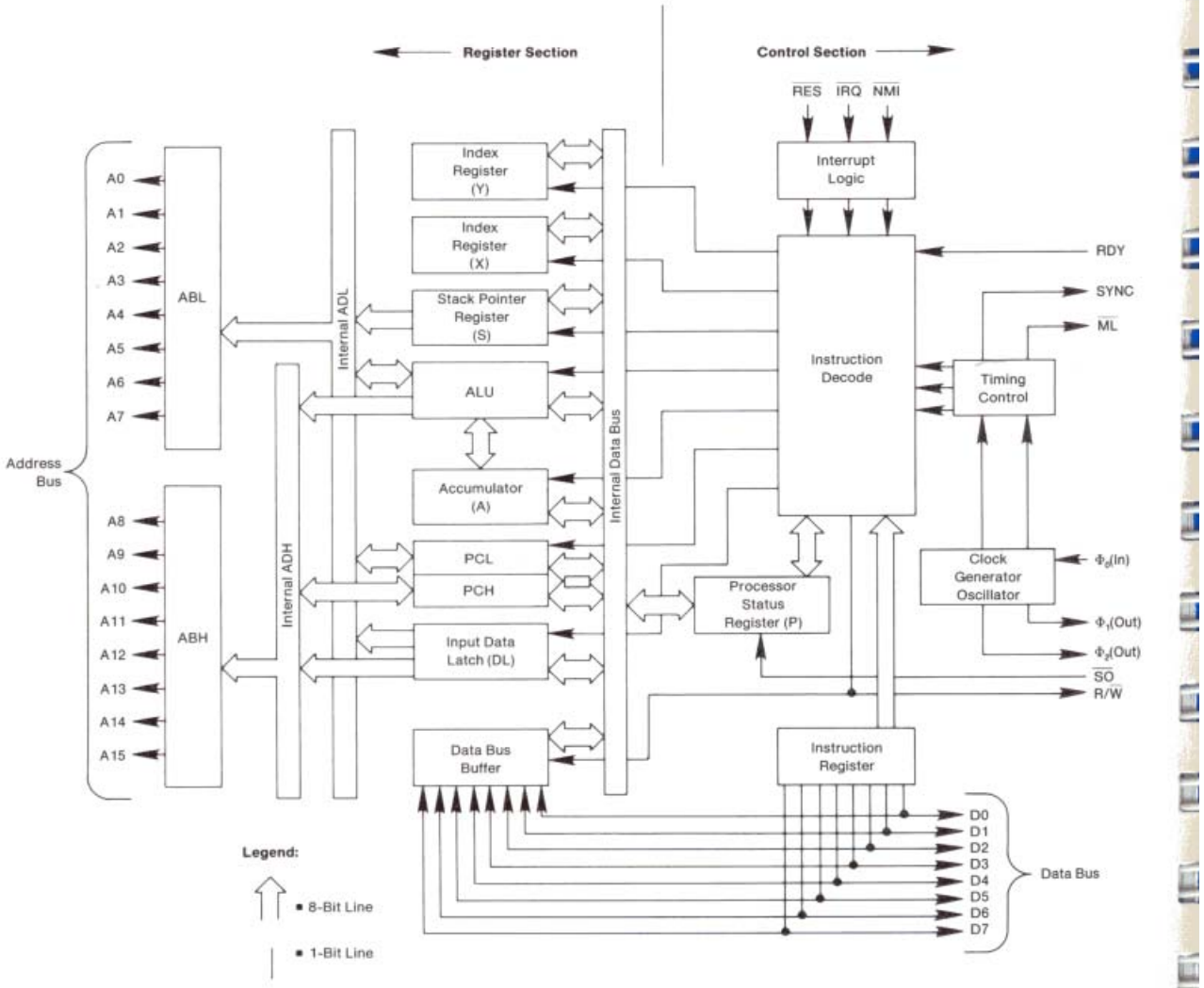
chapter 2

This chapter is an introduction to the microprocessor, the number of separate locations (addresses) it can access, and the addresses set aside for special purposes. The last section of this chapter describes the reset routines, which restore the computer to a known state.

2.1 The 65C02 Microprocessor

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

Figure 2-1. Block Diagram Model of 65C02. Copyright 1982, NCR Corporation. Used by permission of NCR Corporation, Dayton, Ohio.



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 sixteen-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.

The five 8-bit registers in the 65C02 are

- The **accumulator**, or A register. The accumulator is like a desktop 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 1—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 byte in the stack that is currently ready for use.
- A **processor status register**, called the P register. Seven of the eight bits of this register store flags that record the outcome of processor activities, and that can be checked by later instructions to determine what the processor should do next.

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.

2.2 Overview of the Address Space

The Apple IIc's 65C02 microprocessor can address 65,536 (64K) memory locations. All of 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 using soft switches.

Soft switches are described in sections 2.4 and 2.5.

Note: When referring to memory space, **K** stands for 1024, which is 2 to the tenth power. It is called *K* because 1024 is very close to the value 1000, which has long been abbreviated K for *Kilo*. Some early computers even saved the extra 24 locations for spares.

RAM stands for random-access (readable and writable) memory. **ROM** means read-only memory. Refer to the Glossary for further information.

There are two other ROMs in the Apple IIc: one to generate characters corresponding to keystrokes (section 11.7), and another to generate characters for display (section 11.9). 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.

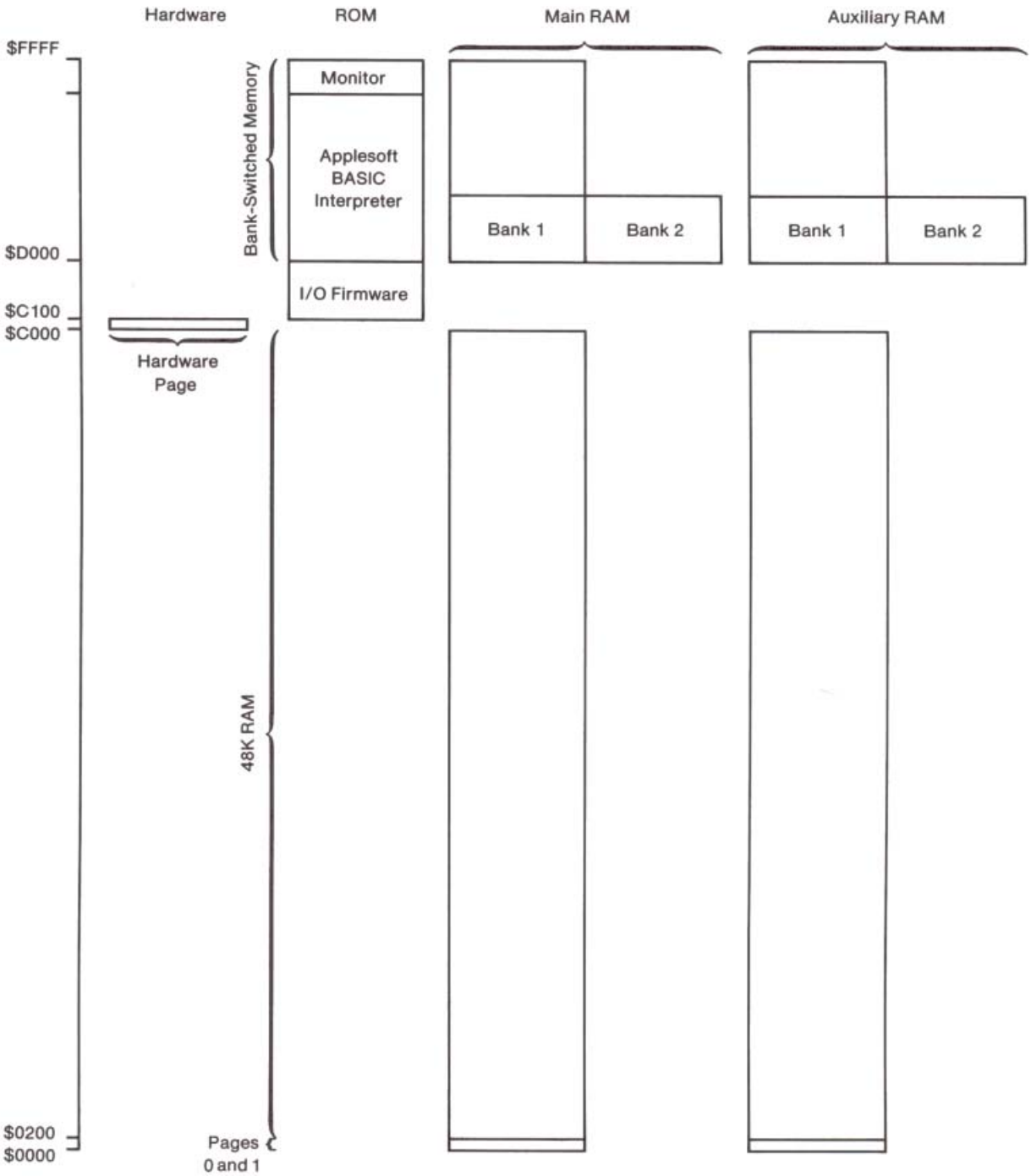
A block of 256 address locations is called a **page**. A one-byte address counter or 8-bit register can specify one of 256 different locations. Thus, page 0 consists of memory locations from 0 to 255 (hexadecimal $0 to $FF), inclusive; page 1 consists of locations 256 to 511 (hexadecimal $100 to $1FF); and so on. In this manual, all page numbers (except some of the low-numbered ones) 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.

2.3 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, three or even 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, switches in the hardware page control which blocks the processor is to use.

Figure 2-2. Apple IIc Memory Map



2.3.1 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.

2.3.2 Auxiliary RAM Addresses ($0000-$BFFF and $D000-$FFFF)

The Apple IIc has another 64K of auxiliary RAM built in. Some or all of auxiliary memory is present in an Apple IIe with one of the 80-column cards installed (Appendix F), but there is no auxiliary RAM in the Apple II or II Plus. This portion of RAM cannot be used simultaneously with main RAM; you must use the soft switches described in this chapter to select main or auxiliary memory for a given range of addresses.

2.3.3 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 in section 2.4.2.

Pages $C1 through CF (addresses $C100 through $CFFF) contain I/O firmware. The following associations apply for the Apple IIc:

- Serial port 1 (RS-232 device) firmware entry points are on page $C1.
- Serial port 2 (communication device) firmware entry points are on page $C2.
- Video output firmware entry points are on page $C3; the enhanced video firmware and miscellaneous I/O support routines occupy pages $C8 through $CF.
- Mouse firmware entry points are on page $C4.
- Disk 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.

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

Pages $D0 through $F7 (addresses $D000 through $F7FF) contain the Applesoft Interpreter firmware.

Pages $F8 through $FF (addresses $F800 through $FFFF) contain the Monitor, which is described in Chapter 10. Monitor routines that make various input and output procedures easier are described in Chapters 3 through 9.

2.3.4 Hardware Addresses ($C000-$C0FF)

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

The Apple IIc's built-in input and output functions, and the switching of blocks of address space, all take place via locations on the $C0 page—that is, in the address range $C000 through $C0FF. This chapter describes the address space (memory) switches.

The hardware functions on this page fall into five basic categories:

Bit numbering in a byte is explained in Appendix H.

- **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. (This data is guaranteed valid only when bit 7 = 1.)
- **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 (section 2.4). Four of these eight switches require that your program read them twice in succession to activate them.

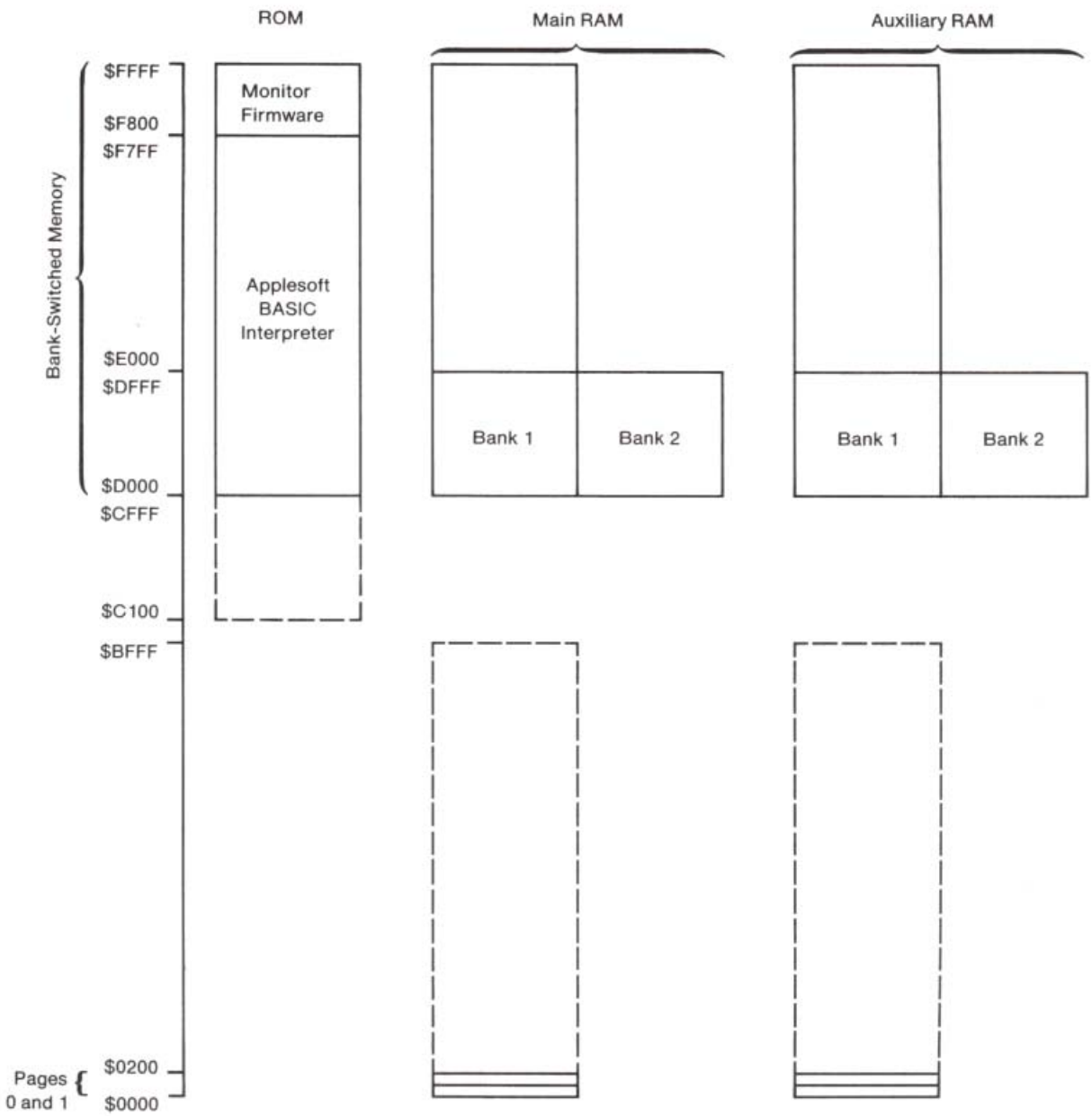
2.4 Bank-Switched Memory

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 (section 2.5).

Section 2.4.1 discusses what functions various addresses are set aside for; section 2.4.2 describes how to select the memory banks you want.

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.

Figure 2-3. *Bank-Switched Memory*



2.4.1 Page Allocations

Pages zero and one 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 zero, or zero page. However, the Monitor, the interpreters, and the operating systems all make extensive use of page zero, too. One way to avoid conflicts is to use only those page-zero locations not already used by these other programs. But there is another way.

See Table B-1 in Appendix B.

As you can see from Table B-1, page zero 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 zero, use that part, then restore the previous contents to page zero, restore interrupts to their previous state, and then pass control to another program.

Page $01 (The 65C02 Stack)

The 65C02 microprocessor uses page 1 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 Through $FF (ROM and RAM)

All these memory banks are controlled by the soft switches described in section 2.4.2.

The memory address space from $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.

2.4.2 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:

1. Select either RAM or ROM in this memory space.
2. Allow or inhibit (write-protect) writing to the RAM when RAM is selected.
3. Select the first or second 4K-byte bank of RAM in the address space $D000 to $DFFF.
4. 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: There is no way to check whether write protection is on or off.

Table 2-1. Bank Select Switches

Action	Hex	Name	Function
R	$C080		Read RAM; no write; use $D000 bank 2.
RR	$C081		Read ROM; write RAM; use $D000 bank 2.
R	$C082		Read ROM; no write; use $D000 bank 2.
RR	$C083		Read and write RAM; use $D000 bank 2.
R	$C088		Read RAM; no write; use $D000 bank 1.
RR	$C089		Read ROM; write RAM; use $D000 bank 1.
R	$C08A		Read ROM; no write; use $D000 bank 1.
RR	$C08B		Read and write RAM; use $D000 bank 1.
R7	$C011	RDBNK2	Read whether $D000 bank 2 (1) or bank 1 (0).
R7	$C012	RDLCRAM	Reading RAM (1) or ROM (0).
W	$C008	ALTZP	Off: use main bank, page 0 and page 1.
W	$C009	ALTZP	On: use auxiliary bank, page 0 and page 1.
R7	$C016	RDALTZP	Read whether auxiliary (1) or main (0) bank.

Figure 2-4. *Read ROM*

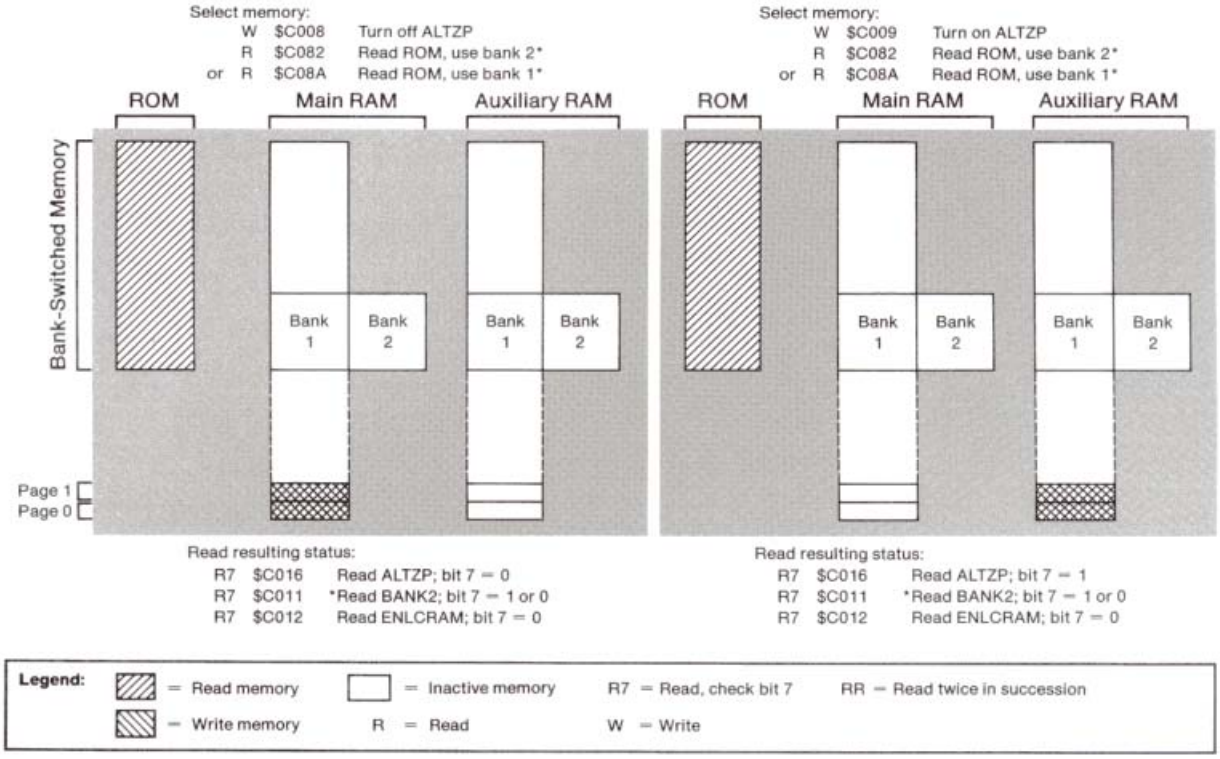


Figure 2-5. *Read ROM, Write RAM, and Use First $D0 Bank*

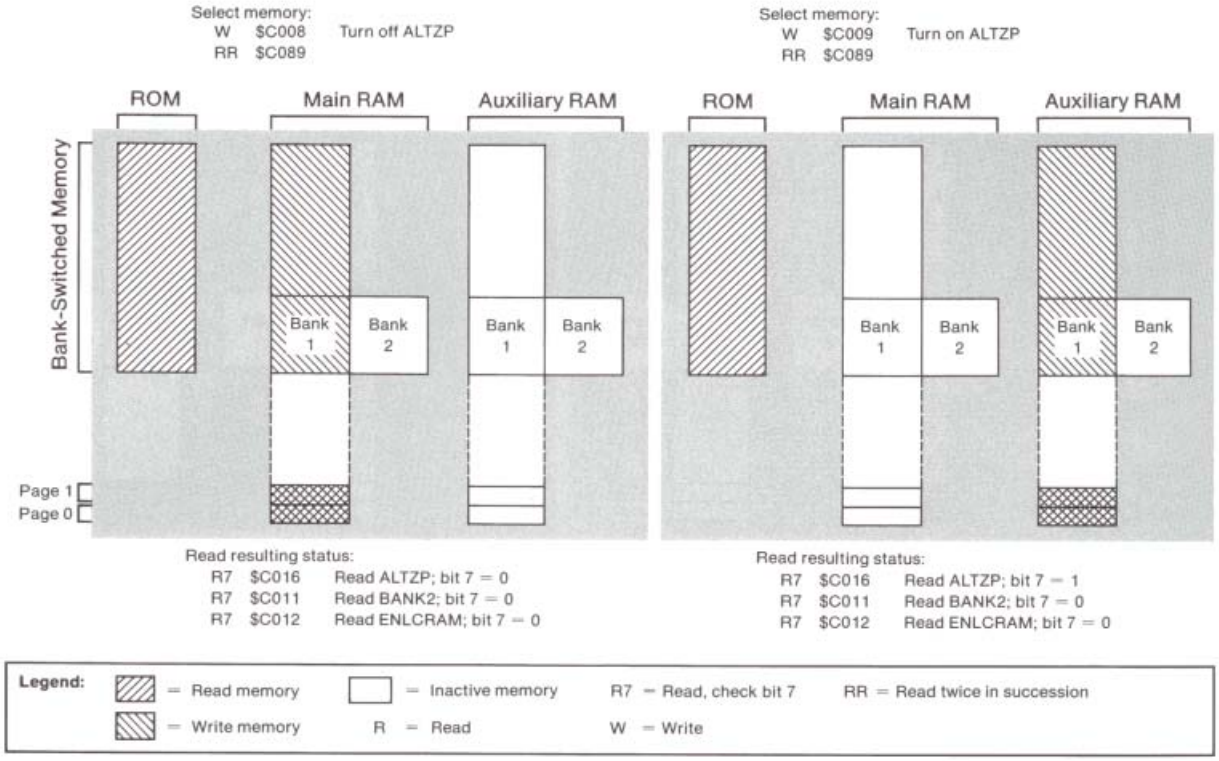


Figure 2-6. *Read ROM, Write RAM, and Use Second $D0 Bank*

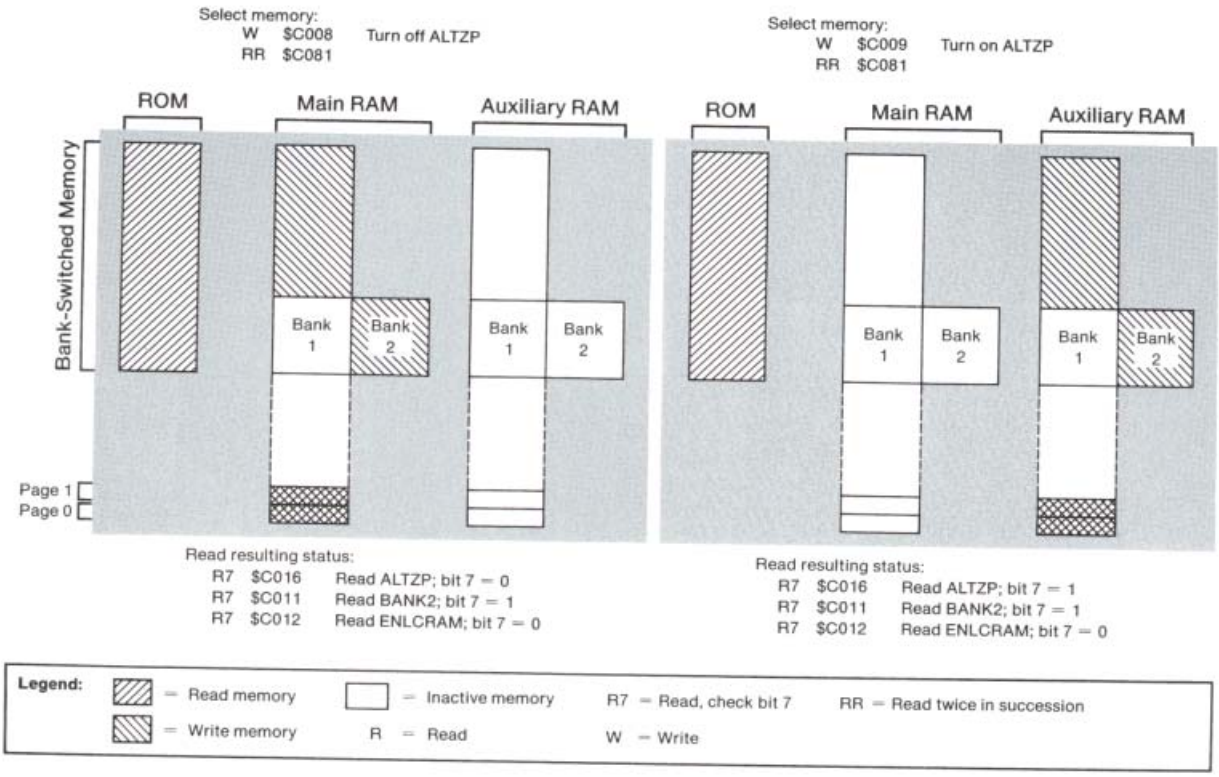


Figure 2-7. *Read RAM and Use First $D0 Bank*

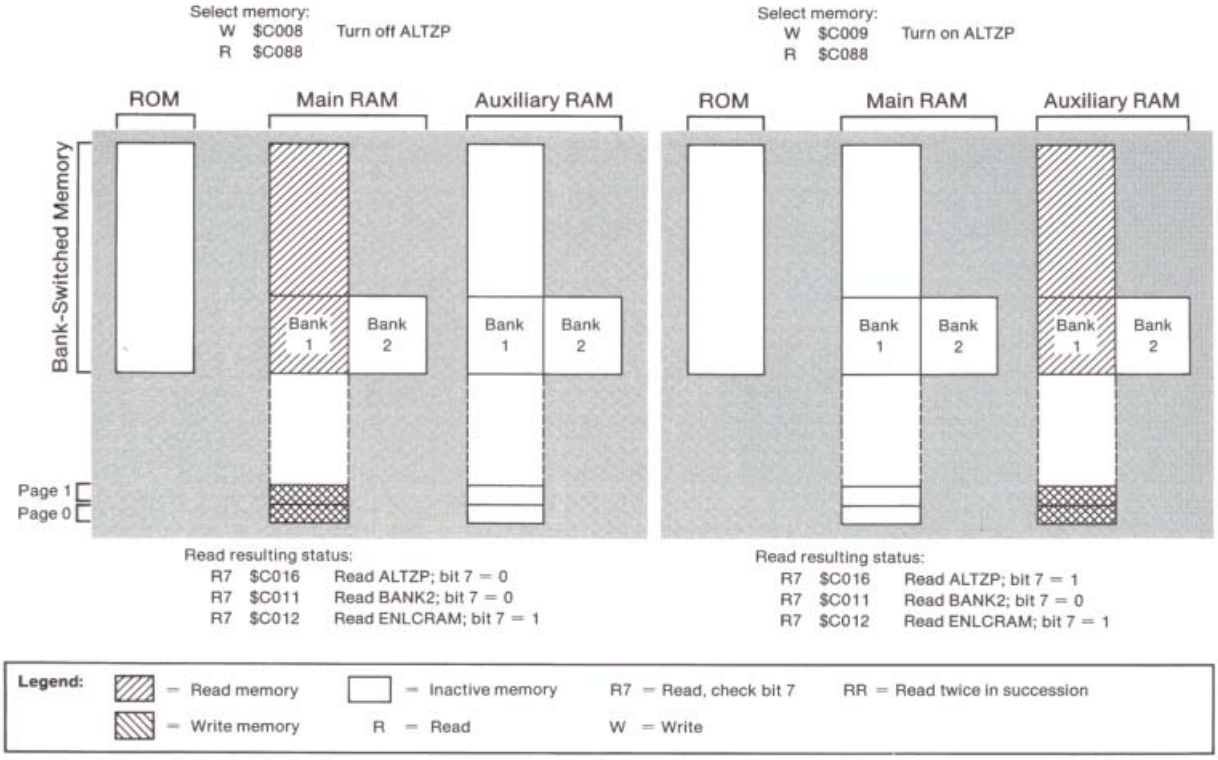


Figure 2-8. Read RAM and Use Second $D0 Bank

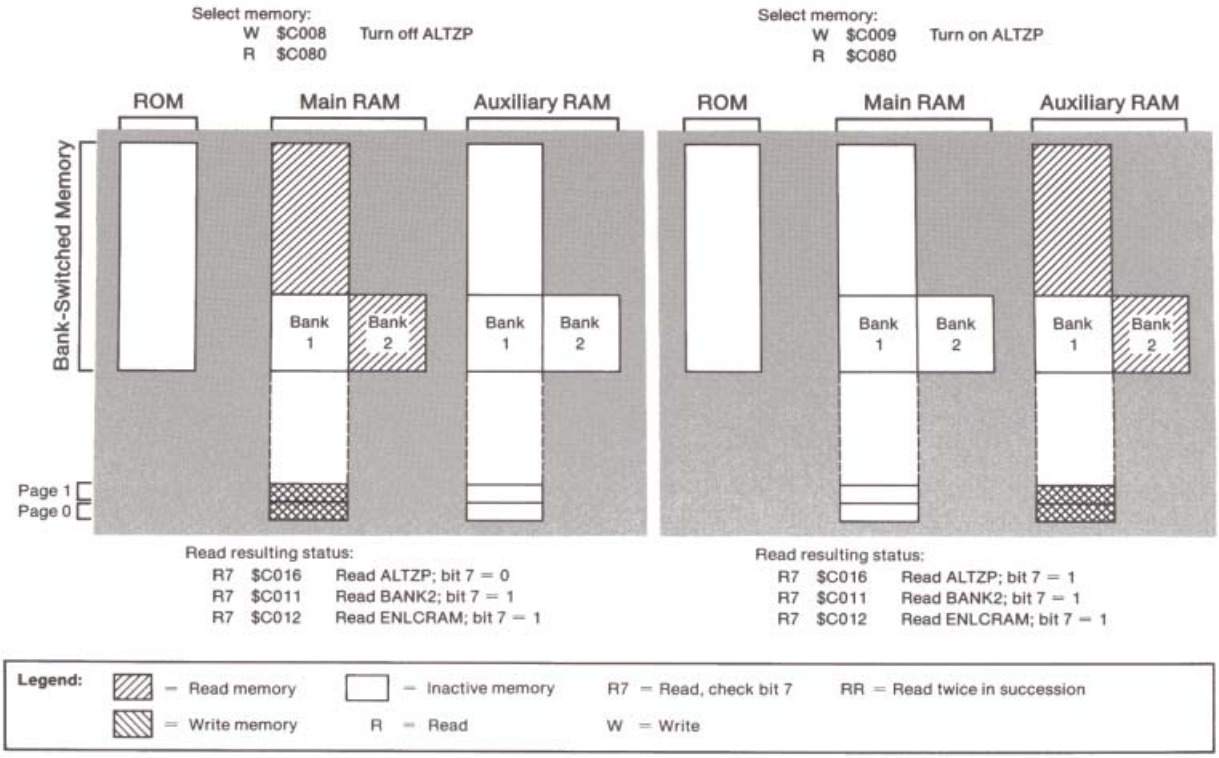


Figure 2-9. *Read and Write RAM and Use First $D0 Bank*

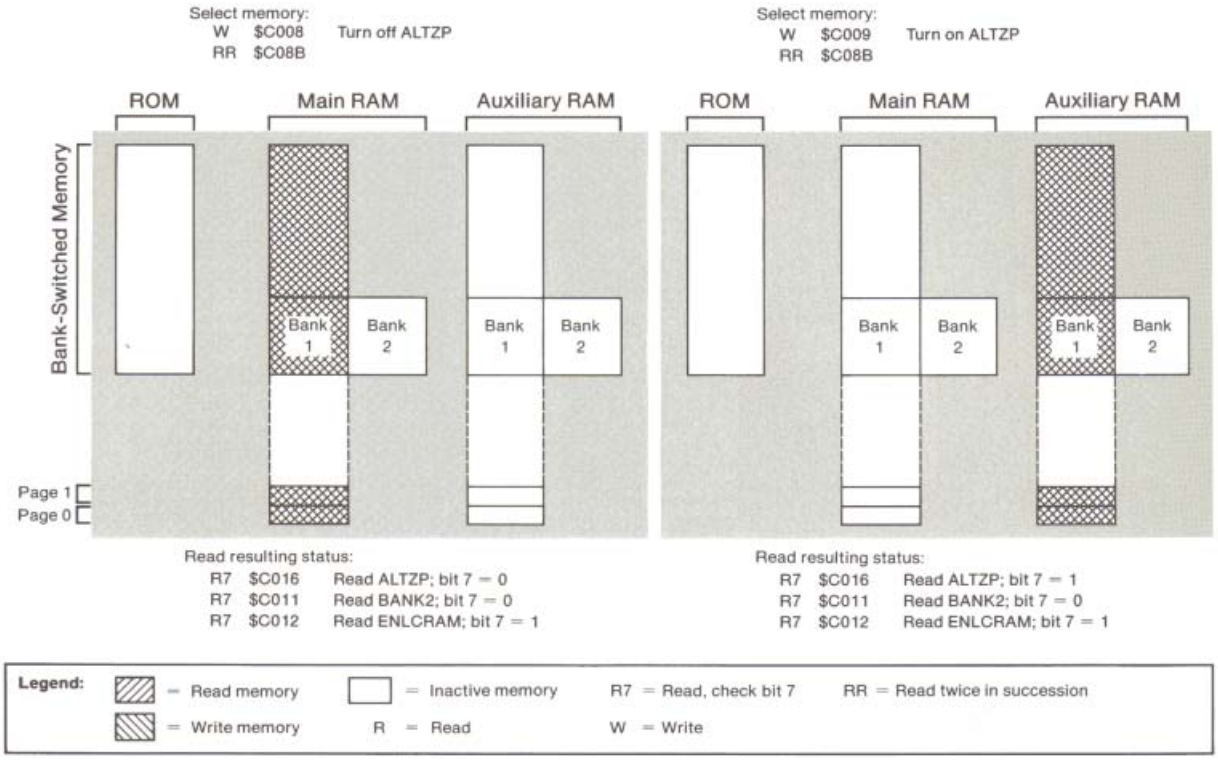
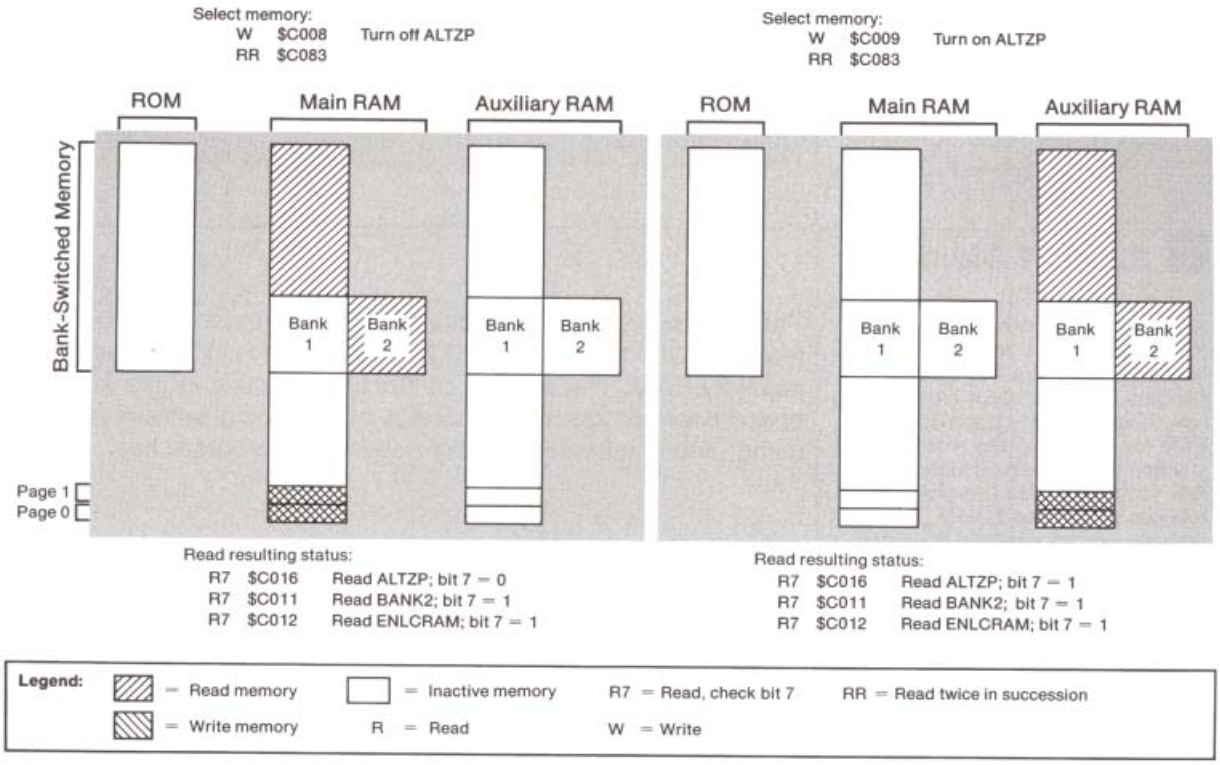


Figure 2-10. Read and Write RAM and Use Second $D0 Bank



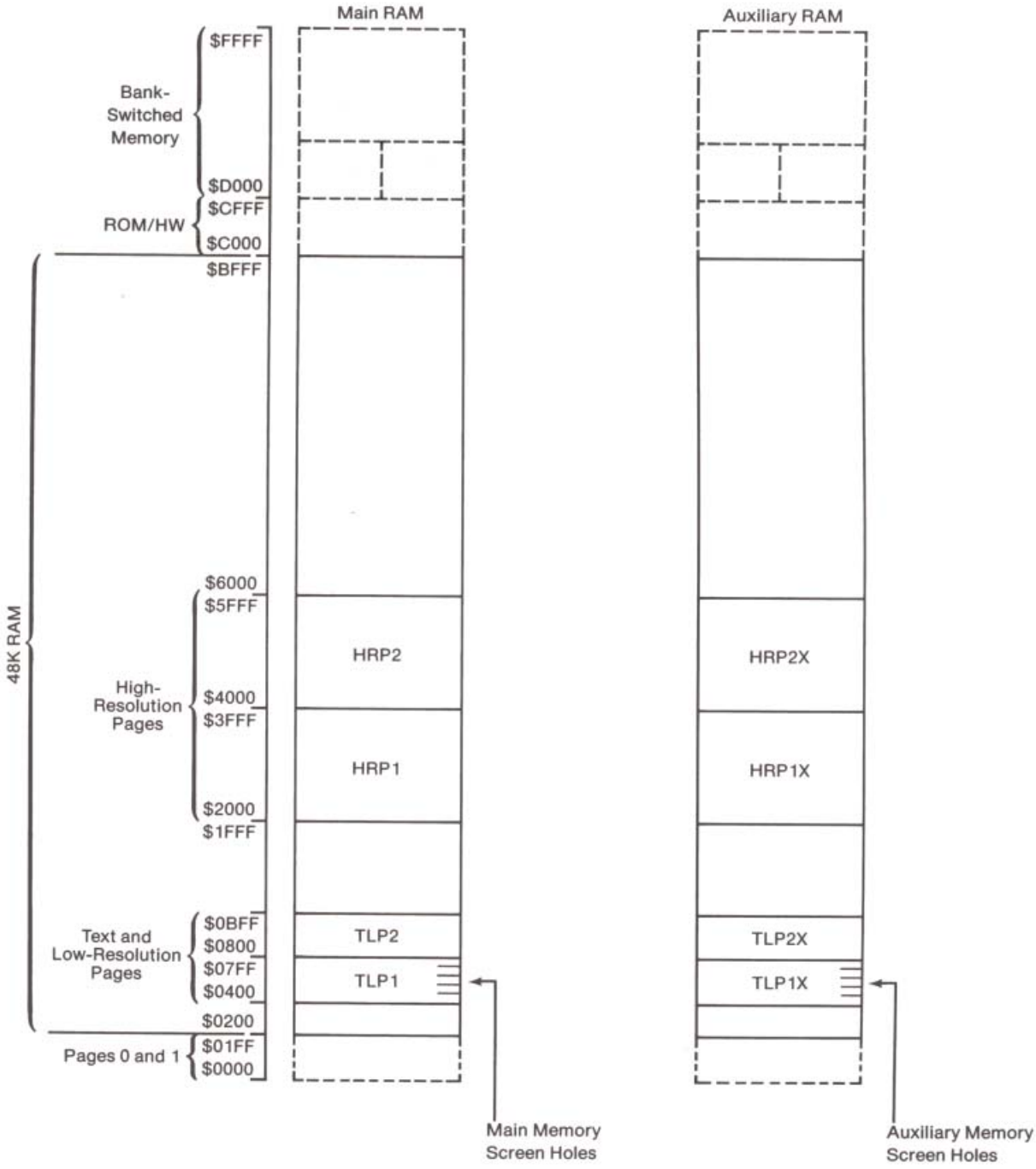
Note: You can't read one RAM bank and write to the other; if you select either RAM bank for reading, you get that one for writing as well. However, you can read ROM and write RAM (Figures 2-5 and 2-6), which makes it easy to transfer firmware to bank-switched RAM if you want to use it with a program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

2.5 48K Memory

Section 2.5.1 describes the memory pages the hardware and firmware use for various purposes. Sections 2.5.2 and 2.5.4 explain how to select main or auxiliary RAM for read/write and video storage, respectively. Section 2.5.3 tells you how to use firmware routines to transfer data or program control between main and auxiliary RAM.

The 48K memory space (actually, 47-1/2K) extends from location $200 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.

Figure 2-11. 48K Memory Map



2.5.1 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.

The term **system** refers to the computer and its firmware plus whatever operating system you are using.

Note: 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 (section 3.2.3) uses page 2 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 2.

Page $03 (Global Storage and Vectors)

Refer to Appendix D and to the appropriate programmer and reference manuals for operating system use of page 3.

The Monitor and operating systems use parts of page 3 for global storage and vectors. Table 2-7 shows the part of page 3 the built-in firmware uses.

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. Section 2.6 discusses the global storage and vectors found on page $03.

Pages $04 Through $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.

See Chapter 5.

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 section 3.4.6.

There are 128 locations in pages $04-$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 Through $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: 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)

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 TPLX.

Pages $20 Through $3F (High-Resolution Page 1)

The primary high-resolution-graphics display buffer, called 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 Through $5F (High-Resolution Page 2)

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

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

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

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

2.5.2 Using 48K Memory Switches

Two switches select main or auxiliary RAM in the 48K memory space (Table 2-2): 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.

Warning
This discussion assumes that the 80STORE switch, used to control display memory, is off.

For details, refer to section 2.5.4.

Each switch has three locations assigned to it: 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.

Figures 2-12 and 2-13 illustrate how the switches work.

Table 2-2. 48K Memory Switches. ***Note:*** *80STORE must be off to switch all memory in this range, including display memory (Table 2-6).*

Action	Hex	Name	Function
W	$C002	RAMRD	Off: read main 48K RAM.
W	$C003	RAMRD	On: read auxiliary 48K RAM.
R7	$C013	RDRAMRD	Read whether main (0) or auxiliary (1)
W	$C004	RAMWRT	Off: write to main 48K RAM.
W	$C005	RAMWRT	On: write to auxiliary 48K RAM.
R7	$C014	RDRAMWRT	Read whether main (0) or auxiliary (1)

Figure 2-12. 48K RAM Selection: Split Pairs

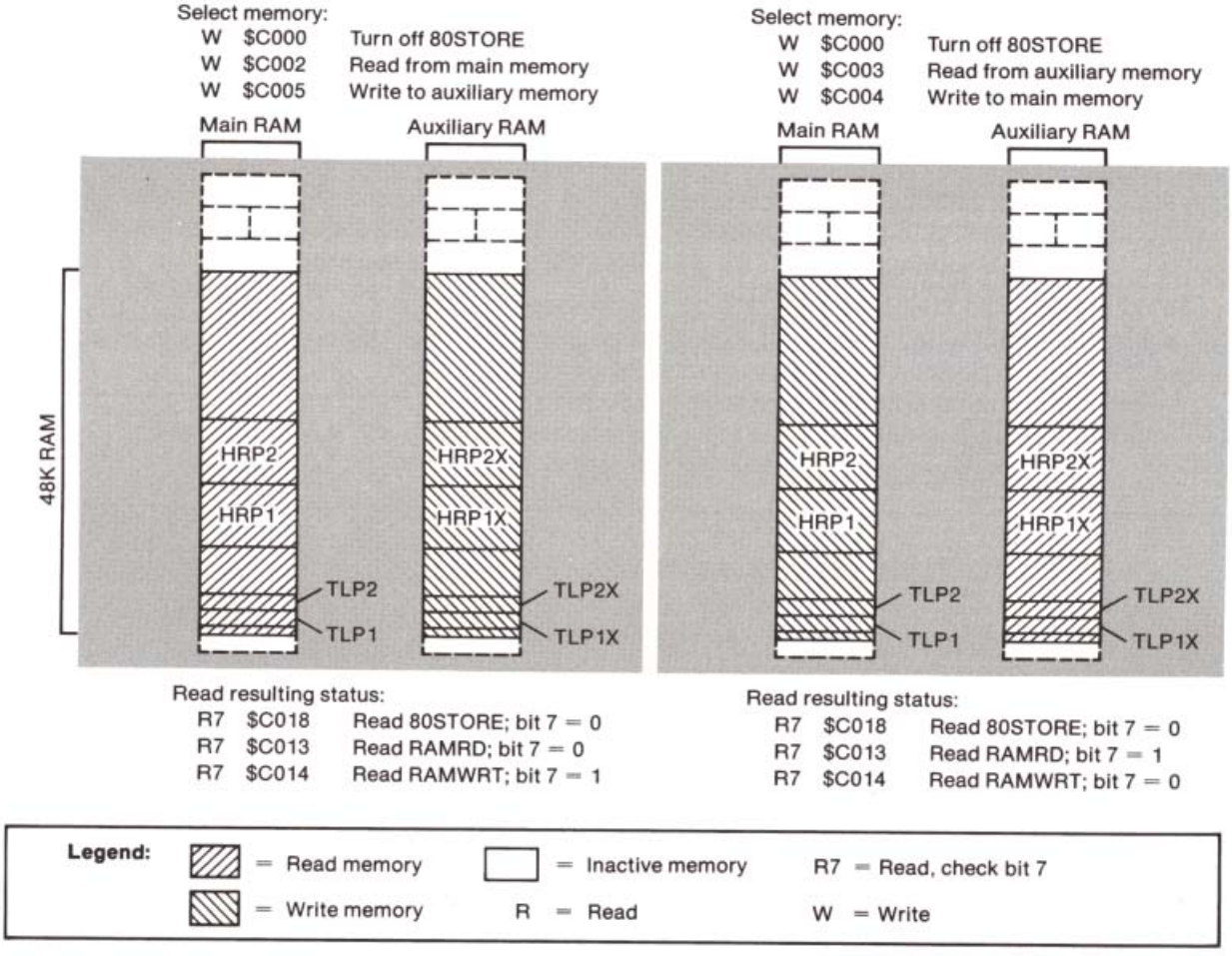
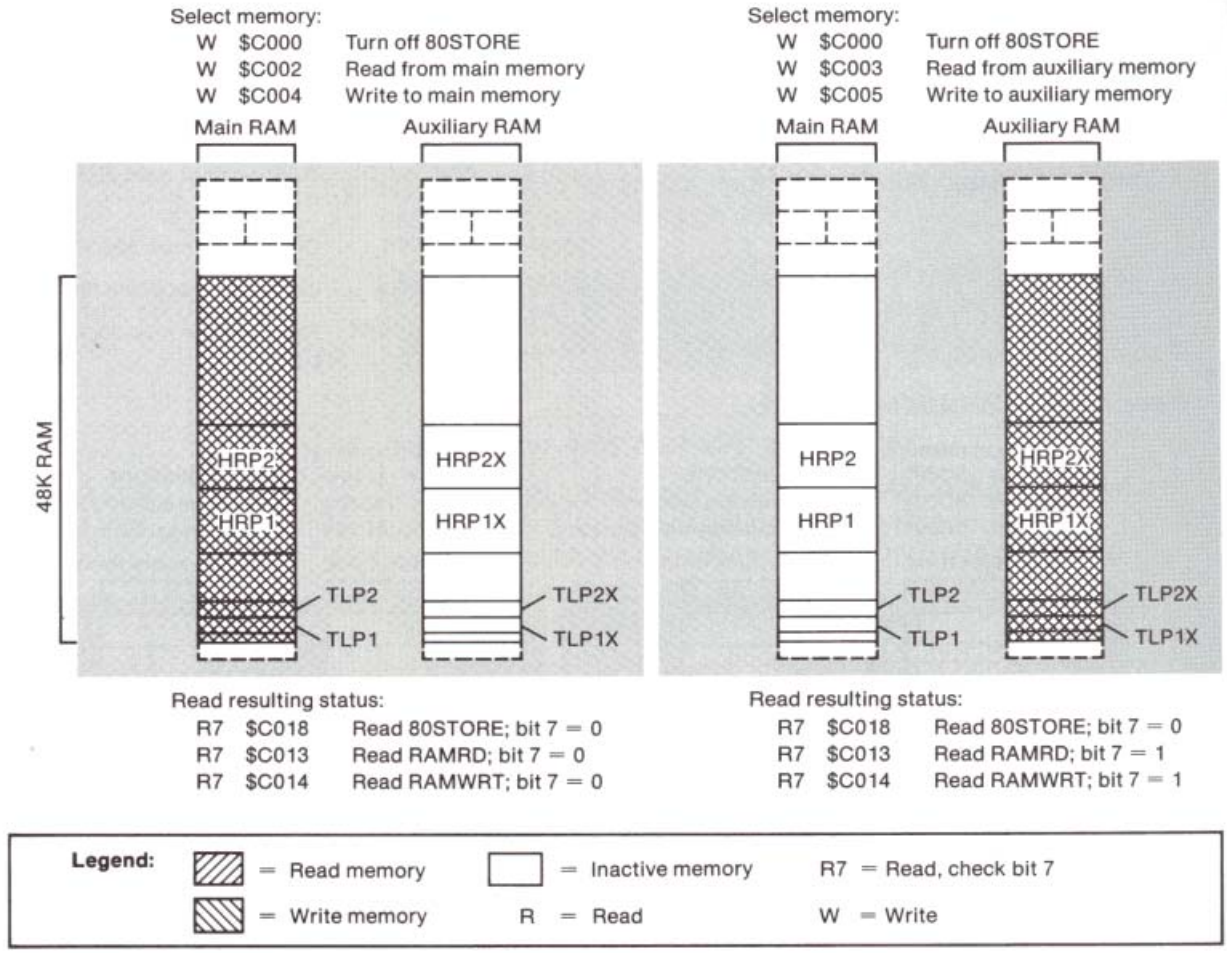


Figure 2-13. 48K RAM Selection: One Side Only



2.5.3 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 make it possible to move between main and auxiliary memory without having to manipulate the soft switches described in Section 2.5.2.

Note: 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

Action	Hex	Name	Function
JSR	$C311	MOVEAUX	Moves data blocks between main and auxiliary 48K memory.
JMP	$C314	XFER	Transfers 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 zero 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 zero, page one or pages $D0 through $FF). MOVEAUX uses page zero 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.

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

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.

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 zero and stack you want to use.

Table 2-5. Parameters for XFER Routine

Name or Location	Parameter Passed	
Carry		1 = Transfer from main to auxiliary memory 0 = Transfer from auxiliary to main memory
Overflow		1 = Use page zero and stack in auxiliary memory 0 = Use page zero and stack in main memory
	$3ED	Program starting address, low-order byte
	$3EE	Program starting address, high-order byte
	X,Y,A	These registers are preserved.

Put the transfer address into the two bytes at locations $3ED and $3EE, 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 zero 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.

2.5.4 Using Display Memory Switches

Section 2.5.2 discusses how to select main or auxiliary RAM for the 48K memory space. 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.

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.

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).

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 (TLP) pages only. If HIRES is on, then PAGE2 switches between TLP pages and High-Resolution Graphics (HRP) pages.
- 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 2-15 and 2-16).

See Chapter 5 for a discussion of text and high-resolution page and the high-resolution pages.

Table 2-6. *Display Memory Switches*

Action	Hex	Name	Function
W	$C000	80STORE	Off: RAMRD and RAMWRT determine RAM locations.
W	$C001	80STORE	On: PAGE2 switches between TLP1 and TLP1X, and (if HIRES on) between HRP1 and HRP1X.
R7	$C018	RD80STORE	Read whether 80STORE on (1) or off (0)
R	$C054	PAGE2	Off: select TLP1 and HRP1.
R	$C055	PAGE2	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.
R7	$C01C	RDPAGE2	Read whether PAGE2 on (1) or off (0)
R	$C056	HIRES	Off: display text and low-resolution page.
R	$C057	HIRES	On: display high-resolution pages; make PAGE2 switch between high-resolution pages.
R7	$C01D	RDHIRES	Read whether HIRES on (1) or off (0)

Figure 2-14. *PAGE2 Selections With 80STORE On and HIRES Off*

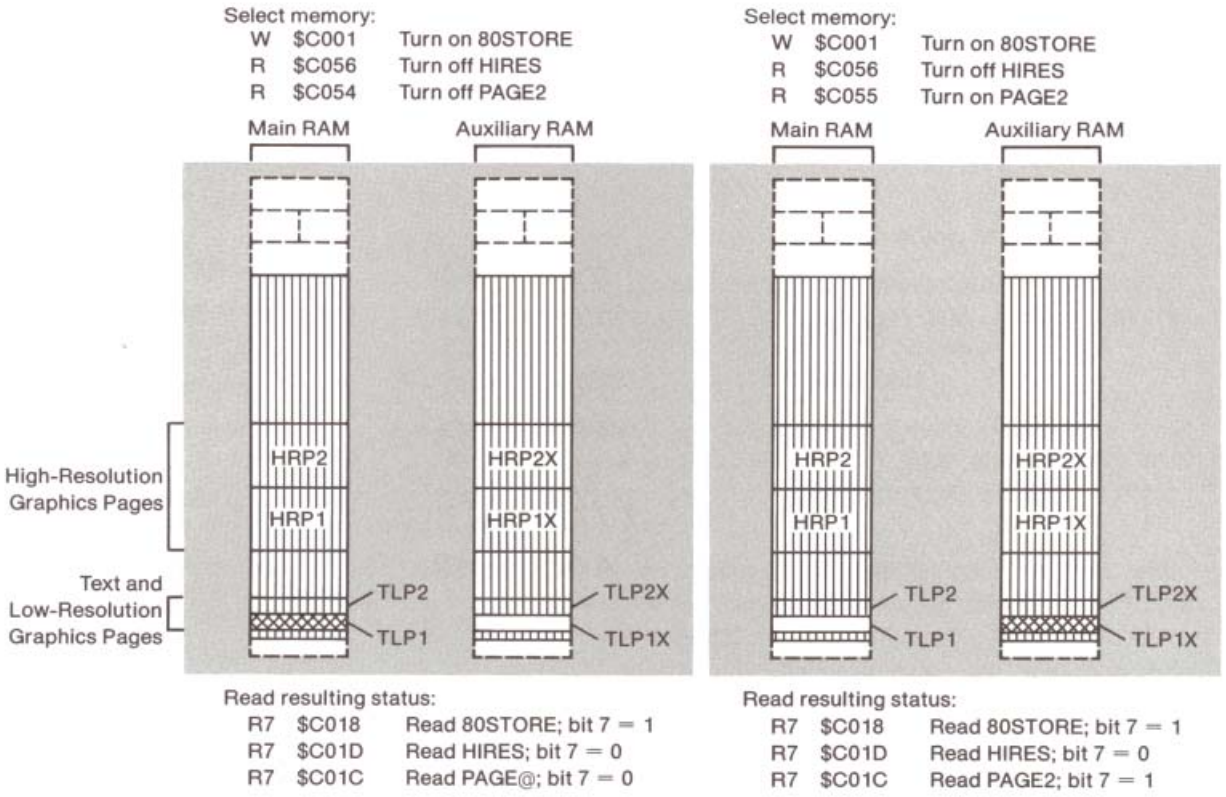
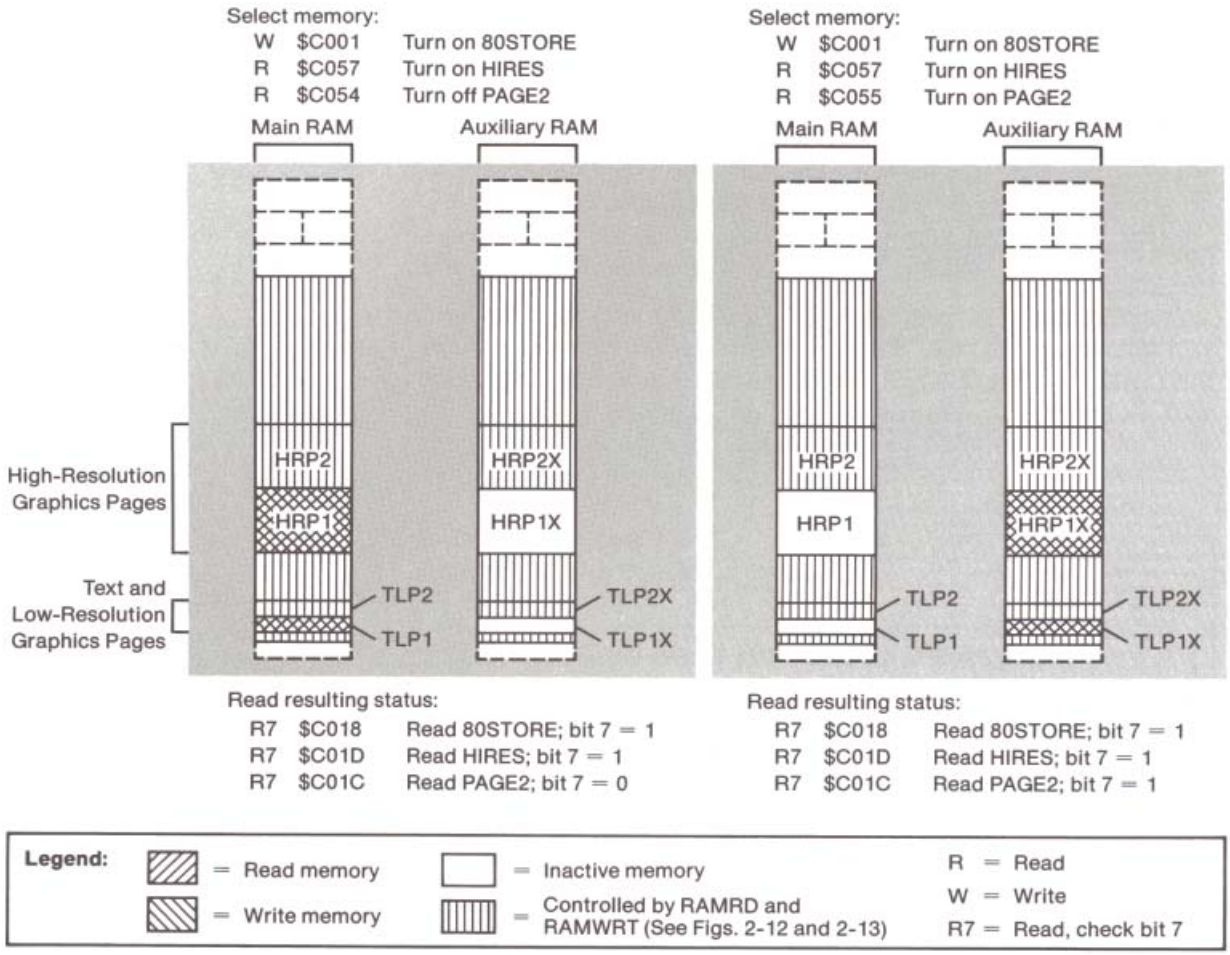


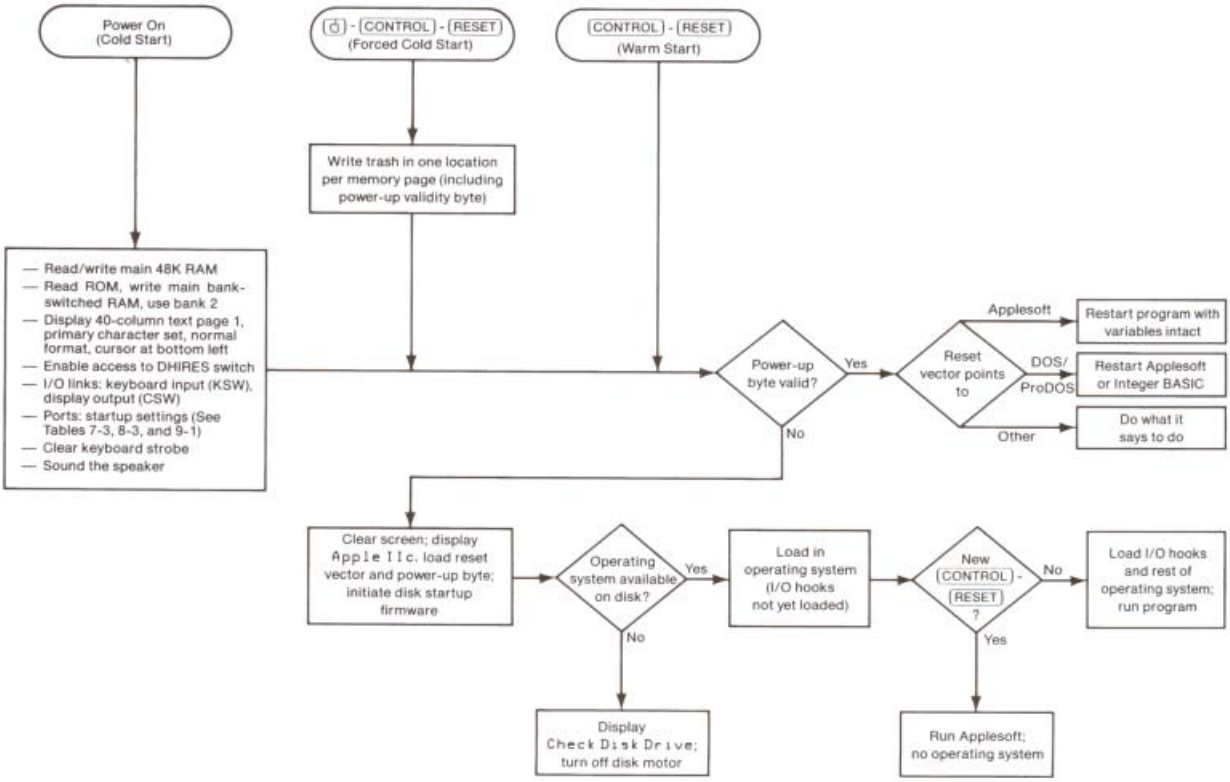
Figure 2-15. PAGE2 Selections With 80STORE On and HIRES On



2.6 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 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 $3F2 and $3F3 (Table 2-7).

Figure 2-16. *Reset Routine Flowchart*



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.

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 in section 2.6.4.

2.6.1 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 //c` at the top of the display. It loads the reset vector and the validity-check byte as described in section 2.6.1, 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.

2.6.2 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).

Note: 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.

2.6.3 Forced Cold Start (-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 ( ), then pressing and releasing (RESET).

Note: 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, ( )-(CONTROL)-(RESET), instead of turning the power off and on.

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 3 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: If you press both [ó] and [é] during power-up or [CONTROL]-[RESET], built-in exercise code is executed. This code is for production and has no end-user value.

2.6.4 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 $3F2 and $3F3. It then stores a validity-check byte, also called the **power-up byte**, at location $3F4. 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.

Table 2-7. Page 3 Vectors

Vector Address	Vector Function
$3F0 $3F1	Address of the subroutine that handles BRK requests (normally $59, $FA).
$3F2 $3F3	Reset vector (see text).
$3F4	Power-up byte (see text).
$3F5 $3F6 $3F7	Jump instruction to the subroutine that handles Applesoft & commands (normally $4C, $58, $FF).
$3F8 $3F9 $3FA	Jump instruction to the subroutine that handles user CONTROL-Y commands.
$3FB $3FC $3FD	Jump instruction to the subroutine that handles non-maskable interrupts (not used on Apple IIc).
$3FE $3FF	Interrupt vector (address of the subroutine that handles interrupt requests (Appendix E).

Introduction to Apple IIc I/O

This chapter is an introduction to the input/output capabilities of the Apple IIc; the next six chapters discuss these capabilities in detail. The remainder of this chapter outlines the common elements of I/O processing—standard I/O links and features, standard port entry points, protocols and storage locations, and direct I/O.

3.1 The Standard I/O Links

When you call one of the character I/O subroutines (COUT and RDKEY), the first thing that happens is an indirect jump to an address stored in programmable memory. Memory locations used for transferring control to other subroutines are sometimes called **vectors**. In this manual, the locations used for transferring control to the I/O subroutines are called the **I/O links**.

In an Apple IIc running without an operating system, each I/O link is normally 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 jump to the link addresses instead of calling the standard subroutines directly, you ensure that your program will work properly in conjunction with other software, such as 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 (flashing checkerboard cursor), and of C3KEYIN and C3COUT1 if that firmware is on (inverse solid cursor).

3.1.1 Changing the Standard I/O Links

The standard I/O links are two pairs of locations in the Apple IIc 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, for 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 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 memory 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, for keyboard input switch. Individually, location $38 is called KSWL (for 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. This address is 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 this link address 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 character input and output routines that are currently active.

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, BASIC programs should issue an empty PRINT statement with a carriage return (to be sure that what follows begins a new line), then another 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 CSW or KSW, assembly-language programs running under DOS should call the subroutine at location $3EA. 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.

3.2 Standard Input Features

GETLN also provides on-screen editing features: see section 3.2.5.

The Apple IIc's firmware includes two different subroutines for reading from the keyboard. One subroutine is named RDKEY (*read key*). It 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. The other subroutine is named GETLN (*get line*). GETLN accepts a sequence of characters terminated with a carriage return. Thus GETLN allows line-oriented input using the current input routine.

3.2.1 RDKEY Input Subroutine

A program gets 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).

3.2.2 KEYIN Input Subroutine

KEYIN is the standard input subroutine. When called, it displays a cursor, waits until the user presses a key, then returns to the calling program with the ASCII code of the key pressed in the accumulator.

If the enhanced video firmware is inactive, KEYIN displays a cursor by alternately storing a checkerboard block in the cursor location, then storing the original character, then the checkerboard again. If the firmware is active, C3KEYIN inverts 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 there is no way to predict what it will be after a key is pressed. A program that reads from the keyboard can use this value as a random number or as a seed for a pseudo-random number routine.

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

3.2.3 GETLN Input Subroutine

Programs often need strings of characters as input. While it is possible to call RDKEY repeatedly to get several characters from the keyboard, there is a more powerful subroutine you can use. This routine is named GETLN, which stands for *get line*, and 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 $200 to $2FF. GETLN also provides the user with on-screen editing and control features.

Section 3.2.5 describes other features of GETLN.

The first thing GETLN does when you call it is to display a prompting character, called simply a **prompt**. The prompt indicates to the user that the program is waiting for input. Different programs use different prompt characters, helping to remind the user which program is requesting the input. For example, an INPUT statement in a BASIC program displays a question mark (?) as a prompt. The prompt characters used by the different programs on the Apple IIc are shown in Table 3-1.

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.

Note: Applesoft uses GETLN1 ($FD6F) when a program is executing. GETLN1 does not print a prompt.

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)

Control characters echoed by GETLN are not executed.

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.

GETLN stores the characters in its buffer, starting at memory location $200 and using the X register to index the buffer. GETLN continues to accept and display characters until 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.

3.2.4 Escape Codes With GETLN

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

Note: Be sure to release (ESC) right away. If you hold it too long, the auto-repeat mechanism will begin, which may cancel the ESC.

In escape mode, you can keep using the arrow keys and the cursor-motion keys (I), (J), (K), and (M) without pressing (ESC) 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-motion key.

Note: The escape codes with the arrow keys are the standard cursor-motion keys on the Apple IIc. The escape codes with (I), (J), (K), and (M) are the standard cursor-motion keys on the Apple II and II Plus, and are present on the Apple IIc for compatability.

Table 3-2. *Escape Codes With GETLN. (1) Cursor-control key: see text. (2) This code functions only when the enhanced video firmware is active.*

Escape Code	Function	Notes
(ESC) (@)	Clears the window and *homes* the cursor (places it in the upper-left corner of the screen), then exits from escape mode.	
(ESC) (A) or (a)	Moves the cursor right one line and exits from escape mode.	
(ESC) (B) or (b)	Moves the cursor left one line and exits from escape mode.	
(ESC) (C) or (c)	Moves the cursor down one line and exits from escape mode.	
(ESC) (D) or (d)	Moves the cursor up one line; exits from escape mode.	
(ESC) (E) or (e)	Clears to the end of the line; exits from escape mode.	
(ESC) (F) or (f)	Clears to the bottom of the window; exits from escape mode.	

Table 3-2—Continued. *Escape Codes With GETLN*

Escape Code	Function	Notes
ESC I or i or ESC ↑	Moves the cursor up one line; remains in escape mode.	1
ESC J or j or ESC ←	Moves the cursor left one space; remains in escape mode.	1
ESC K or k or ESC →	Moves the cursor right one space; remains in escape mode.	1
ESC M or m or ESC ↓	Moves the cursor down one line; remains in escape mode.	1
ESC 4	Switches to 40-column mode; activates the enhanced video firmware; sets links to C3KEYIN and C3COUT1; restores normal window size (Table 3-5); exits from escape mode.	2
ESC 8	Switches to 80-column mode; activates the enhanced video firmware; sets links to C3KEYIN and C3COUT1; restores normal window size (Table 3-5); exits from escape mode.	2
ESC CONTROL-D	Disables control characters; only carriage return, line feed, BELL, and backspace have an effect when printed.	
ESC CONTROL-E	Reactivates control characters.	
ESC CONTROL-Q	Deactivates the enhanced video firmware; sets links to KEYIN and COUT1; restores normal window size (Table 3-5): exits from escape mode.	2

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.

3.2.5 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 ← (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 will replace the character you backspaced over, both on the display and in the line buffer.

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.

Each time you press ←, it moves the cursor left and deletes another character, until you are back at the beginning of the line. If you then press ← one more time, you have effectively canceled the line, and GETLN issues a carriage return and displays the prompt.

Retype

→ (or CONTROL-U) has a function that is complementary to the backspace function. When you press →, 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.

See section 3.2.4.

3.3 Standard Output Features

The standard output routine is named COUT (pronounced *C-out*) which stands for 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 below.

3.3.1 COUT Output Subroutine

Your program makes a subroutine call to COUT at memory location $FDED with a character in the accumulator. COUT then passes control via the output link CSW to the current output subroutine, normally COUT1 or 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-hand 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 will 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 $57B. When the 80-column video firmware is active, use $57B instead of CH.

3.3.2 Control Characters With COUT1

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.

Escape codes are described in section 3.2.4.

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.

3.3.3 Control Characters With C3COUT1

C3COUT1 does not display control characters. Instead, the control characters listed in the two parts of Table 3-4 are used to initiate some action by the firmware. Other control characters are ignored. Most of the functions listed here can also be invoked by using the appropriate escape code. The stop-list function, described separately, can only be invoked from the keyboard.

Escape Codes: see section 3.2.4.

Table 3-4. *Control Characters With C3COUT1. (1) Only available when enhanced video firmware is active. (2) Only works from the keyboard. (3) Doesn't work from the keyboard.*

Control Character	ASCII Name	Apple IIc Name	Action Taken by C3COUT1	Notes
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.	1,3
CONTROL-L	FF	home and clear	Moves cursor position to upper-left corner of window and clears window.	1,3
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.	1,3
CONTROL-O	SI	inverse	Sets display format inverse.	1,3
CONTROL-Q	DC1	40-column	Sets display to 40-column.	1,3
CONTROL-R	DC2	80-column	Sets display to 80-column.	1,3
CONTROL-S	DC3	stop-list	Stops listing characters on the display until another key is pressed.	2
CONTROL-U	NAK	quit	Deactivates enhanced video firmware.	1,3
CONTROL-V	SYN	scroll	Scrolls the display down one line, leaving the cursor in the current position.	1,3
CONTROL-W	ETB	scroll-up	Scrolls the display up one line, leaving the cursor in the current position.	1,3
CONTROL-X	CAN	disable MouseText	Disable MouseText character display; use inverse uppercase.	
CONTROL-Y	EM	home	Moves cursor position to upper-left corner of window (but doesn't clear).	1,3
CONTROL-Z	SUB	clear line	Clears the line the cursor position is on.	1,3

Table 3-4—Continued. *Control Characters With C3COUT1*

Control Character	ASCII Name	Apple IIc Name	Action Taken by C3COUT1	Notes
CONTROL-[	ESC	enable MouseText	Map 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.	1,3
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).	1,3
CONTROL-_	US	up	Moves cursor up a line, no scroll.	

3.3.4 The Stop-List Feature

When you are using any program that displays text via COUT1 or C3COUT1, you can make it stop updating the display by pressing CONTROL-S (that is, by holding down CONTROL while pressing S). Whenever COUT1 or C3COUT1 gets a carriage return from the program, it checks for CONTROL-S. If it has been pressed, COUT1 or C3COUT1 stops and waits for another keypress. Then it continues normally. The character code of the key you pressed to resume displaying is ignored unless it is CONTROL-C. COUT1 or C3COUT1 passes CONTROL-C back to the program; if it is a BASIC program, this enables you to terminate the program while in stop-list mode.

3.3.5 The Text Window

After you start up the computer or perform a reset, the firmware uses the entire display. However, you can restrict video activity to any rectangular portion of the display you wish. The active portion of the display is called the text window. COUT1 or 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 set the top, bottom, left side, and width of the text window by storing the appropriate values into four locations in memory. This enables your programs to control the placement of text in the display and to protect other portions of the screen from being written over by new text.

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

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

Warning
Be careful not to let the sum of the window width and the leftmost position in the window exceed the width of the display you are using (40- or 80-columns). If this happens, it is possible for COUT1 or C3COUT1 to put characters into memory locations outside the display page, possibly destroying programs or data.

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

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

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

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

Table 3-5. Text Window Memory Locations

Window Parameter	Minimum Location		Normal Values: Value				Maximum Values:			
	80col.		40col.		80col.		40col.		80col.	
	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
Left Edge	32	$20	0	$0	0	$0	0	$0	39	$27	79	$4F
Width	33	$21	0	$0	40	$28	80	$50	40	$28	80	$50
Top Edge	34	$22	0	$0	0	$0	0	$0	23	$17	23	$17
Bottom Edge	35	$23	1	$1	24	$18	24	$18	24	$18	24	$18

3.3.6 Normal, Inverse, and Flashing Text

The form of a displayed character depends on two things: what value is stored in zero page location $32 (the inverse flag), and whether the enhanced video firmware is off or on. The effects of the inverse flag are discussed in the next two subsections.

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.

Both these display character sets are described in Chapter 5.

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 this set.

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

To display a character, load it in the accumulator, and then jump to the character-output subroutine COUT. For example, to display the character corresponding to $C8

```
LDA #$C8
JSR COUT
```

Primary Character Set Display

Subroutine COUT1 (the standard output link when enhanced video firmware is off) can display text in normal, inverse, or flashing format, but cannot display inverse or flashing *lowercase* text.

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

If the value of the character is greater than or equal to $A0, the value is logically ANDed with the value of the inverse flag (at location $32), then displayed.

If the inverse flag value is 255 (hexadecimal $FF), the character is displayed in normal format; if the value is 63 (hexadecimal $3F), the character is displayed in inverse format. If the value is 127 (hexadecimal $7F) the character is displayed in flashing format.

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

Character values from $80 through $9F are interpreted as control characters and are executed, if possible.

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

Alternate Character Set Display

MouseText: see Chapter 5.

Subroutine C3COUT1 (the standard output link when the enhanced video firmware is active) can display characters in normal or inverse format, and can display a set of icons called MouseText.

If the character is in the range $00 through $1F or $80 through $9F, it is interpreted as a control character and not displayed. Values $20 through $7F and $A0 through $FF are displayed.

If inverse flag (location $32) bit 7 is 1, the character value is left alone. If inverse flag bit 7 is 0, the character value is ANDed with $7F (turning off bit 7) to make it display as an inverse character.

If MouseText has not been turned on, then the values $40 through $5F are mapped to values $00 through $1F, so they display as the inverse uppercase set. If MouseText has been turned on, the values $40 through $5F are left unchanged, and they display as MouseText icons.

See section 5.2.2.

Warning
Use only $3F, $7F or $FF in location $32. Other values will cause unpredictable results.

3.4 Port I/O

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.

3.4.1 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 would have. Table 3-6 shows these equivalents, as well as listing the chapter where each port is described.

Section 3.1 describes under what conditions 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.

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	Reserved		
6	$C600	Disk drives	Built-in and external drives	6
7	$C700	No device	External drive startup (under ProDOS only)	6

Note: 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.

3.4.2 Firmware Protocol

Besides the standard link address, there is also a standard firmware protocol that provides a table of device identification and entry points to standard and optional firmware subroutines (Table 3-7).

Each table begins with identification bytes. 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 the address is $Cn, where *n* is the port number. Using these byte values, a program can construct its own jump table for subroutine calls.

On entry, all routines require that the X register contain $Cn (*n* is the port number), and that the Y register contain $n0.

On exit, all routines 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 of the Apple IIc ports except the disk port conform to this protocol.

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): $0 reserved $1 printer $2 hand control or other X-Y device $3 serial or parallel I/O card/port $4 modem $5 sound or speech device $6 clock $7 mass-storage device $8 80-column card/port $9 network or bus interface $A special purpose (none of the above) $B-F 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). (Enter with character to write in A register)
$Cn10	ss	$Cnss is the status routine entry address (PSTATUS). (Enter 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

3.4.3 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 sixteen memory locations of the form $C080 + #n0, where *n* is the port or slot number. These locations are set aside for data input and output. 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

3.4.4 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) 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 so there is not a strict correspondence between firmware for port n and the $Cn00 space, except as regards entry points.

3.4.5 Expansion ROM Space

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

3.4.6 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.

These addresses are unused bytes in the RAM memory 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.*

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

Port firmware use of these RAM locations and their assigned hardware addresses appear in the six chapters that follow this one.

3.5 Interrupts

Appendix E gives a full description of interrupt handling on the Apple IIc.

When the IRQ line on the 65C02 microprocessor is activated, the 65C02 transfers control through the vector in locations $FFFE-$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 ($3F0-$3F1). Otherwise, control is transferred through the IRQ vector ($3FE-$3FF).

Keyboard and Speaker

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

4.1 Keyboard Input

Table 4-1 describes the overall characteristics of the keyboard. Monitor keyboard support includes the three standard input routines described in Chapter 3.

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	**Description**
$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	[⌥] status on bit 7; 1 = pressed (also game input switch 0)
$C062	[⌘] status on bit 7; 1 = pressed

Game input switches: see Chapter 9.

Table 4-1—Continued. *Keyboard Input Characteristics*

	Monitor Firmware Routines		
	Location	**Name**	**Description**
GETLN, GETLN1, and RDKEY: see Chapter 3.	$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
	Use of Other Pages		
	Page 2	The standard character string input buffer (see GETLN description)	

4.1.1 Reading the Keyboard

The keyboard encoder and ROM generate all 128 ASCII codes, so all the special character codes in the ASCII character set are available from the keyboard. Machine-language programs obtain character codes from the keyboard by using RDKEY, which reads a byte from the keyboard-data location shown in Table 4-1.

Here is how reading the keyboard is done:

1. To see if a key has been pressed, test bit 7 at address $C000.
2. When that bit goes to a 1, the low-order seven bits are the character.
3. Clear the high bit at $C000 by reading or writing anything to address $C010.

$C010 has another function: 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.**

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

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

After the keyboard strobe has been cleared, it remains low until another key is pressed.

Table 4-2 shows the ASCII codes for the keys on the Apple IIc keyboard. If the strobe bit is set, add $80 to these values.

Codes are shown here in hexadecimal; to find the decimal equivalents, refer to section H.6.

Table 4-2. *Keys and ASCII Codes*

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

Table 4-2—Continued. Keys and ASCII Codes

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

There are several special-function keys that do not generate ASCII codes. For example, you cannot read CONTROL, SHIFT, and CAPS-LOCK directly, but pressing one of these keys alters the character codes produced by the other keys. Programs can also read the status of ⌂ and ◆ when checking keyboard input, and, if one or both of them is pressed, branch to a special routine, such as a help program.

Keystrokes can also generate interrupts: see Appendix E.

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 circuits. Pressing RESET with CONTROL depressed normally causes the system to stop whatever program it's running and restart itself.

This restarting process is called the **reset routine**, and is described in Chapter 2.

4.1.2 Monitor Firmware Support

Chapter 3 describes the three standard Monitor input routines serving the keyboard: GETLN, READKEY and KEYIN. This section discusses the three other Monitor routines available.

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

GETLNZ

GETLNZ (at address $FD67) is an alternate entry point for GETLN that 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 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, → (CONTROL-U) causes a character to be read from the screen location and returned.

4.2 Speaker Output

Electrical specifications of the speaker circuit appear in Chapter 11.

The Apple IIc has a speaker mounted toward the front of the bottom plate. The speaker is connected to a soft switch that toggles; it has two states, off and on, and it changes from one to the other each time it is accessed. 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	**Description**
$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

4.2.1 Using the Speaker

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

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.

See Chapter 3.

4.2.2 Monitor Firmware Support

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.

Video Display Output

The primary output device of the Apple IIc is the video display. You can use any ordinary video monitor, either color or monochrome, to display video information from the Apple IIc. An ordinary monitor is one that accepts composite video compatible with the standard set by the NTSC. If you use Apple IIc color graphics with, for example, a black-and-white monitor, the display will appear as black, white, and three shades of gray.

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

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

Note: 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 is a summary of the video output port's characteristics and a guide to other information in this chapter.

Table 5-1. *Guide to the Information in This Chapter*

Port Number	Output port 3
Commands	Figure 5-3
Initial Characteristics	Figure 5-2 **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 5-8
I/O Firmware Entry Points	See Table 5-9

5.1 Specifications

Table 5-2 summarizes the specifications for the video display, 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. It is available at two places: the RCA-type phono jack on the back of the Apple IIc, and the 15-pin D-type connector on the back panel. Use the RCA-type phono jack to connect a video monitor or the DB-15 connector for an external video modulator; use the 15-pin connector to attach video expansion hardware (section 11.9.5).

Either of the two text modes can display all 96 ASCII characters: uppercase and lowercase letters, numbers, and symbols.

5.2 Text Modes

The text characters displayed include the uppercase and lowercase letters, the ten digits, punctuation marks, and special characters. Each character is displayed in an area of the screen that is seven dots wide by eight dots high. The characters are formed by a dot matrix five dots wide (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) dots on a dark background. Characters can also be displayed as black dots on a white background; this is called inverse format.

5.2.1 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.

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

The alternate character set sacrifices the flashing format for a complete inverse format. 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.

MouseText: see section 5.2.2.

You select the character set by means of the alternate-text soft switch, described in section 5.6. 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.

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 (section 3.3.6).

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

Table 5-3. *The Display Character Sets*

Hex Values	Primary Character Set Character Type	Format	Alternate Character Set Character Type	Format
$00-$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
$20-$3F	Special characters	Inverse	Special characters	Inverse
$40-$5F	Uppercase letters	Flashing	MouseText (section 5.2.2)	
$60-$7F	Special characters	Flashing	Lowercase letters	Inverse
$80-$9F	Uppercase letters	Normal	Uppercase letters	Normal
$A0-$BF	Special characters	Normal	Special characters	Normal
$C0-$DF	Uppercase letters	Normal	Uppercase letters	Normal
$E0-$FF	Lowercase letters	Normal	Lowercase letters	Normal

5.2.2 MouseText

The character-generator ROM can display 32 graphics characters called **MouseText** in place of the inverse uppercase series from $40 through $5F. These graphics are especially convenient to use with a mouse, since 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: issue PR#3 or ESC 8 or ESC 4.
2. Set inverse mode: use the INVERSE command or put $3F in location $32, or print CONTROL-O.
3. Turn on the MouseText feature: `PRINT CHR$(27);` or pass $1B to COUT in the accumulator.
4. Print the uppercase letter (or other ASCII character in the range $40-$5F: @ [\ ] ^ or _) that corresponds to the MouseText character you want, using COUT1.

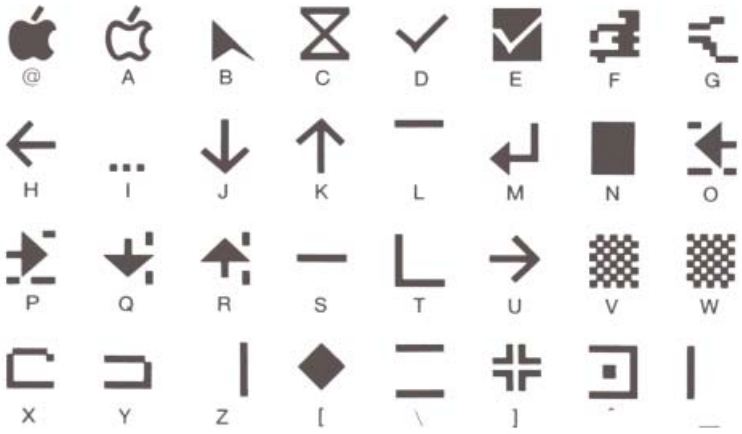
5. Turn off the MouseText feature: `PRINT CHR$(24);` or pass $18 to COUT1 in the accumulator.
6. Set normal mode: use the NORMAL command or put $FF in location $32, or print a CONTROL-N.

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
```

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

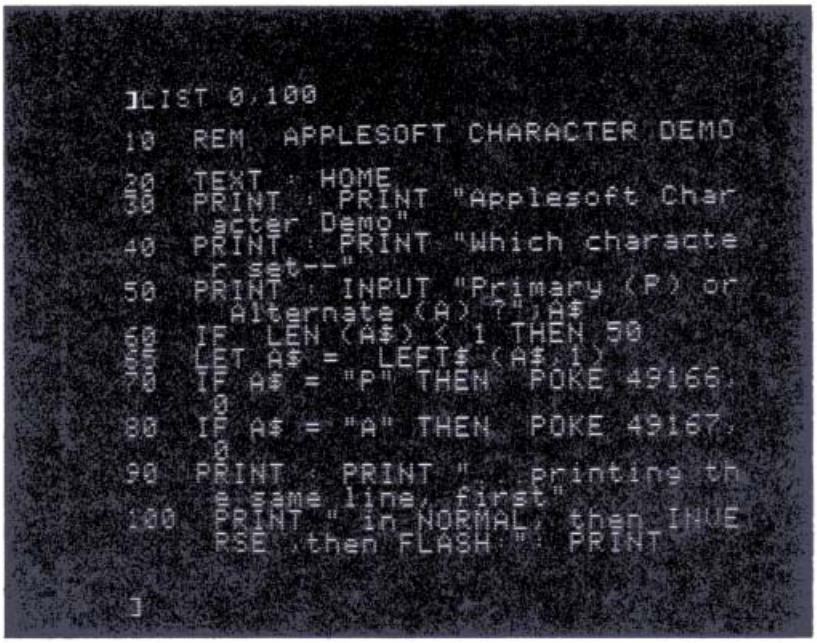
Figure 5-1. *MouseText Characters*



5.2.3 40-Column Versus 80-Column Text

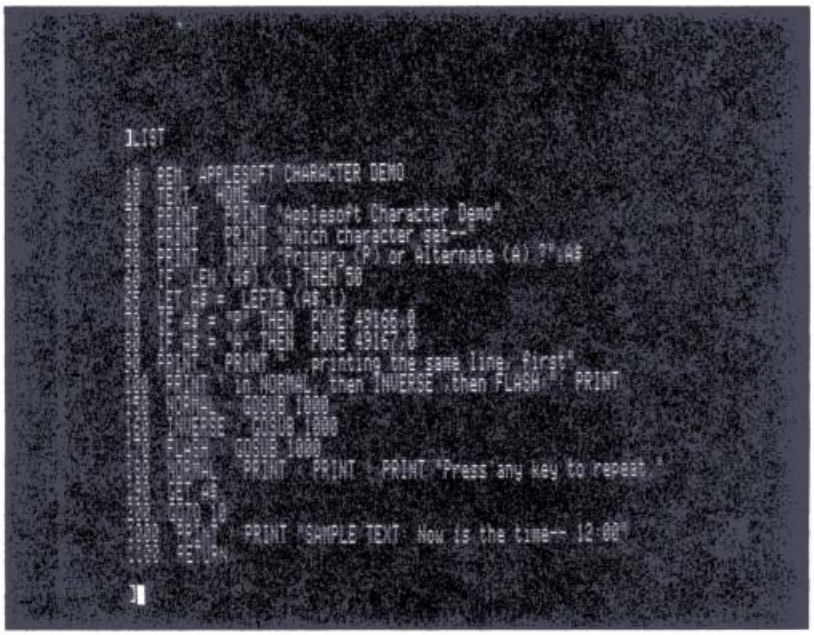
The Apple IIc has two modes of text display: 40-column and 80-column. The number of dots in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare the two displays in Figure 5-2. On an ordinary color or black-and-white television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

Figure 5-2. *40-Column and 80-Column Text Display (With Alternate Character 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$) < 1 THEN 50
70  LET A$ =  LEFT$ (A$,1)
80  IF A$ = "P" THEN  POKE 49166,
    0
90  IF A$ = "A" THEN  POKE 49167,
    0
100  PRINT : PRINT "   printing th
    e same line, first"
110  PRINT " in NORMAL, then INVE
    RSE ,then FLASH": PRINT
120  NORMAL : GOSUB 1000
130  INVERSE : GOSUB 1000
140  FLASH : GOSUB 1000
150  NORMAL : PRINT : PRINT : PRINT "Press any key to repeat."
160  GET A$
170  RUN 10
1000  REM  PRINT "SAMPLE TEXT: Now is the time-- 12:00"
1010  RETURN

]
```

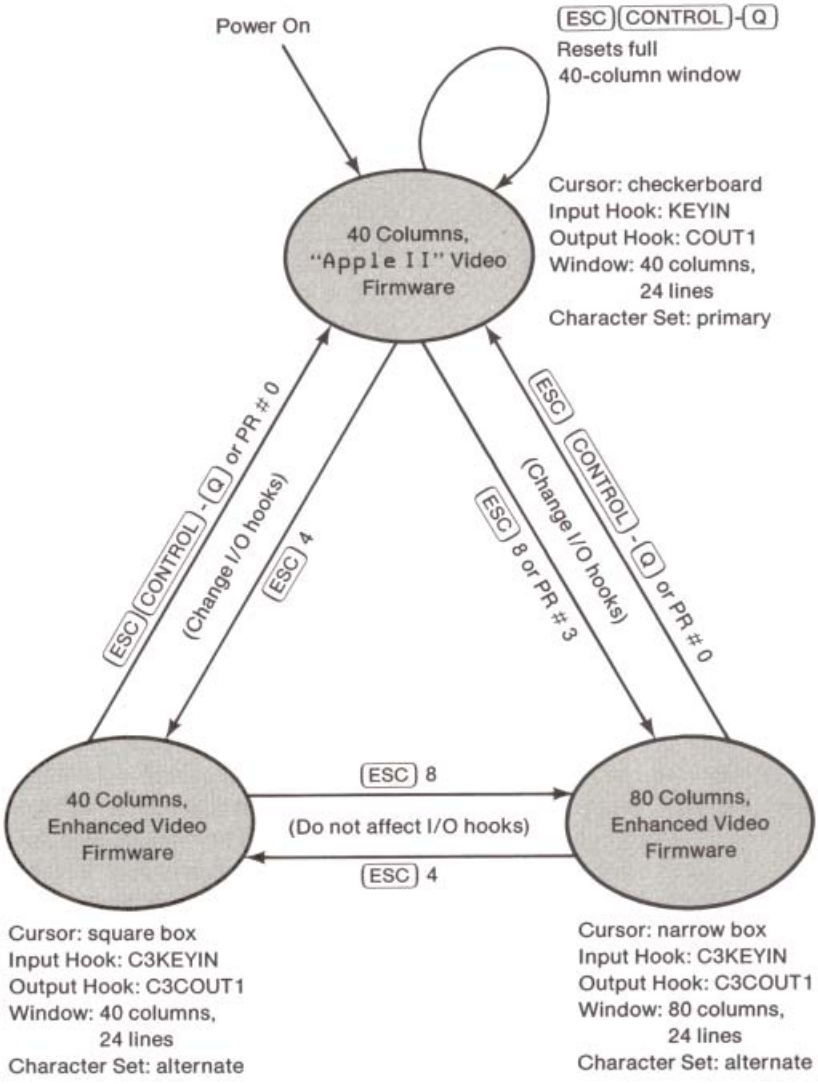


```
]LIST
10  REM  APPLESOFT CHARACTER DEMO
20  TEXT : HOME
30  PRINT : PRINT "Applesoft Character Demo"
40  PRINT : PRINT "Which character set--"
50  PRINT : INPUT "Primary (P) or Alternate (A) ?";A$
60  IF LEN (A$) < 1 THEN 50
70  LET A$ = LEFT$ (A$,1)
80  IF A$ = "P" THEN  POKE 49166,0
90  IF A$ = "A" THEN  POKE 49167,0
100  PRINT : PRINT "   printing the same line, first"
110  PRINT " in NORMAL, then INVERSE ,then FLASH": PRINT
120  NORMAL : GOSUB 1000
130  INVERSE : GOSUB 1000
140  FLASH : GOSUB 1000
150  NORMAL : PRINT : PRINT : PRINT "Press any key to repeat."
160  GET A$
170  RUN 10
1000  REM  PRINT "SAMPLE TEXT: Now is the time-- 12:00"
1010  RETURN

]
```

Figure 5-3 illustrates the methods of switching text display modes, and the characteristics of each.

Figure 5-3. *Text Mode Characteristics and Switching*



5.3 Graphics Modes

The Apple IIc can produce video graphics in any of three different modes. 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 in these arrays; this section describes the way the resulting graphics data are stored in the Apple IIc's memory.

5.3.1 Low-Resolution Graphics

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

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

Half a byte—four bits, or one nibble—is assigned to each graphics block. Each nibble can have a value from 0 to 15, and this value determines which one of sixteen colors appears on the screen. The colors and their corresponding nibble values are shown in Table 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.

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

Table 5-4. Low-Resolution Graphics Colors

Nibble Value Decimal	Hex	Color	Nibble Value Decimal	Hex	Color
0	$0	Black	8	$8	Brown
1	$1	Magenta	9	$9	Orange
2	$2	Dark Blue	10	$A	Gray 2
3	$3	Purple	11	$B	Pink
4	$4	Dark Green	12	$C	Light Green
5	$5	Gray 1	13	$D	Yellow
6	$6	Medium Blue	14	$E	Aquamarine
7	$7	Light Blue	15	$F	White

As explained in section 5.5, the text display and the low-resolution graphics display use the same area in memory. Most programs that generate text and graphics clear this part of memory when they change display modes, but it is possible to store data as text and display it as graphics, or vice versa. All you have to do is change the mode switch, described in section 5.6, 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.

5.3.2 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. Adjacent dots of the same color merge to form a larger colored area.

Data for the high-resolution graphics displays is stored in either of two 8192-byte areas in memory. These areas are called high-resolution Page 1 and Page 2; think of them as buffers where you can put data to be displayed. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes to display; this section describes the way the resulting graphics data are stored in the Apple 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 forty adjacent bytes in memory control a row of 280 (7 times 40) dots. The least significant bit of each byte is displayed as the leftmost dot in a row of seven, followed by the second-least significant bit, and so on, as shown in Figure 5-4. The eighth bit (the most significant) of each byte is not displayed; it selects one of two color sets, as described below.

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

On an NTSC color monitor or a color television set, a dot whose controlling bit is off (0) is black. If the bit is on, the dot will be white or a color, depending on its position, the dots on either side, and the setting of the high-order bit of the byte. Call the left-most column of dots column zero, and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control 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 other words, high-resolution graphics displayed on a color monitor or television set are made up of colored dots, according to the following rules:

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

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*

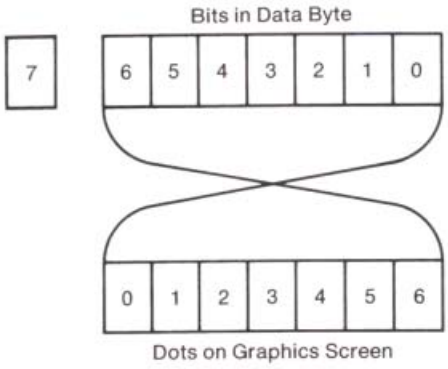
Bits 0-6	Bit 7 Off	Bit 7 On
Adjacent columns off	Black 1	Black 2
Even columns on	Purple	Blue
Odd columns on	Green	Orange
Adjacent columns on	White 1	White 2

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

The peculiar behavior of the high-resolution colors reflects 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. For information about the way NTSC color television works, see the magazine articles listed in the Bibliography.

Figure 5-4. High-Resolution Display Bits



5.3.3 Double-High-Resolution Graphics

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

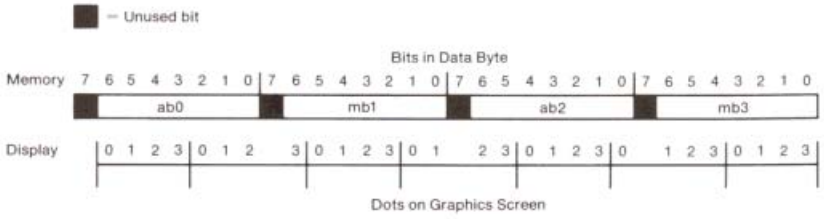
Unlike high-resolution color (section 5.3.2), double-high-resolution color has no restrictions on which colors can be adjacent. Color is determined by any four adjacent dots along a line. Think of a 4-dot-wide window moving across the screen: at any given time, the color displayed will correspond to the four-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 four).

To use the table, divide the column number by 4, 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*.

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

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	$455	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



5.4 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).

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

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

5.5 Display Pages

The Apple IIc generates its video displays using data stored in specific areas in memory. These areas, called **display pages**, serve as buffers where your programs can put data to be displayed. Each byte in a display buffer controls an object at a certain location on the display: a character, a colored block, or a group of adjacent dots.

The 40-column-text and low-resolution-graphics modes use two display pages of 1024 bytes each. These are called Text Page 1 and Text Page 2, and they are located at $400-$7FF and $800-$BFF in main memory. Normally, only Page 1 is used, but you can put text or graphics data into Page 2 and switch displays instantly. Either page can be displayed as 40-column text, low-resolution graphics, or mixed-mode (four rows of text at the bottom of a graphics display).

The 80-column text mode displays twice as much data as the 40-column mode—1920 bytes—but it cannot switch pages 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.

Note: The built-in video firmware always displays Page 1 text. You cannot write text to Page 2 unless you do it yourself.

The high-resolution graphics mode also has two display pages, but each page is 8192 bytes long. In the 40-column text and low-resolution graphics modes each byte controls a display area seven dots wide by eight dots high. In high-resolution graphics mode each byte controls an area seven dots wide by one dot high. Thus, a high-resolution display requires eight times as much data storage, as shown in Table 5-7.

The double-high-resolution graphics mode interleaves the two high-resolution 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. (1) This is not supported by firmware; for instructions on how to switch pages, refer to section 5.6. (2) See section 5.3.3.*

Display Mode	Display Page	Lowest Address		Highest Address	
40-column text, low-resolution graphics	1	$400	1024	$7FF	2047
	2 (1)	$800	2048	$BFF	3071
80-column text	1	$400	1024	$7FF	2047
	2 (1)	$800	2048	$BFF	3071
high-resolution graphics	1	$2000	8192	$3FFF	16383
	2	$4000	16384	$5FFF	24575
double-high-resolution graphics	1 (2)	$2000	8192	$3FFF	16383
	2 (2)	$4000	16384	$5FFF	16383

5.6 Display Mode Switching

Table 5-8 shows the reserved locations for the soft switches that control the different display modes. The left column of the table, labelled 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).

The IOUDIS switch must be on for 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. (1) The firmware normally leaves IOUDIS on. See also note 2. (2) Reading or writing any address in the range $C070-$C07F also triggers the paddle timer and resets VBLINT (Chapter 9).

Action	Hex	Name	Function	Note
W	$C00E	ALTCHAR	Off: display text using primary character set	
W	$C00F	ALTCHAR	On: display text using alternate character set	
R7	$C01E	RDALTCHAR	Read ALTCHAR switch (1 = on)	
W	$C00C	80COL	Off: display 40 columns	
W	$C00D	80COL	On: display 80 columns	
R7	$C01F	RD80COL	Read 80COL switch (1 = on)	
W	$C000	80STORE	Off: cause PAGE2 on to select auxiliary RAM	
W	$C001	80STORE	On: allow PAGE2 to switch main RAM areas	
R7	$C018	RD80STORE	Read 80STORE switch (1 = on)	
R/W	$C054	PAGE2	Off: select page 1	
R/W	$C055	PAGE2	On: select page 1X (80STORE on) or 2	
R7	$C01C	RDPAGE2	Read PAGE2 switch (1 = on)	
R/W	$C050	TEXT	Off: display graphics or (if MIXED on) mixed	
R/W	$C051	TEXT	On: display text	
R7	$C01A	RDTEXT	Read TEXT switch (1 = on)	
R/W	$C053	MIXED	Off: display only text or only graphics	
R/W	$C054	MIXED	On: (if TEXT off) display text and graphics	
R7	$C01B	RDMIXED	Read MIXED switch (1 = on)	
R/W	$C057	HIRES	Off: (if TEXT off) display low-resolution graphics	
R/W	$C058	HIRES	On: (if TEXT off) display high-resolution or (if DHIRES on) double-high-resolution graphics	
R7	$C01D	RDHIRES	Read HIRES switch (1 = on)	

Table 5-8—Continued. *Display Soft Switches*

Action	Hex	Name	Function	Note
W	$C07E	IOUDIS	On: disable IOU access for addresses $C058 to $C05F; enable access to DHIRES switch	(1)
W	$C07F	IOUDIS	Off: enable IOU access for addresses $C058 to $C05F; disable access to DHIRES switch	(1)
R7	$C07E	RDIOUDIS	Read IOUDIS switch (1 = off)	(2)
R/W	$C05E	DHIRES	On: (if IOUDIS on) turn on double-high-resolution	
R/W	$C05F	DHIRES	Off: (if IOUDIS on) turn off double-high-resolution	
R7	$C07F	RDDHIRES	Read DHIRES switch (1 = on)	(2)

Table 5-9. *Display Modes Supported by Firmware (Including Applesoft). (1) 80STORE is set by the firmware when 80COL is turned on.*

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	(1)		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	(1)	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	(1)	off	off	on	on	off

Table 5-10. *Other Display Modes. (1) 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.*

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	(1)	off	off	off	off	on
dbl-low	graphics	2	on	off	on	off	off	off	on
80/dbl-low	mixed	1	on	(1)	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	(1)	off	off	off	on	on
dbl-high	graphics	2	on	off	on	off	off	on	on
80/dbl-high	mixed	1	on	(1)	off	off	on	on	on
80/dbl-high	mixed	2	on	off	on	off	on	on	on

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

STA	$C00D	Turn on 80COL; firmware then turns on 80STORE
LDA	$C054	Turn off PAGE2; you could also have done a STA
STA	$C050	Turn off TEXT; that is, turn on graphics mode
STA	$C053	Turn on MIXED; it works now that TEXT is off
STA	$C057	Turn on HIRES; it works now that TEXT is off
STA	$C07E	Make sure IOUDIS is on so you can access DHIRES
LDA	$C05E	Turn on DHIRES; it works now that

5.7 Display Page Maps

You should never have to store directly into display memory. Most high-level languages enable you to write statements that control the text and graphics displays. Similarly, if you are programming in assembly language, you should be able to 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.

The display memory maps are shown in Figures 5-5 through 5-9. All the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes ($80 hexadecimal). For a full description of the way the Apple IIc handles its display memory, refer to section 11.9.2.

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

The display maps show addresses only for each Page 1. To obtain addresses for text or low-resolution graphics Page 2, add 1024 ($400); to obtain addresses for high-resolution Page 2, add 8192 ($2000).

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

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*

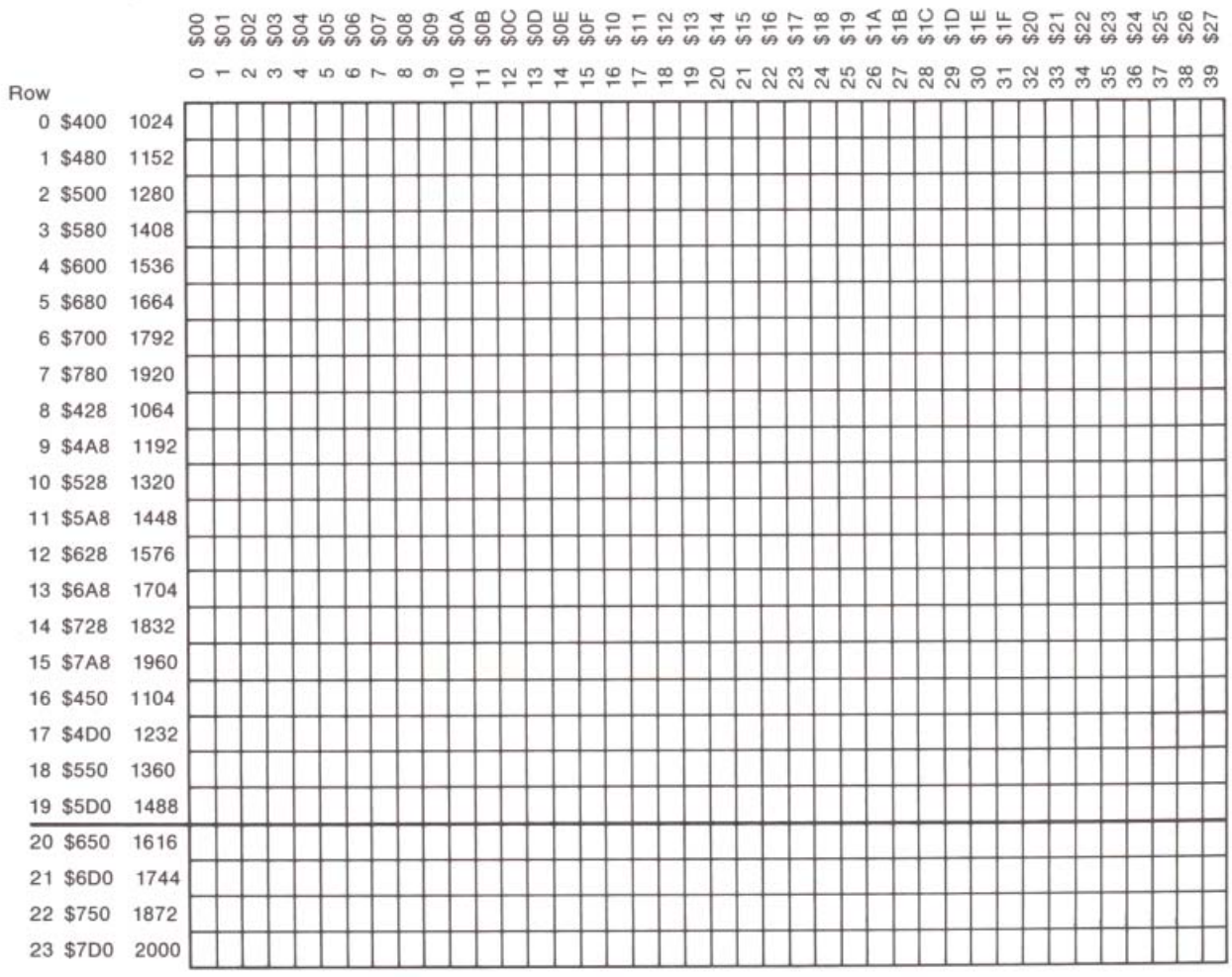


Figure 5-6. Map of 80-Column Text Display

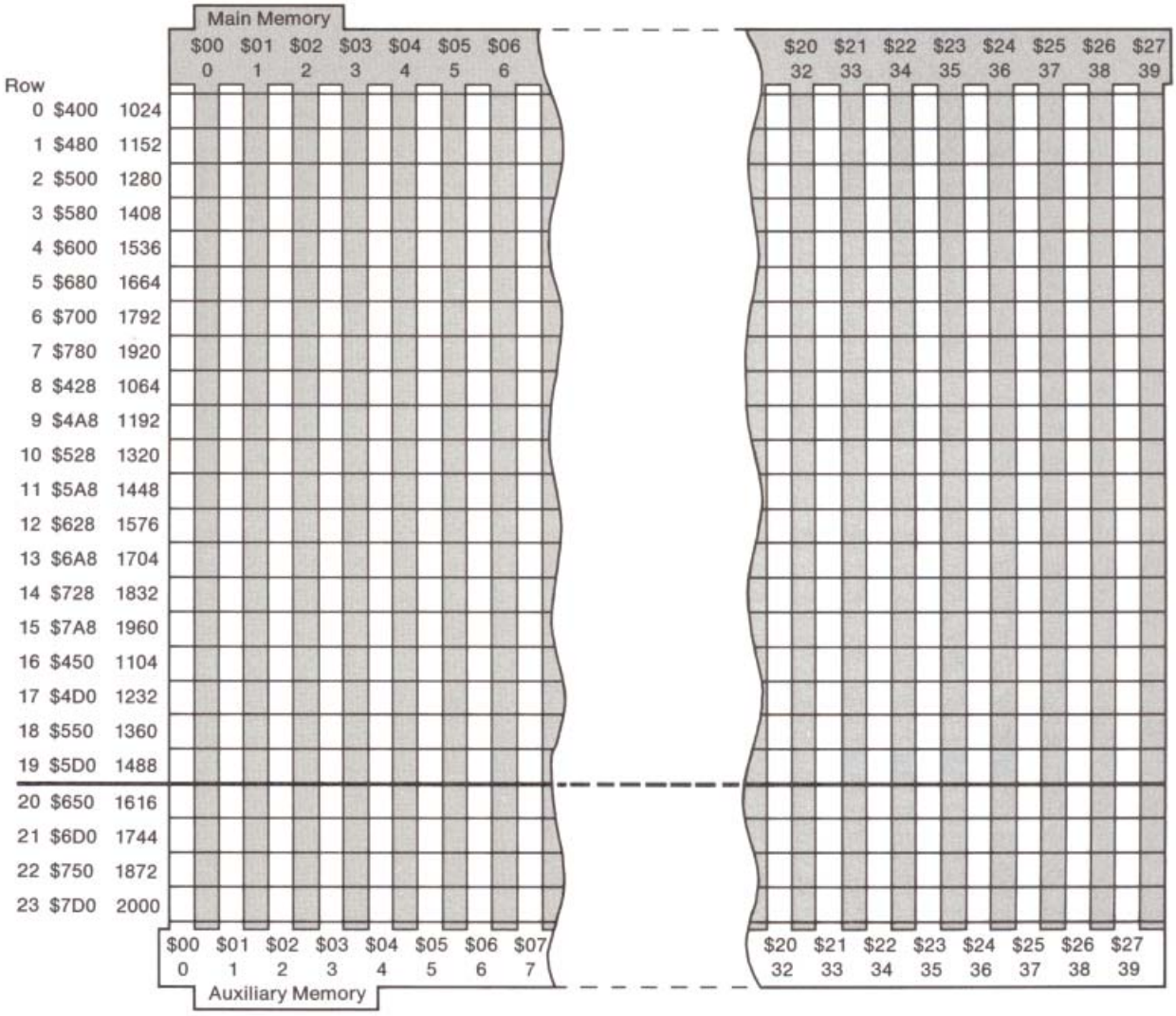


Figure 5-7. Map of Low-Resolution Graphics Display

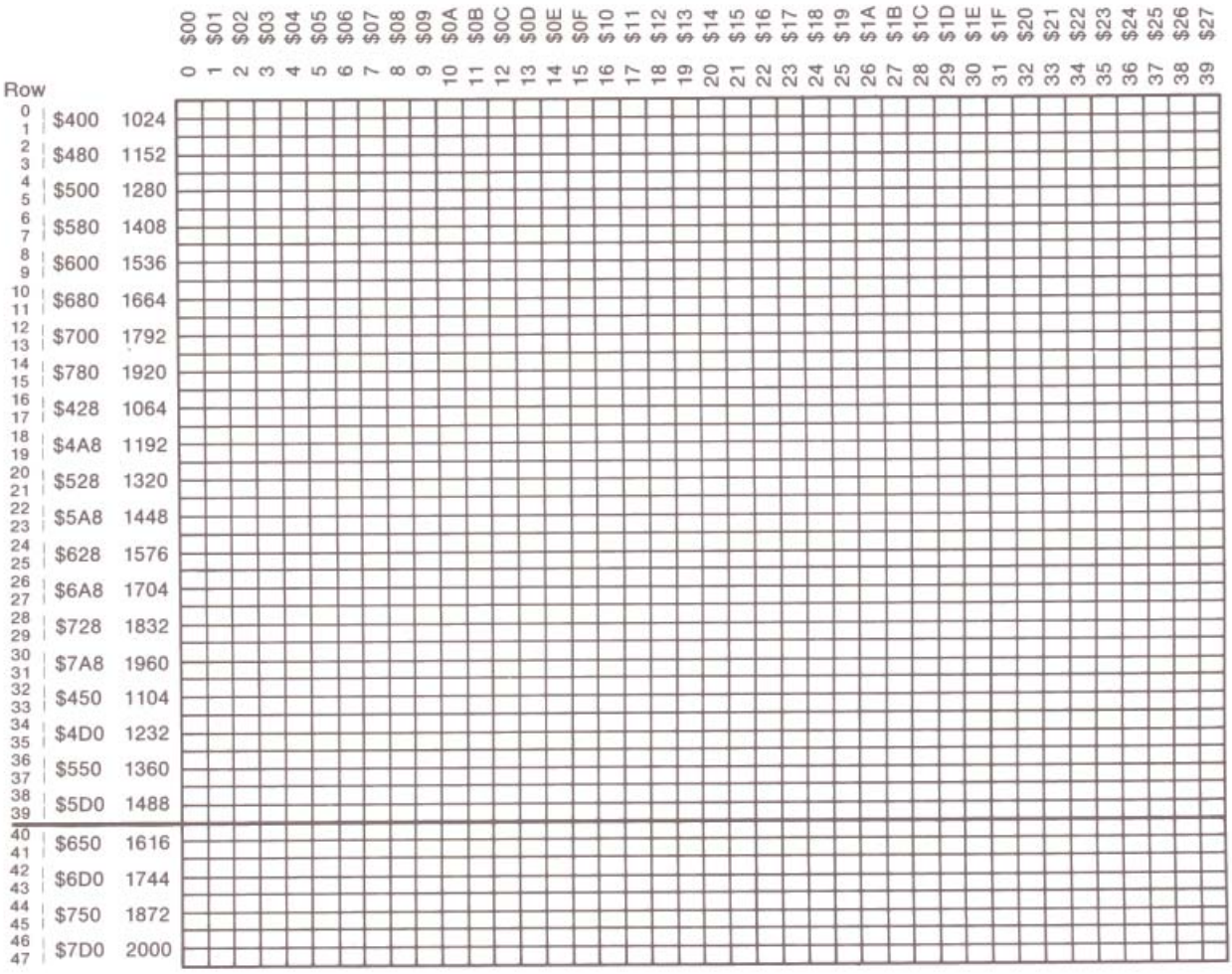


Figure 5-8. *Map of High-Resolution Graphics Display*

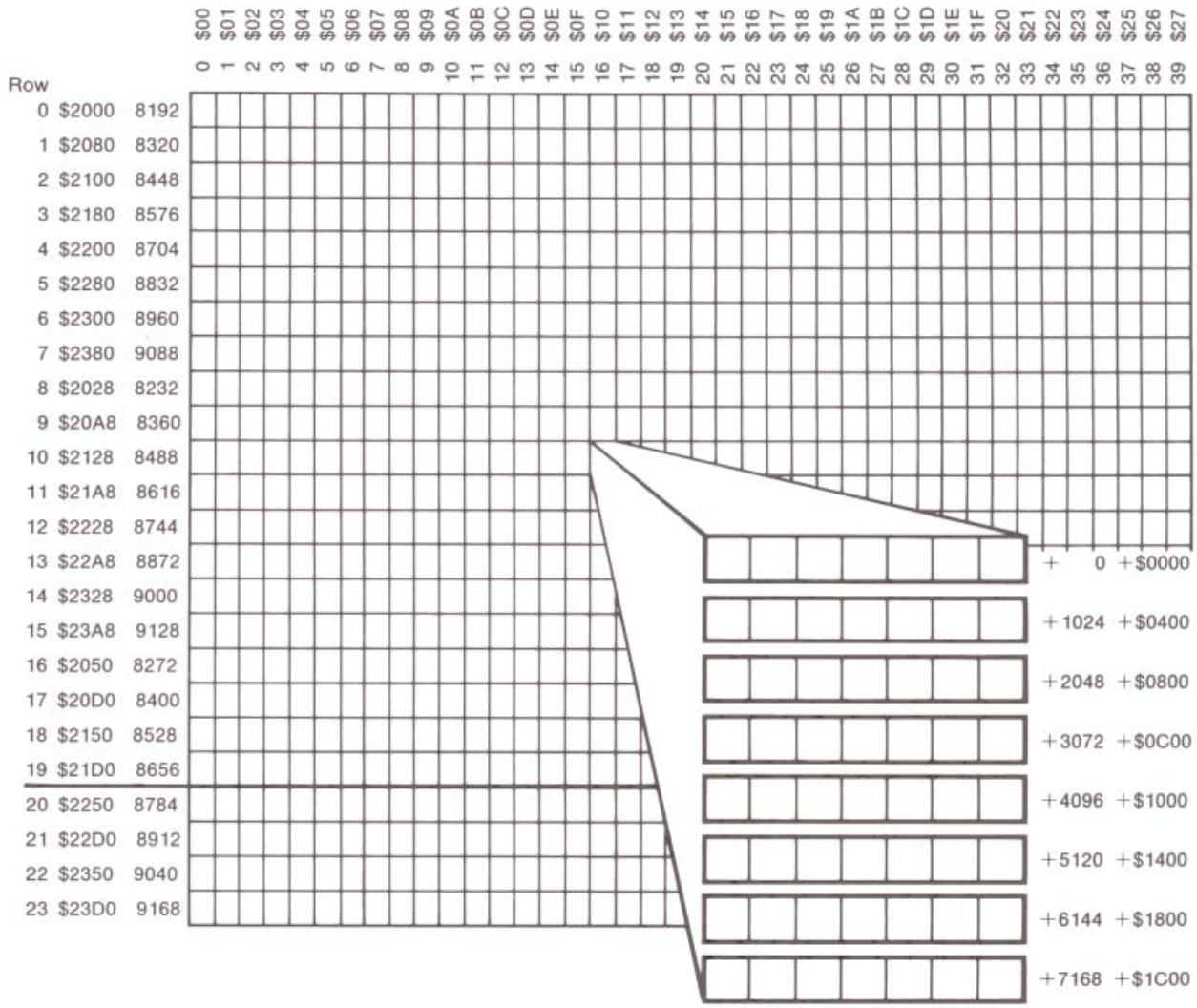
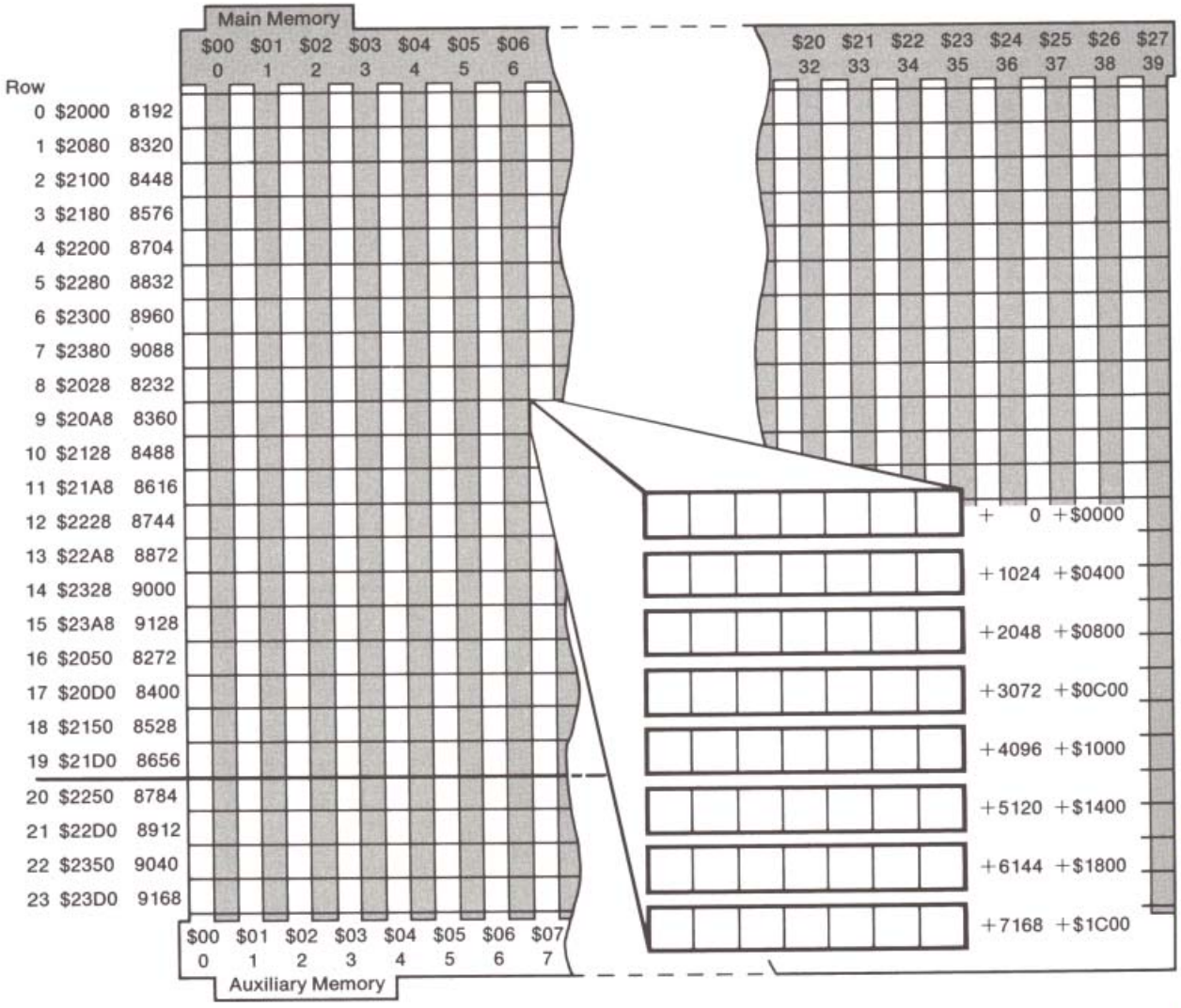


Figure 5-9. Map of Double-High-Resolution Graphics Display



5.8 Monitor Firmware Support

Table 5-9 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*

Location	Name	Description
$FC9C	CLREOL	Clears to end of line from current cursor position
$FC9E	CLEOLZ	Clears to end of line using contents of Y register as cursor position
$FC42	CLREOP	Clears to bottom of window
$F832	CLRSCR	Clears the low-resolution screen
$F836	CLRTOP	Clears top 40 lines of low-resolution screen
$FDED	COUT	Calls output routine whose address is stored in CSW (normally COUT1, Chapter 3)
$FDF0	COUT1	Displays a character on the screen (Chapter 3)
$FD8E	CROUT	Generates a carriage return character
$FD8B	CROUT1	Clears to end of line, then generates a carriage return character
$F819	HLINE	Draws a horizontal line of blocks
$FC58	HOME	Clears the window and puts cursor in upper-left corner of window

Table 5-11—Continued. Monitor Firmware Routines

Location	Name	Description
$F800	PLOT	Plots a single low-resolution block on the screen
$F94A	PRBL2	Sends 1 to 256 blank spaces to the output device whose address is in CSW
$FDDA	PRBYTE	Prints a hexadecimal byte
$FF2D	PRERR	Sends ERR and CONTROL-G to the output device whose output routine address is in CSW
$FDE3	PRHEX	Prints 4 bits as a hexadecimal number
$F941	PRTAX	Prints contents of A and X in hexadecimal
$F871	SCRN	Reads color value of a low-resolution block on the screen
$F864	SETCOL	Sets the color for plotting in low-resolution
$FC24	VTABZ	Sets cursor vertical position (Setting CV at location $25 does not change vertical positon until a carriage return.)
$F828	VLINE	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 (section 3.3.1). The character to be sent to the output device should be in the accumulator. COUT calls the subroutine whose address is stored in CSW (locations $36 and $37), which is usually the standard character output COUT1.

COUT1

COUT1 (section 3.3.2) 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, 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.

PRTAX

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

SCRN

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 acumulator. 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.

5.9 I/O Firmware Support

Apple IIc video firmware conforms to the I/O firmware protocol (section 3.4.2). 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 zero 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. See Table 5-13.
- Carries out control functions as shown in Table 5-10.

Table 5-13. Pascal Video Control Functions

CONTROL-	Hex	Function performed
E or e	$05	Turns cursor on (enables cursor display)
F or f	$06	Turns cursor off (disables cursor display)
G or g	$07	Sounds bell (beeps)
H or h	$08	Moves cursor left one column. If cursor was at beginning of line, moves it to end of previous line
J or j	$0A	Moves cursor down one row; scrolls if needed
K or k	$0B	Clears to end of screen
L or l	$0C	Clears screen; moves cursor to upper-left of screen
M or m	$0D	Moves cursor to column 0
N or n	$0E	Displays subsequent characters in normal video. (Characters already on display are unaffected.)
O or o	$0F	Displays subsequent characters in inverse video. (Characters already on display are unaffected.)
V or v	$16	Scrolls screen up one line; clears bottom line
W or w	$17	Scrolls screen down one line; clears top line
Y or y	$19	Moves cursor to upper-left (home) position on screen
Z or z	$1A	Clears entire line that cursor is on
| or \	$1C	Moves cursor right one column; if at end of line, does CONTROL-M
] or]	$1D	Clears to end of the line the cursor is on, including current cursor position; does not move cursor
^ or 6	$1E	GOTOxy: initiates a GOTOxy sequence; interprets the next two characters as x+32 and y+32, respectively
_	$1F	If not at top of screen, moves cursor up one line

When PWRITE has completed this, it

- turns the cursor back on (if it was not intentionally turned off).
- puts a zero in the X register (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).

Disk Input and Output

The Apple IIc supports both its built-in disk drive and an optional external drive; both drives use single-sided, 35-track, 16-sector format. The disk I/O port characteristics are summarized in Table 6-1.

The external disk drive connector is described in section 11.10.

The firmware resides in the $C600 address space. It supports the built-in drive as if it were slot 6 drive 1, and the external drive as if it were slot 6 drive 2. If disk startup is unsuccessful, the firmware shuts off the disk drive motor and displays the message, `Check Disk Drive` on the display screen.

Table 6-1. *Disk I/O Characteristics*

Port Number	I/O Port 6 Drive 1 (built-in drive) I/O Port 6 Drive 2 (external 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.
Hardware Location	**Description**
$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.

6.1 Startup

A power-on startup, an (ó)-(CONTROL)-(RESET) startup, or a (CONTROL)-(RESET) startup that does not find a valid reset vector results in a cold start. The cold-start routine first sets a number of soft switches (see Chapter 2) and then passes control to the program entry point at $C600. This code turns on the internal drive motor, recalibrates the read/write head at track zero, then reads sector zero from that track. The sector contents are loaded and decoded starting at address $800; then program control passes to $801. This loaded program varies depending on the operating system or application program on the disk.

To restart the system, issue a `PR#6` command from BASIC, (6) (CONTROL)-(P) from Monitor command mode, or `JMP $C600` from a machine-language program.

6.2 External Drive Startup

The ProDOS operating system (but not the DOS or Pascal operating systems) supports startup using the external disk drive. This ProDOS feature makes it possible to start the Apple IIc with a diagnostic program in the event that the built-in drive does not work.

To restart using the external drive, insert a ProDOS disk in the external drive, then invoke the Monitor (CALL -151) and issue (7) (CONTROL)-(P).

Remember: External drive startup works with ProDOS-based programs, but not with Pascal 1.1 or 1.0, or with DOS.

Serial I/O Port 1

If you need to change port characteristics from a program, read section 7.6 for the methods to use.

Serial port 1 is the first 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) using the System Utilities Disk.

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

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.

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.

The serial port back panel connectors are described in section 11.11.

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 Table 7-2
Initial Characteristics	Table 7-3
Hardware Page Locations	Table 7-4
Monitor Firmware Routines	None
I/O Firmware Entry Points	Table 7-5
Use of Screen Holes	Table 7-6
Use of Other Pages	None

7.1 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.

Refer to Table 7-5 for the standard firmware entry points that Pascal 1.1 and 1.2 use.

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

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. 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.

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

Table 7-2. Printer Port Commands

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

nn	Rate	nn	Rate
1	50	9	1800
2	75	10	2400
3	110 (109.92)	11	3600
4	135 (134.58)	12	4800
5	150	13	7200
6	300	14	9600
7	600	15	19200
8	1200		

nD — Set data format to values corresponding to n:

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

I	Echo printer output on the screen.
K	Disable automatic line feed after carriage return.
L	Generate line feed after carriage return.
nnnN	Change line width to nnn (from 1 through 255; nnn is optional); do 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 $579 or use System Utilities Disk.

Table 7-2—Continued. *Printer Port Commands*

Command	Description
nP	Set parity corresponding to n:

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

R	Reset port 1 (Table 7-3) and exit from serial port 1 firmware.
S	Send a 233 millisecond BREAK character (used with some printers to synchronize with serial ports).
Z	Zap (ignore) further command characters (until CONTROL-RESET or PR#1). Do not format output or insert carriage returns into output stream.

The command character starts off as CONTROL-I for the printer port. 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-I to the printer without firmware intervention. For example, to change the command character from CONTROL-I to CONTROL-V, simply press CONTROL-I CONTROL-V. (CONTROL-V and CONTROL-W are the recommended substitute control characters.) To change the command character back again, press CONTROL-V CONTROL-I.

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.

Echo output to the display screen:

CONTROL-I I

Set line width 72, disable line feed, and echo:

CONTROL-I K CONTROL-I 7 2 N

Change control character to CONTROL-V:

CONTROL-I CONTROL-V RETURN

An example of how you can send CONTROL-I as part of a character stream:

CONTROL-V (command) RETURN

7.2 Characteristics at Startup

After power-up, the printer firmware sets the configuration given in Table 7-3. These values are stored in the auxiliary-memory screen holes (Table 7-6).

Table 7-3. *Initial Characteristics of Printer Port*

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).

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. Section 7.6 describes how port characteristics change as a result of various activities.

Note: You can type more than one command on a line, but each must be preceded by the command character.

7.3 Hardware Page Locations

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 7-4 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 section 11.11.

Table 7-4. *Serial 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

7.4 I/O Firmware Support

Table 7-5 lists the locations and values of the I/O firmware protocol table. This standardized protocol is available for use by any application program. Section 3.4.2 describes how to use this protocol.

Table 7-5. *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-0	No optional routines

7.5 Screen Hole Locations

The ACIA register bits are defined in Chapter 11.

Table 7-6 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 7-6. *Serial Port 1 Screen Hole Locations*

Auxiliary Memory Screen Holes (firmware loads at power-up reset):

Location	Description
$478	$9E (ACIA control reg: 8 data + 2 stop bits, 9600 baud)
$479	$0B (ACIA command reg: no parity)
$47A	$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

$47B	$50 (printer width: 80 columns)

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

Main Memory Screen Holes:

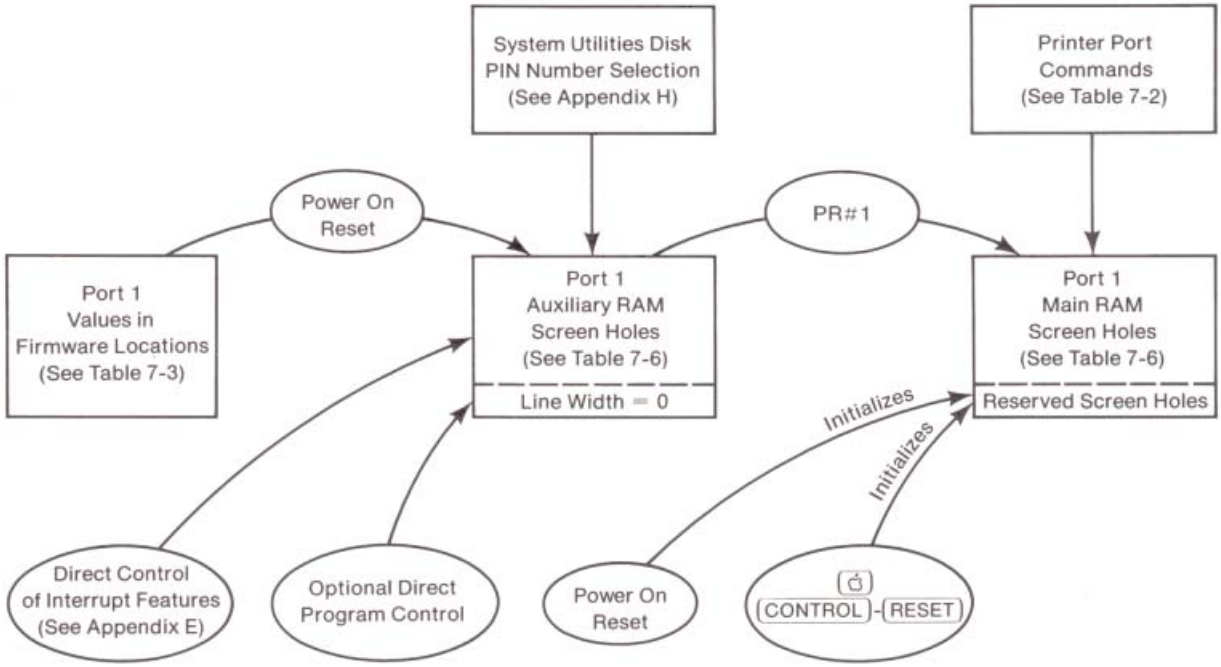
Location	Description
$479	Reserved
$4F9	Reserved
$579	Printer width (1 - 255; 0 = disable formatting)
$5F9	Temporary storage location
$679	Bit 7 = 1 while the firmware is parsing a command string.
$6F9	Current command character (initially CONTROL-I)
$779	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return.
$7F9	Current printer column

7.6 Changing Port Characteristics

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

- When the power is first turned on, the Monitor reset firmware moves the predefined set of port characteristics listed in Table 7-3 from ROM into the auxiliary memory screen holes listed in Table 7-6.
- If you specify new characteristics using the System Utilities Disk, the SUD 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 (ó)-(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.
- 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 $579.
- 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. Port Characteristics



7.6.1 Data Format and Baud Rate

Serial data transfer consists of a string of ones and zeros sent down a wire at a prearranged rate of speed, called the **baud rate**. With most equipment, baud simply equals the number of bits per second.

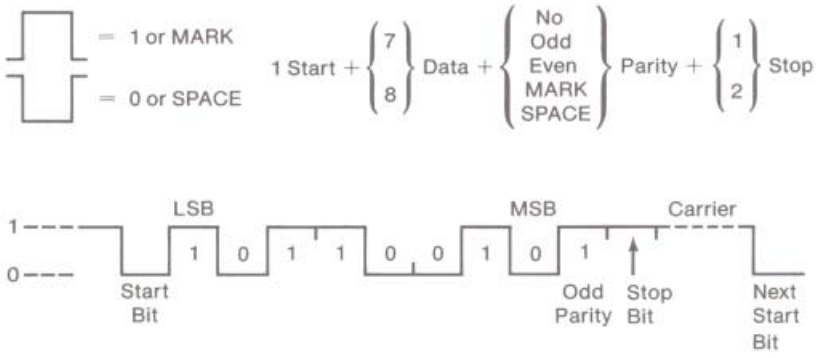
Before transfer begins, both sender and receiver look for a continuous value of 1: this is called the **carrier** (Figure 7-2). When the value goes to zero, the receiver presumes it is a **start bit**—that is, the bit that designates the beginning of a character of data. If it lasts longer than a bit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

If the first zero proves to be a bit, it is interpreted as the start bit. Next come the 7 or 8 data bits (6 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** (with a value of 1) 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 ones among the data bits, and sends the appropriate parity bit to make the total number of ones odd. With even parity, the sender adds the appropriate parity bit to make the total number of one-bits even. MARK parity is always a 1 bit; SPACE parity is always a zero. The receiver can then check that the parity bit is correct.

If the baud rate is 300, and the data format is 1 start bit plus 7 data bits plus 1 parity bit plus 1 stop bit, then the actual transfer rate is about 30 characters per second.

Figure 7-2. Data Format



ASCII letter M = $4D; sent as 8 data, odd parity, 1 stop bit.

7.6.2 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). ↓ 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 will keep printing 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 will result.

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.

7.6.3 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. 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.

7.6.4 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.

Serial I/O Port 2

Serial port 2 is the second 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. If you need to change port characteristics from a program, read section 8.6.

Warning

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

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.

Table 8-1 summarizes the characteristics of this port and is a guide to the other information in this chapter.

The serial port connectors are described in section 11.11.

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) (This command works only if there is no operating system in RAM.) All other commands Table 8-2
Initial Characteristics	Table 8-3
Hardware Page Locations	Table 8-4
Monitor Firmware Routines	None
I/O Firmware Entry Points	Table 8-5
Use of Screen Holes	Table 8-6
Use of Other Pages	In terminal mode, firmware uses auxiliary memory locations $800-$87F to store keyboard input, and $880-$8FF as a serial input buffer.

8.1 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.

Note: 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).

Refer to Table 8-5 for the standard firmware entry points that Pascal 1.1 and 1.2 use.

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

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. 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) after commands.

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

Table 8-2. Modem Port Commands

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

nn	Rate	nn	Rate
1	50	9	1800
2	75	10	2400
3	110 (109.92)	11	3600
4	135 (134.58)	12	4800
5	150	13	7200
6	300	14	9600
7	600	15	19200
8	1200		

nD Set data format to values corresponding to n:

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

Table 8-2—Continued. *Modem Port Commands*

Command	Description
I	Echo output on the screen.
K	Disable automatic line feed after carriage return.
L	Generate line feed after carriage return.
nnnN	Set line width to nnn (from 1 through 255); do not echo output on the screen. **Note:** 0N does not turn off automatic generation of carriage return; to do so, use Z command, put 0 directly in location $57A, or use the System Utilities Disk.
nP	Set parity corresponding to n:

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

Q	Quit terminal mode.
R	Reset port 2 (Table 8-3) and exit from serial port 2 firmware.
S	Send a 233 millisecond BREAK character.
T	Enter terminal mode. Use this command after IN#2 only. Also, if you follow this command by PR#2, the Apple IIc will echo input to output. (If the other device does so too, the first character entered will loop endlessly, locking up the system. Use CONTROL-RESET to get out.)
Z	Zap (ignore) further command characters until CONTROL-RESET. Do 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.
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.

The command character starts off as CONTROL-A for the communication port. 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, simply press (CONTROL)-(A) (CONTROL)-(V). (CONTROL-V and CONTROL-W are the recommended substitute control characters.) To change the command character back again, press (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.

Enable echo to the screen:

(CONTROL)-(A) (I)

Send a BREAK character to a remote device:

(CONTROL)-(A) (B)

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)

8.2 Characteristics at Startup

After power-up, the firmware sets the configuration given in Table 8-3. These values are stored in the auxiliary-memory screen holes (Table 8-6).

Table 8-3. *Initial Characteristics of Communication Port*

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.

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. Section 8.6 describes how port characteristics change as a result of various activities.

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. Refer to section 8.6 for a complete description of these processes.

8.3 Hardware Page Locations

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 1.

Table 8-4 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 section 11.11.

Table 8-4. *Serial 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

8.4 I/O Firmware Support

Table 8-5 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. Section 3.4.2 describes how to use this protocol.

Table 8-5. *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	$C1ii is entry point of initialization routine (PINIT).
$C20E	$rr	$C1rr is entry point of read routine (PREAD).
$C20F	$ww	$C1ww is entry point of write routine (PWRITE).
$C210	$ss	$C1ss is entry point of the status routine (PSTATUS).
$C211	non-0	No optional routines

8.5 Screen Hole Locations

The ACIA register bits are defined in Chapter 11.

Table 8-6 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 8-6. *Serial Port 2 Screen Hole Locations*

Location	Description
Auxiliary Memory Screen Holes (firmware loads values at power-up):	
$47C	$16 (ACIA control reg: 8 data + 1 stop bit, 300 baud)
$47D	$0B (ACIA command reg: no parity)
$47E	$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

$47F	$00 (line length: do not add any CR to output stream)

Bit	Interpretation
7-0	Line length (0 = do not insert CR)

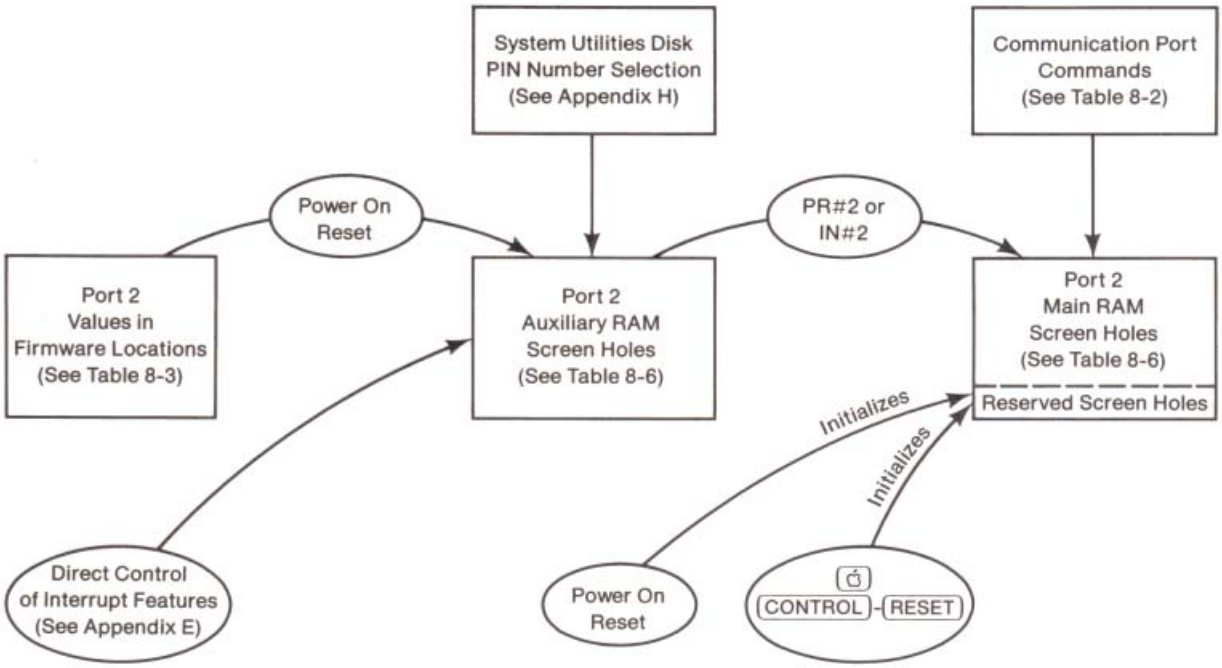
Main Memory Screen Holes:	
$47A	Reserved
$4FA	Reserved
$57A	Line length (1 - 255; 0 = disable formatting)
$5FA	Temporary storage location
$67A	Bit 7 = 1 if and only if the firmware is currently parsing a command string.
$6FA	Current command character (initially CONTROL-I)
$77A	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return.
$7FA	Current column

8.6 Changing Port Characteristics

Figure 8-1 is a diagram of where the port characteristics are stored and moved under different circumstances. As you can see 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 8-6.
- 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 (ó)-(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 $57A.
- 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. Port 2 Characteristics



8.6.1 Data Format and Baud Rate

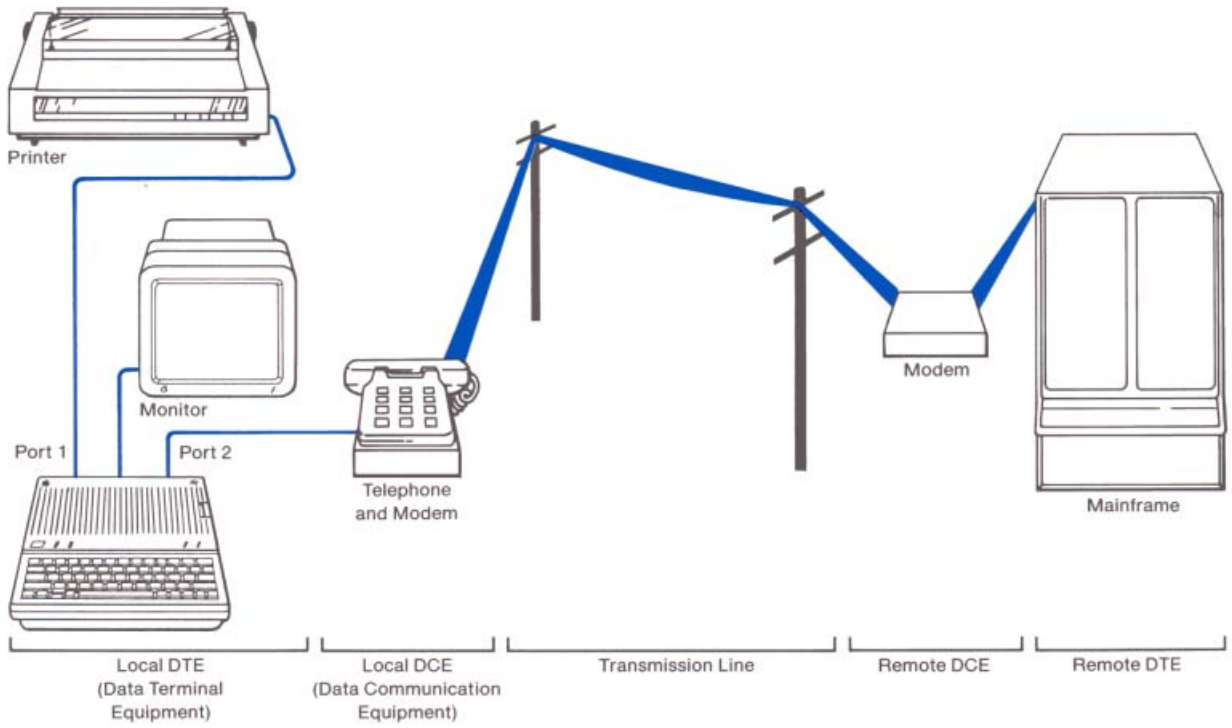
Section 7.6.1 describes data format and baud rate, and explains how they apply to printers. Refer to that section or to the Glossary for the definition of terms.

A noteworthy characteristic of data communication is its strangeness: sometimes the oddest changes make a given communication arrangment 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 computer or terminal on the other end.

Figure 8-2. *Devices in a Typical Communication Setup*



As you can imagine, some method is required for success. 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.

8.6.2 Carriage Return and Line Feed

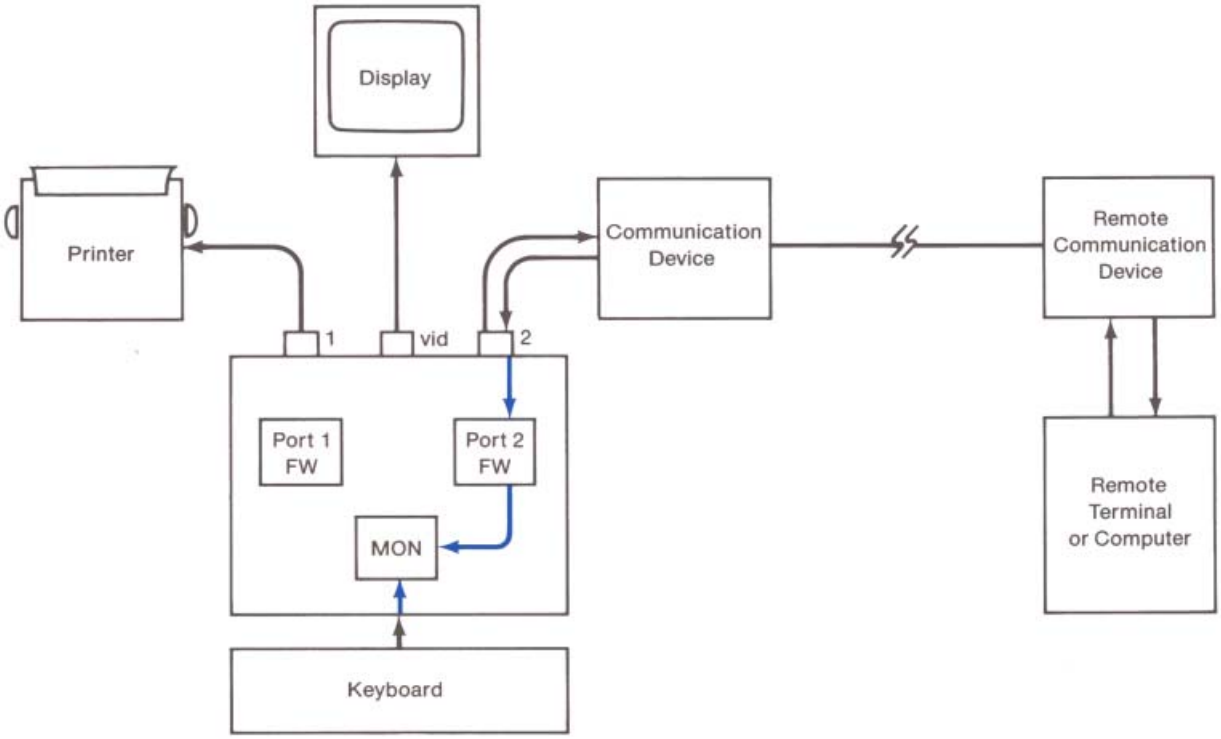
If you are communicating with a computer or terminal, carriage return and line feed may or may not be involved. Start off without generating them, and turn on automatic generation only as needed. They are described as used with printers in section 7.6.2.

8.6.3 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. This section and the following subsections tell you how to use them.

It is best to read all of this material as a unit: questions that arise while you read one description may be answered elsewhere.

Figure 8-3. Effect of IN#2



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 2 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, typing (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.

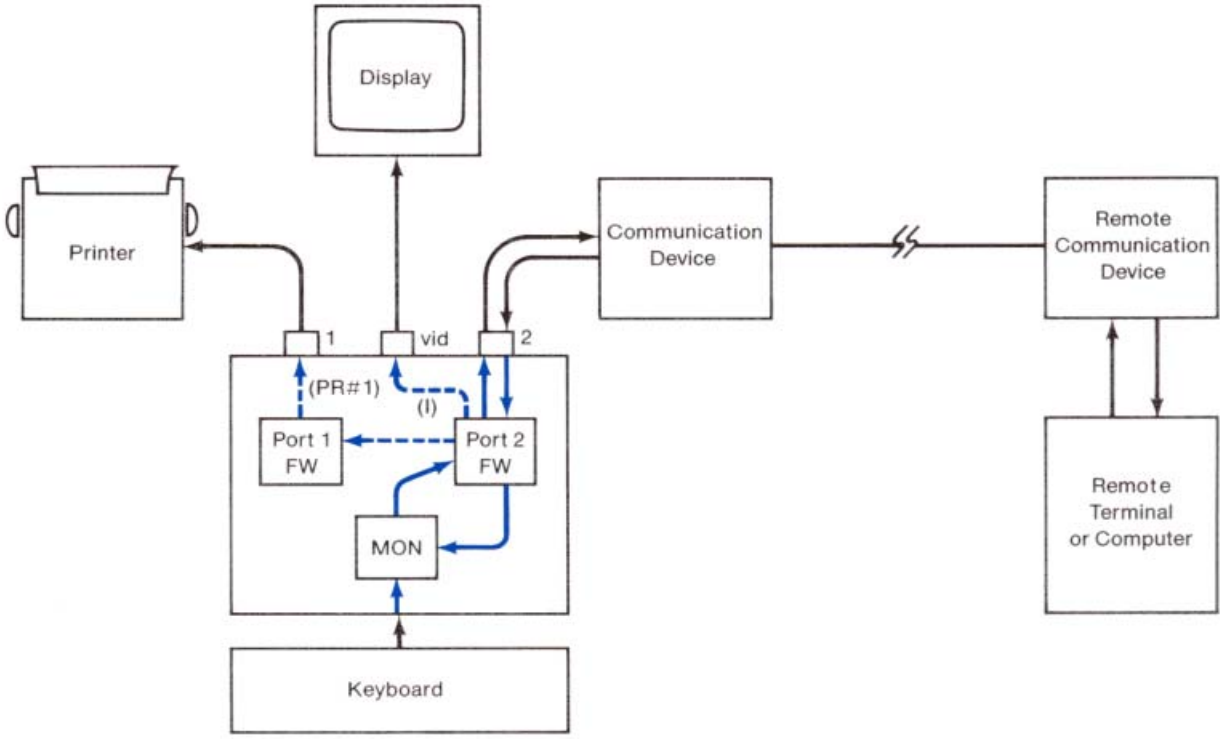
In the discussion that follows, **local** refers to your Apple IIc. **Remote** refers to some other device, usually in a distant location and at the other end of a communication link. The remote device can be any ASCII-generating unit: a terminal or a computer.

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.

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.

Figure 8-4. *Effect of IN#2 and T Command (Half Duplex)*



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.)

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.)

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 8-5. Effect of IN#2 and T Command (Full Duplex Terminal)

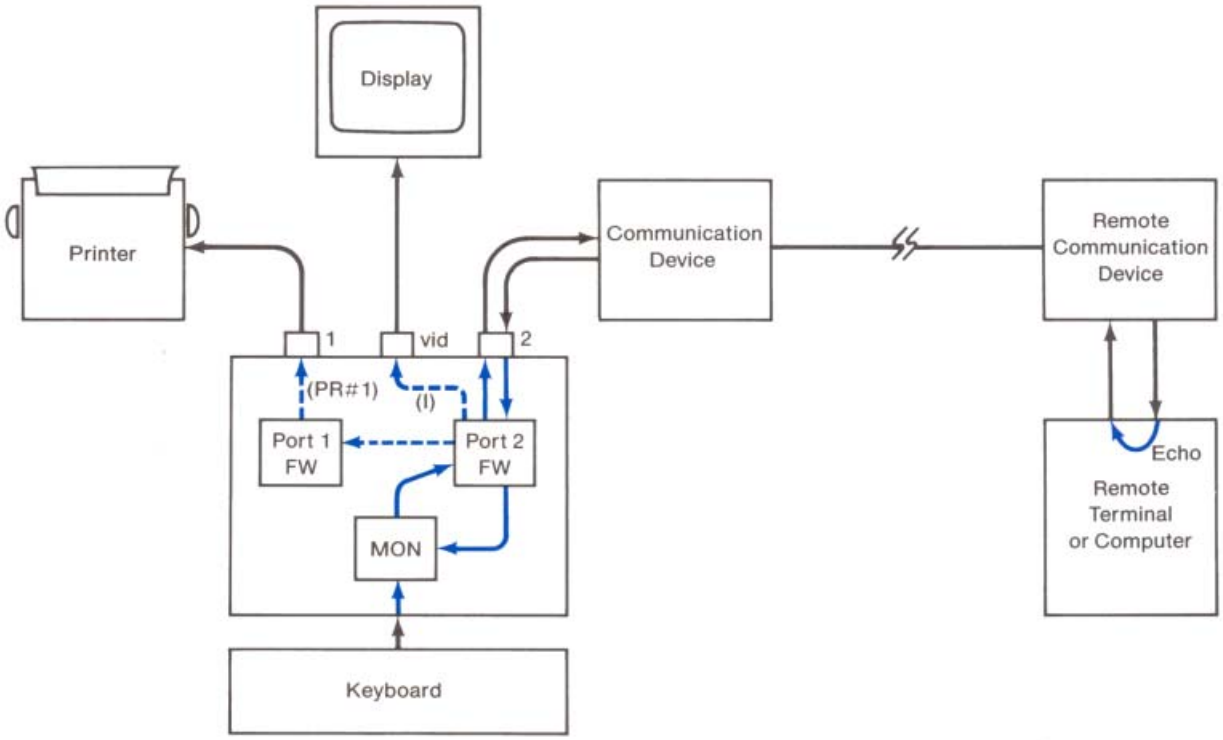


Figure 8-6 shows the flow of information when the Apple IIc is a full-duplex host. In this case, the local Apple IIc must echo input to output for the remote device. The setup commands include PR#2 in this case.

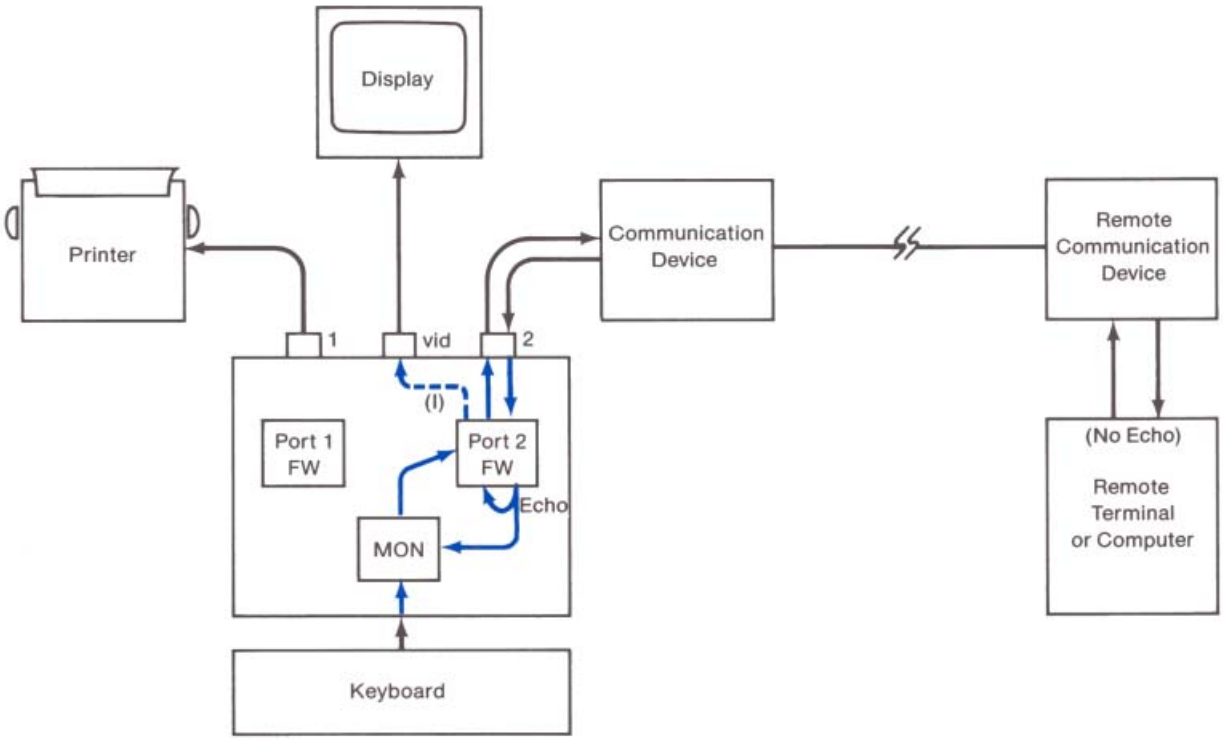
Warning

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.

Figure 8-6. *Effect of IN#2, PR#2 and T Command (Full Duplex Host)*



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 8 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, $800-$8FF 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) typed 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.

Mouse and Game Input

chapter 9

Section 9.1.6 gives an example of how to use the mouse as a hand control.

This chapter describes the mouse port and hand control (game) input capabilities of the Apple IIc. The mouse and hand controls use the same 9-pin connector on the back panel; the firmware uses the port as directed by keyboard or program commands.

A program can tell if a hand control is connected (section 9.2) but not if a mouse is connected, unless the computer user moves it.

9.1 Mouse Input

Table 9-1 is a summary of the characteristics of the mouse port and a guide to the other information in this part of the chapter.

Warning
If you want to ensure compatibility with mouse operation on the Apple IIe and other Apple II series computers, always use the I/O firmware entry points listed in Tables 9-4 and 9-5, rather than dealing with mouse hardware and RAM locations directly.

The mouse back panel connector is described in section 11.12.

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 section 5.2.2.

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	Table 9-2
Monitor Firmware Routines	None
I/O Firmware Entry Points	Table 9-3 and Table 9-4
Use of Screen Holes	Table 9-5

9.1.1 Mouse Connector Signals

The mouse uses the same 9-pin D-type miniature connector as the hand controls. However, the interpretation of the signals arriving on the pins differs depending on the commands and signals received. The names of the pin assignments when a mouse is connected are shown in Figure 11-37.

9.1.2 Mouse Operating Modes

Later sections of this chapter describe how to set various modes for mouse operation. This section tells what the modes are for.

In all the interrupt modes—that is, all but transparent mode—the user program should call the SERVEMOUSE routine to determine the source of an interrupt as soon as it receives one.

Transparent Mode

In this mode, a program must read screen holes to check for mouse movement. In reality, however, 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.

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 provides for efficient program coordination of the mouse cursor with mouse movement.

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 $3FE and $3FF; the interrupt handler can then update the cursor smoothly to its next screen position.

The receiving 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.

Section 5.2.2 contains recommendations for using MouseText characters with a mouse.

The maximum amount of mouse movement that can occur between successive VBLINT interrupts is limited only by 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 alleviates the program overhead required to poll the button constantly to provide fast response.

Movement/Button Interrupt Mode

This is a combination of the two modes just described. It provides the best response possible without constant polling of the mouse position and button. Processing of a main task can be concurrent with cursor and menu updating, as well as menu-selected command processing.

Vertical Blanking Active Modes

These modes are the same as the four just described except that they allow VBLINT interrupts to be sent to the user.

9.1.3 Mouse Hardware Page Locations

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 and masks out all mouse interrupts.

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 there is a change of at least a count of 1 in the X0 or Y0 coordinate. Since VBL occurs every sixtieth of a second, at most that amount of time will elapse before the resulting interrupt can be acknowledged and acted upon. To reset the VBL interrupt, read $C070.

Software can also select which edge of X0 and Y0 information will cause the XINT or YINT.

Once an interrupt has occurred, you can read the mouse's X1 and Y1 direction in data-bus bit 7 by reading address $C066 and $C067, respectively.

A program can read the status of the soft switches by reading one of the locations $C040-$C043 and then testing data bit 7.

Section 11.12 explains what X0, Y0, X1, Y1 are and what they mean with respect to mouse movement.

Appendix E explains how the firmware handles interrupts.

Warning

Table 9-2 is included here for your information; however, you should use the built-in firmware to access the mouse. If you do 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 Volume 2 of this manual.

Using the built-in firmware is much easier and guarantees compatibility with all other Apple II series computers.

Table 9-2. *Mouse Hardware Page Locations. (1) When IOUDIS is on, $C058-$C05F do not affect mouse, and $C05E and $C05F become DHIRES (Table 5-8). (2) Read or write to $C07x also resets VBLINT and triggers paddle timers. (3) These work only if IOUDIS is off. (4) This location is also the [Ó] key (Table 4-1). (5) This is also the location of the shift-key mod (Appendix F).*

Action	Hex	Name	Function	Notes
W	$C07E	IOUDIS	On: disable IOU access for addresses $C058 to $C05F; enable access to DHIRES switch	(1)
W	$C07F	IOUDIS	Off: enable IOU access for addresses $C058 to $C05F; disable access to DHIRES switch	(1)
R7	$C07E	RDIOUDIS	Read IOUDIS switch (1 = off)	(2)
R/W	$C058	DISXY	Disable (mask) X0 and Y0 movement interrupts	(3)
R/W	$C059	ENBXY	Enable (allow) X0 and Y0 movement interrupts	(3)
R7	$C040	RDXYMSK	Read status of X0/Y0 interrupt mask (1 = mask on)	
R	$CO48	RSTXY	Reset X0/Y0 interrupt flags	
R/W	$C05C	X0EDGE	Select rising edge of X0 for interrupt	(3)
R/W	$C05D	X0EDGE	Select falling edge of X0 for interrupt	(3)
R7	$C042	RDX0EDGE	Read status of X0 edge selector (1 = falling)	
R	$C015	RSTXINT	Reset mouse X0 interrupt flag	
R/W	$C05E	Y0EDGE	Select rising edge of Y0 for interrupt	(3)
R/W	$C05F	Y0EDGE	Select falling edge of Y0 for interrupt	(3)
R7	$C043	RDY0EDGE	Read status of Y0 edge selector (1 = falling)	
R	$C017	RSTYINT	Reset mouse Y0 interrupt flag	
R/W	$C05A	DISVBL	Disable (mask) VBL interrupts	(3)
R/W	$C05B	ENVBL	Enable (allow) VBL interrupts	(3)
R7	$C041	RDVBLMSK	Read status of VBL interrupt mask (1 = mask on)	
R	$C019	RSTVBL	Read and then reset VBLINT flag	
R/W	$C070	PTRIG	Reset VBLINT flag; trigger paddle timer	

Table 9-2—Continued. *Mouse Hardware Page Locations*

Action	Hex	Name	Function	Notes
R7	$C061	RDBTN0	Read hand control button status (1 = pressed)	(4)
R7	$C063	RD63	Read mouse button status (0 = pressed)	(5)
R7	$C066	MOUX1	Read status of X1 (mouse X direction) (1 = high)	
R7	$C067	MOUY1	Read status of Y1 (mouse Y direction) (1 = high)	

9.1.4 I/O Firmware Support

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, which you might need to use for sophisticated mouse applications, you can use direct firmware support. 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:

$00	Turn mouse off
$01	Set transparent mode
$03	Set movement-interrupt mode
$05	Set button-interrupt mode
$07	Set movement- or button-interrupt mode
$08	Turn mouse off, VBLINT active
$09	Set transparent mode, VBLINT active
$0B	Set movement-interrupt mode, VBLINT active
$0D	Set button-interrupt mode, VBLINT active
$0F	Set movement- or button-interrupt mode, VBLINT active

The firmware will then initialize 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.

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 $7FC, Table 9-5). 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 $77C 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. Don't re-enable 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
$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.

Table 9-3—Continued. *Mouse Firmware Routines*

$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 mode (see address $7FC in Table 9-5) in the accumulator, then call SETMOUSE.
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 re-enable interrupts.

Pascal Support

Table 9-4 lists the locations and values of the I/O firmware protocol that Pascal 1.1 and 1.2 use. However, Pascal must use a special attach driver to support the mouse.

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

In BASIC, before you can get input from the mouse, you must turn it on by printing PR#4 and then CHR$(1). This sets transparent mode. After that, re-enable 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:

```
PRINT CHR$(4)"PR#4"
PRINT CHR$(0)
PRINT CHR(4)"PR#3"
```

9.1.5 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.

Note: Some screen holes are different for the Apple IIe mouse. Refer to Appendix F.

Table 9-5. Mouse Peripheral Card RAM Locations

Scratch Area:

Location	Description
$478	Low byte of clamping minimum
$4F8	Low byte of clamping maximum
$578	High byte of clamping mimimum
$5F8	High byte of clamping maximum

Port 4 Screen Holes:

Location	Description
$47C	Low byte of X coordinate
$4FC	Low byte of Y coordinate
$57C	High byte of X coordinate
$5FC	High byte of Y coordinate
$67C	Reserved
$6FC	Reserved
$77C	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

$7FC — Mode byte (current mode; mask out bits 4-7 when testing)

Bit	1 Equals
7-4	Reserved
3	VBLINT active
2	VBLINT interrupt on button
1	VBLINT interrupt on movement
0	Mouse active

Port 5 Screen Holes:

Reserved

9.1.6 Using the Mouse as a Hand Control

This section describes how to use the mouse as if it were a set of hand controls, or an X-Y pointing device in port 4. If you turn the mouse on, the Monitor hand-control (game paddle) routines will take input from the mouse. This is possible because the mouse and the hand controls 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 this:

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.

Note: Many copy-protected games will not work with a mouse. Also, many games don't use built-in firmware for the paddles.

9.2 Game Input

The Apple IIc supports game paddles, joysticks, and other hand controls 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	**Description**	
$C061	Switch input 0 and [ó]	
$C062	Switch input 1 and [é]	
$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	**Name**	**Description**
$FB1E	PREAD	Read a paddle position
I/O Firmware Entry Points	None	
Use of Screen Holes	None	

9.2.1 The Hand Control Connector Signals

Several inputs are available on a 9-pin D-type miniature connector on the back of the Apple IIc: two one-bit inputs, or switches, and two analog inputs. You can access all of these signals from your programs.

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 controls to the 9-pin connector, the rotary controls use two analog inputs, and the push-buttons use two one-bit inputs. However, you can also use these inputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick.

Switch Inputs (SW0 and SW1)

The two one-bit inputs can be connected to the output of another electronic device that meets the electrical requirements (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 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, as shown in Table 9-6. Switch 0 and switch 1 are permanently connected to [ó] and [é] on the keyboard; these are the ones connected to the buttons on the hand controls. Location $C063 is a second address for the mouse button, so that a program can distinguish it from an [ó] keypress. When the mouse button is pressed, $C063 (Bit 7) goes from 1 to 0, and $C061 (Bit 7) goes from 0 to 1. When the mouse button is pressed, $C063 (Bit 7) goes from 1 to 0.

Analog Inputs (PDL0 and PDL1)

The two analog inputs are designed for use with 150K ohm variable resistors or potentiometers. The variable resistance is connected between the +5V supply and each input, so that it makes up part of a timing circuit (refer to section 11.13 for details). The circuit changes state when its time constant has elapsed, and the time constant varies as the resistance varies. Your program can measure this time by counting in a loop until the circuit changes state, or times out.

Before a program can read the analog inputs, it must first reset the timing circuits. Accessing memory location $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 one. If you PEEK at them from BASIC, the values will be 128 or greater. Within about 3 milliseconds, these bits will change back to zero—byte values less than 128—and remain there until you reset the timing circuits again. The exact 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.

You can read and reread the same paddle at arbitrarily short intervals. However, you must wait at least 3 milliseconds between reading one paddle and reading a different paddle.

9.2.2 Monitor Support

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 zero, 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.

PREAD

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 control. You pass the number of the hand control in the X register.

Warning
If the hand control number you furnish in the X register does not equal 0 or 1, strange things may happen.

Warning
The paddle and vertical blanking both use $C070. If you are reading the paddles and using VBL interrupts, disable interrupts before calling PREAD.

Using the Monitor

chapter 10

The System Monitor is a set of subroutines in the Apple IIc firmware. The Monitor 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. The starting addresses for all of the standard subroutines are listed in Appendix C. If you wish, you can call the standard subroutines from your programs in the same fashion.

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 programs in machine language to be executed directly by the Apple IIc's microprocessor
- to move and compare blocks of memory
- to invoke other programs from the Monitor.

10.1 Invoking the Monitor

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.

The System Monitor starts at memory location $FF69 (-151). To invoke the Monitor, you make a CALL 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.

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 $3D0.

Note: If DOS or ProDOS 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 into the three locations starting at address $3F2 (the reset-vector address and the power-up byte).

10.2 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.

A Monitor command 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 zeros; 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 many examples of the use of Monitor commands. 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 programmable memory.

10.3 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.

10.3.1 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.

10.3.2 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 location 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 eight—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 simply 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 causes the Monitor to 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
*
```

10.3.3 Changing Memory Contents

Section 10.3.2 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 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 replace it with another value.

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 the values of up to eighty-five consecutive locations at a time—or even more, if you omit leading zeros from the values—by typing only the initial address and colon followed by all the values separated by spaces; end with (RETURN). The Monitor will store 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 0 3

*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
*
```

10.3.4 Moving Data in Memory

You can copy a block of data stored in a range of memory locations from one area in memory to another 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. You give this information to the Monitor by means of three addresses: the address of the first location in the destination and the addresses of the first and last locations in the source. You specify the starting and ending addresses of the source range by separating them with a period. You separate the destination address from the range addresses with a less-than character (<), which you may think of as an arrow pointing in the direction of the move. Finally, you tell the Monitor that this is a MOVE command by pressing (M). The format of the complete MOVE command looks like this:

{destination} < {start} . {end} (M)

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. 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.

See section 10.6 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 sub-range and the values in this sub-range are replicated throughout the source range.

10.3.5 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 $D, copy them to locations starting at $300 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 will be compared.

Like the MOVE command, the VERIFY command also does unusual things if the destination address is within the source range; see section 10.6.

10.4 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.

10.4.1 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.

10.4.2 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

A=0A X=FF Y=D8 P=B0 S=F8
*:B0 02

*CONTROL-E

A=B0 X=02 Y=D8 P=B0 S=F8
*
```

10.5 Miscellaneous Monitor Commands

These Monitor commands enable you to change the video display format from normal to inverse and back, and to assign input and output to external devices.

10.5.1 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 such 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
*
```

10.5.2 Back to BASIC

If you are using one of the Apple disk operating systems (ProDOS or DOS), press CONTROL-RESET or type

See Appendix D.

```
3D0G
```

to return to the language you were using, with your program and variables intact.

Note: If you type the latter command, make sure that the third character you type is a *zero*, not a letter *O*. The letter *G* is the Monitor's GO command, described below in section 10.7.

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 had previously in BASIC will be lost. If you want to re-enter BASIC with your previous program and variables intact, use the CONTINUE BASIC command CONTROL-C.

10.5.3 Redirecting Input and Output

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

Chapter 3 lists the Apple IIc port numbers available.

|port number| (CONTROL)-(P)

A CONTROL-P command to port number 0 will switch the stream of output characters back to the Apple IIc's video display. However, use (ESC) (CONTROL)-(Q) if the enhanced video firmware is active (solid-block cursor).

In much the same way that the CONTROL-P command switches the output stream, the CONTROL-K command substitutes a device connected to a specified port for the Apple IIc's normal input device, the keyboard. The format for the command is:

|port number| (CONTROL)-(K)

Pressing (0) (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 section 3.1.

The CONTROL-P and CONTROL-K commands are the exact equivalents of the BASIC (but not DOS and ProDOS) commands PR# and IN#.

10.5.4 Hexadecimal Arithmetic

The Monitor will also perform one-byte hexadecimal addition and subtraction. 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
*FF+4
=03
*3-4
=FF
*
```

10.6 Special Tricks With the Monitor

This section describes some more complex ways of using the Monitor commands.

10.6.1 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. Since 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 grinds to a halt with a noisy beep and ignores the remainder of the input line.

10.6.2 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 3. Use the number to compute addresses for the MOVE command, like this:

{start+number}<{start}.{end-number}(M)

This MOVE command will first replicate the pattern at the locations immediately following the original pattern, then replicate 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 do a similar trick with 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 $300 to $320 with zeros 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)
*
```

10.6.3 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 $200. 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
*
```

10.6.4 Creating Your Own Commands

The USER command, (CONTROL)-(Y), forces the Monitor to jump to memory location $3F8. 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 $300; the command line that starts with $3F8 stores a jump to $300 at location $3F8.

```
*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

*
```

10.7 Machine-Language Programs

The main reason to program in machine language is to get more speed. 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.

10.7.1 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 $300, examine it to be sure you typed it correctly, then type 300G 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
*
```

10.7.2 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**.

Since programs that translate assembly language into machine language are called **assemblers**, a program like the Monitor's LIST command that translates machine language into assembly language is called a **disassembler**.

The Monitor's LIST 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 word **mnemonic** comes from the same root as memory and refers to short acronyms that are easier to remember than the hexadecimal operation codes themselves. For example, for *clear carry* you write CLC instead of $18.

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      JSR   $FDED
        FD
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 zeros 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 will display another screenfull of instructions, starting where the previous display left off.

10.8 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).
ESC 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 $3F8.

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.

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.

11.1 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 Apple recommends that you use its carrying case or some other protective container (such as an attache case).

Table 11-1 defines the conditions under which the Apple IIc is designed to function properly.

Table 11-1. *Summary of Environmental Specifications*

Operating Temperature:	10° to 40° C (50° to 104° F)
Relative Humidity:	20% to 95%
Line Voltage:	105 to 129 VAC (normal USA voltage range)

You should treat the Apple IIc with the same kind of care as any other electrical appliance. You should 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.

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 manage to 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.

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.

11.2 Power Requirements

The electrical power that the Apple IIc, and everything that draws power from it, is limited by the tolerances of its 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.

11.2.1 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. 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.

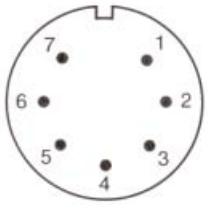
Table 11-2. *Power Supply Specifications*

Line Voltage:	105 to 129 VAC, 60 Hz
Maximum Input Power Consumption:	25 W
Supply Voltage:	+15 VDC (nominal)
Supply Current:	1.2 A (nominal)

11.2.2 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.

Figure 11-1. *External Power Connector*



Pin	Signal
1	Not connected
2,3	Signal ground
4	Shield ground
5,6	+15 VDC
7	Not connected

Table 11-3. External Power Connector Signals

Pin Number	Name	Description
1,7		Not connected
2,3	Ground	Common electrical ground
4	Chassis	Chassis ground
5,6	+15V	+15 volt DC input to converter

11.2.3 The Internal Converter

The internal converter in the Apple IIc operates on 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. Listed amperages are those available in addition to the current drawn by the Apple IIc itself. Minus 5 volts is derived from the -12 volts 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. It is small and lightweight, and it generates less heat than other types of voltage converters do.

The Apple IIc's 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.

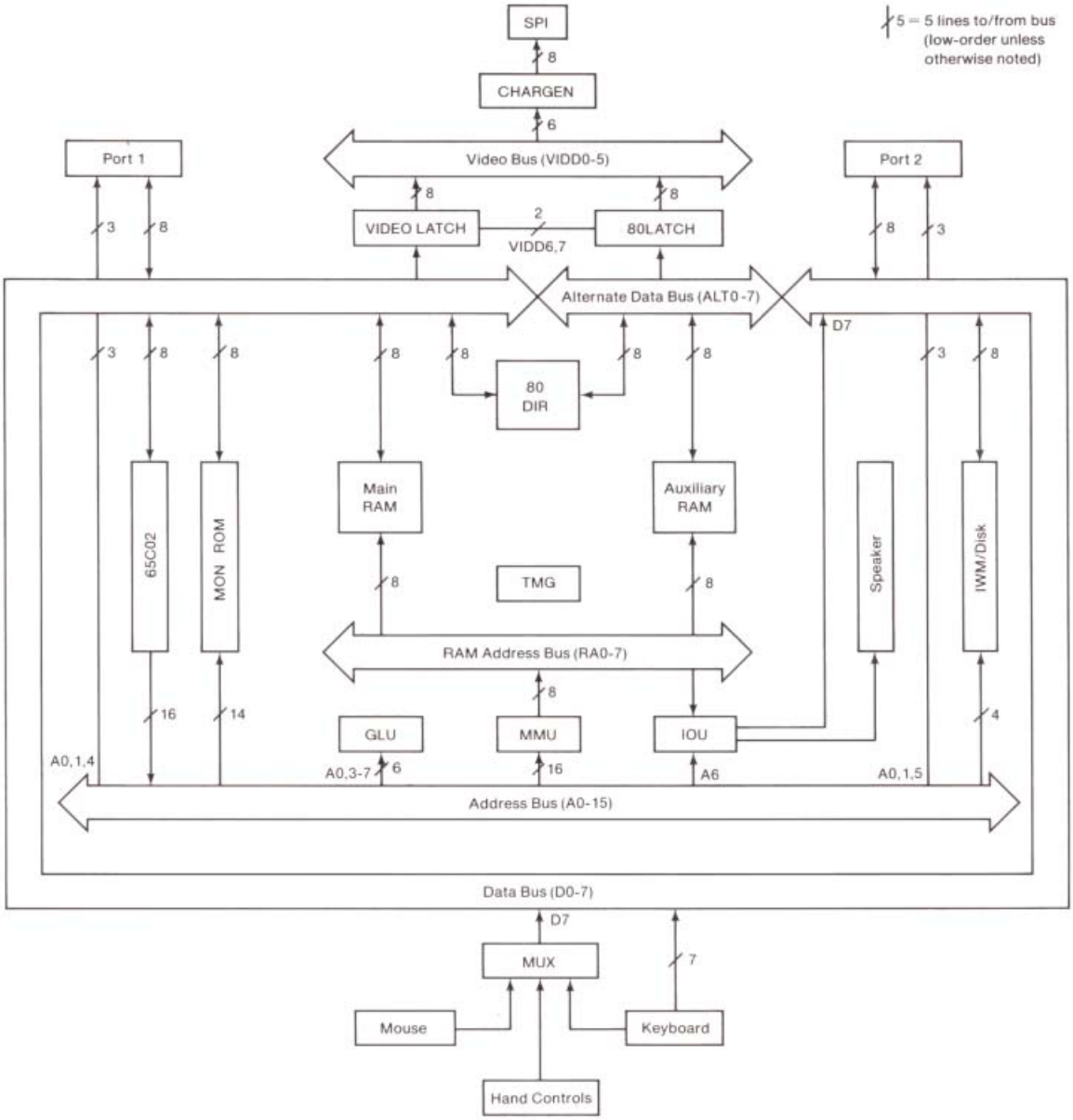
Any time one of these malfunctions occurs, the protection circuit varies the duty cycle of the oscillator, and all the output voltages drop to zero.

11.3 Apple IIc Overall Block Diagram

A full set of schematic diagrams of the Apple IIc appears in section 11.14.

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.

Figure 11-2. *Apple IIc Block Diagram*



11.4 The CMOS 65C02 Microprocessor

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 eight-bit operations per second.

Note: You should not use the clock rate as a 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.

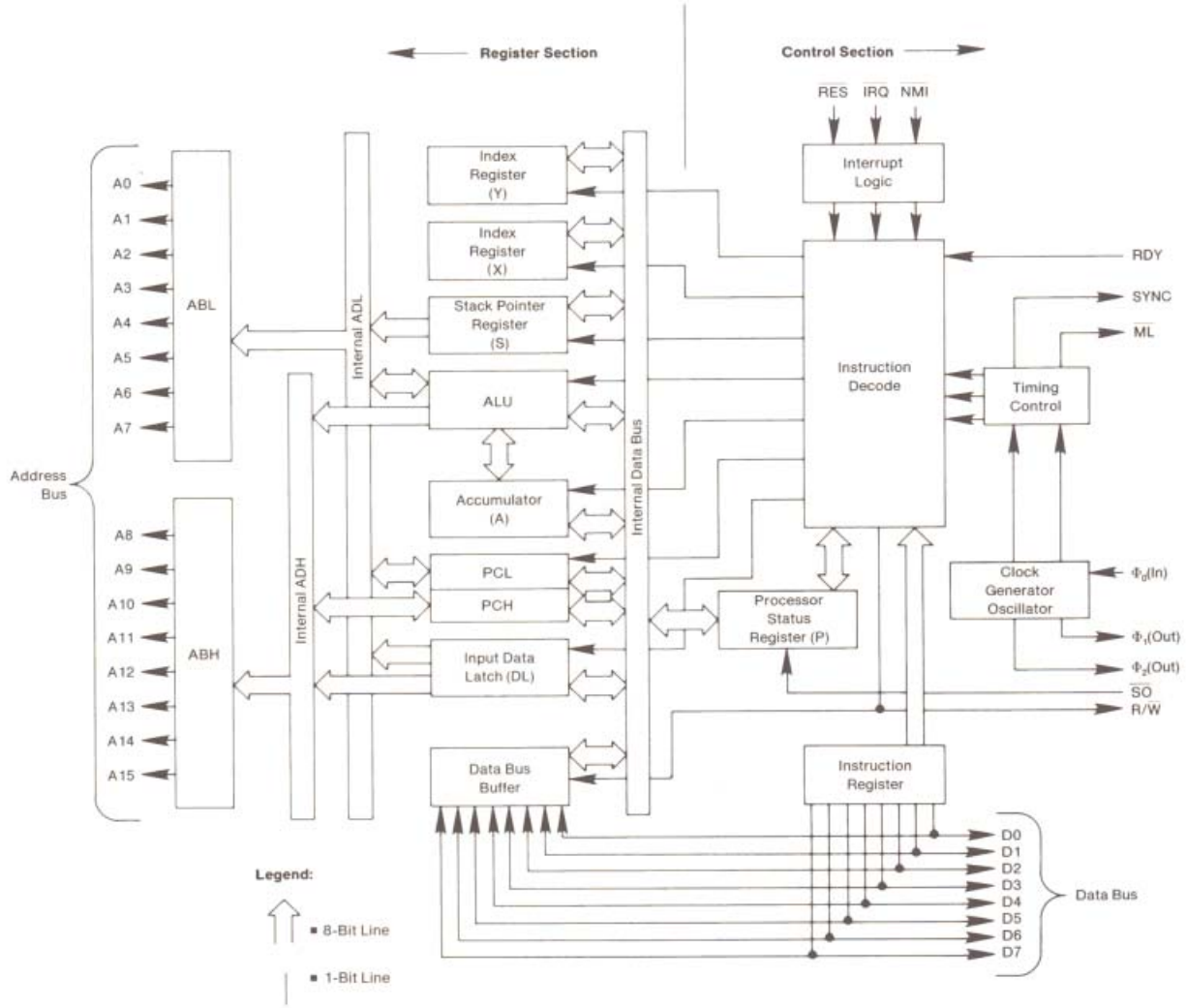
In addition to requiring lower power than earlier NMOS 6502 processors, the 65C02 in the Apple IIc provides the programmer with 27 new instructions. However, programs that use these additional instructions will not be backward compatible with other Apple II series computers that are not equipped with a CMOS 6502.

These instructions are described in Appendix A.

11.4.1 65C02 Block Diagram

Figure 11-3 is a block diagram of the 65C02 microprocessor. Table 11-5 contains the general specifications of this chip.

Figure 11-3. *65C02 Block Diagram. Copyright 1982, NCR Corporation. Used by permission of NCR Corporation, Dayton, Ohio.*



The 65C02 has a sixteen-bit address bus, giving it an address space of 64K (two to the sixteenth power or 65536) bytes. The Apple IIc uses special techniques to address a total of more than 64K: for details, refer to Chapter 2.

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:	Eight bits wide
Address Bus:	Sixteen bits wide
Address Range:	65,536 (64K)
Interrupts:	IRQ (maskable) NMI (non-maskable) BRK (programmed)
Operating Voltage:	+5V (± 5%)
Power Dissipation:	5 mW (at 1 MHz)

11.4.2 65C02 Timing

The operation of the Apple IIc is controlled by a set of synchronous timing signals, sometimes called clock signals. The frequency of the oscillator that generates the master timing signal is 14.318 MHz. Circuitry in the Apple IIc uses this clock 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 operation of the 65C02 are described in this section. Other timing signals are described in sections 11.6.2, 11.9.3, and 11.9.4.

The main 65C02 timing signals are listed in Table 11-6, and their relationships are diagrammed in Figure 11-4. The 65C02 clock signals are $\phi 1$ and $\phi 0$, complementary signals at a frequency of 1.0227 MHz. The Apple IIc signal named $\phi 0$ is equivalent to the signal called $\phi 2$ in Appendix A (it isn't identical—it's a tiny bit early).

Figure 11-4. 65C02 Timing Signals

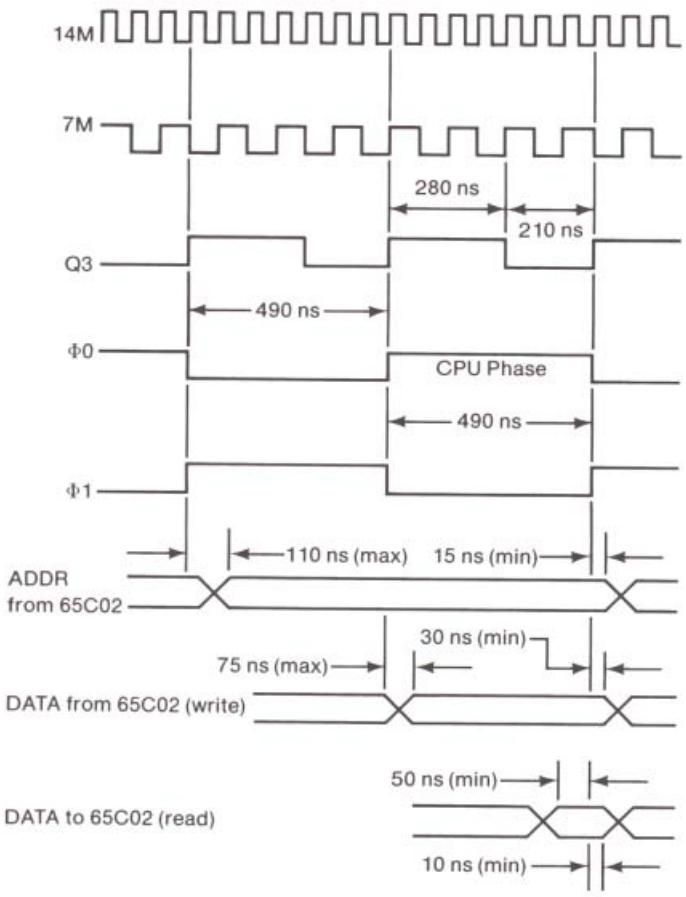


Table 11-6. 65C02 Timing Signal Descriptions

Signal Name	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
ϕ0	Phase 0 of 65C02 clock, 1.0227 MHz; complement of ϕ1
ϕ1	Phase 1 of 65C02 clock, 1.0227 MHz; complement of ϕ0

The operations of the 65C02 are related to the clock signals in a simple way: internal during ϕ1, external during ϕ0. The 65C02 puts an address on the address bus during ϕ1. This address is valid not later than 110 nanoseconds after ϕ1 goes high and remains valid through all of ϕ0. The 65C02 reads or writes data during ϕ0. If the 65C02 is writing, the read/write signal is low during ϕ0 and the 65C02 puts data on the data bus. The data is valid not later than 75 nanoseconds after ϕ0 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 ϕ0.

11.5 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 (IWM)

The soft switches used for controlling 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 of 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.

11.5.1 The Memory Management Unit (MMU)

The circuitry inside the MMU implements these soft switches, which are described in the following chapters:

- Page 2 display (PAGE2): 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.

Note: A signal name followed by an asterisk is active low, that is, it is true when not asserted.

The 64K dynamic RAMs used in the Apple IIc use a multiplexed address, as described below in the section *Dynamic-RAM Timing* (in section 11.6.2). The MMU generates this multiplexed address for memory reading and writing by the 65C02 CPU.

Figure 11-5. The MMU Pinouts

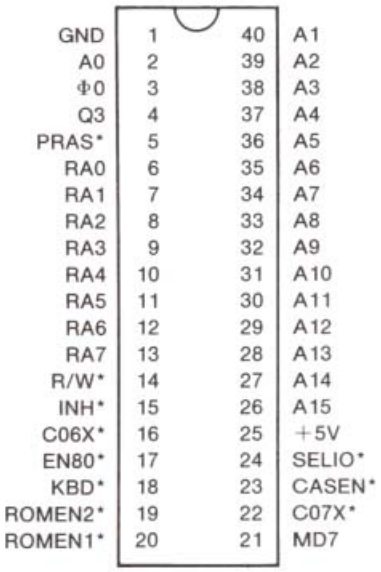


Table 11-7. The MMU Signal Descriptions

Pin Number	Name	Description
1	GND	Power and signal common
2	A0	65C02 address input
3	ϕ0	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 +5 V)
16	C06X*	Causes $C06x outputs to go to 0 during ϕ0
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 ϕ0
23	CASEN*	Enables main RAM
24	SELIO*	Goes to 0 during ϕ0 for any access to $C0 page except $C08x, Bx, Cx or Fx
25	+5V	Power
26-40	A15-A1	65C02 address input

11.5.2 The Input/Output Unit (IOU)

The circuitry inside the 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 MMU pinouts; Table 11-8 describes the signals.

The 64K dynamic RAMs used in the Apple IIc require a multiplexed address, as described in the section *Dynamic-RAM Timing* (in section 11.6.2). The IOU generates this multiplexed address for the data transfers required for display and memory refresh during clock phase 1. The way this address is generated is described in section 11.9.1.

Figure 11-6. *The IOU Pinouts*

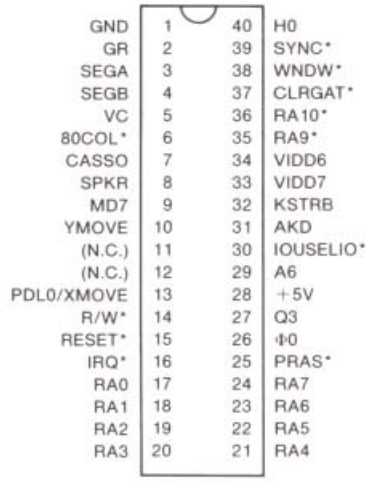


Table 11-8. *The IOU Signal Descriptions*

Pin Number	Name	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	Display 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. *The IOU Signal Descriptions*

Pin Number	Name	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	ϕ0	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*	Display blanking signal
39	SYNC*	Display synchronization signal
40	H0	Display horizontal timing signal (low bit of character counter)

11.5.3 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.

Figure 11-7. *The TMG Pinouts*

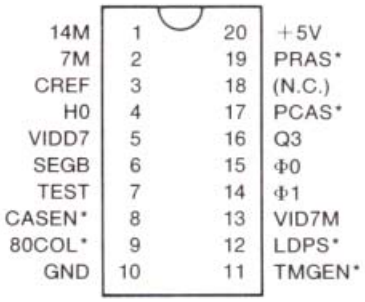


Table 11-9. *The TMG Signal Descriptions*

Pin Number	Name	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*	Enable 80-column display mode
10	GND	Power and signal common
11	TMGEN*	Enable master timing
12	LDPS*	Video shift-register load enable
13	VID7M	Video dot clock enable, 7 MHz or continuous 0
14	ϕ1	Phase 1 system clock
15	ϕ0	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

11.5.4 The General Logic Unit (GLU)

The General Logic Unit is a single-chip version of the miscellaneous logic required for the system. It provides all RAM read/write timing, double-high-resolution enable/disable, soft switch status registers and write command registers. It also provides IOU control for mouse interrupts and double-high-resolution soft-switches. Its pin assignments are shown in Figure 11-8 and its signals are listed in Table 11-10.

Figure 11-8. *The GLU Pinouts*

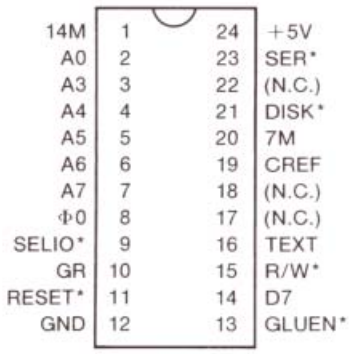


Table 11-10. *The GLU Signal Descriptions*

Pin Number	Name	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	PH0	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	Vcc	+5 volt supply

11.5.5 The Disk Controller Unit (IWM)

For further information on group code recording, refer to section 11.10.

The IWM is an integrated GCR (group code recording) disk drive controller in its state right after reset. In addition, it 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. *The IWM Pinouts*

Table 11-11. *The IWM Signal Descriptions*

Pin Number	Name	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-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-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
20	DR1*	Drive 1 select
21	WRPROT	Write-protect input; this can be polled via bit 7 of the status register.

Table 11-11—Continued. *The IWM Signal Descriptions*

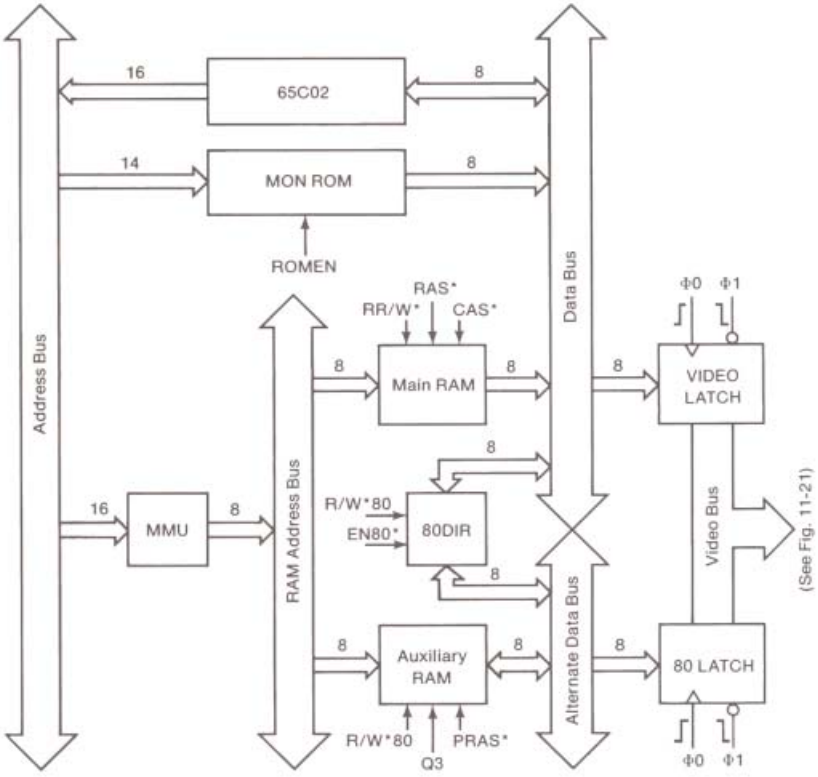
Pin Number	Name	Description
22	RDDATA	Serial data input line. The IWM synchronizes the falling transition of each pulse.
23	RESET*	IWM reset: places all IWM outputs in their inactive state and sets all state and mode register bits to zero.
24	7M	7 MHz clock input
25	Q3	A 2.0 MHz clock input used to qualify the timing of the serial data being written or read.
26	Vcc	The +5 volt supply
27	SEEKPH3	Stepper motor control phase 3
28	SEEKPH1	Stepper motor control phase 1

11.6 Memory Addressing

The 65C02 microprocessor can 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 overall memory bus organization and memory selection signals.

Figure 11-10. *Memory Bus Organization*



Note: Some Apple IIc's have ROMs with 27xx designations, some have 23xx. They are functionally equivalent.

11.6.1 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.

Figure 11-11. *The 23128 ROM Pinouts*

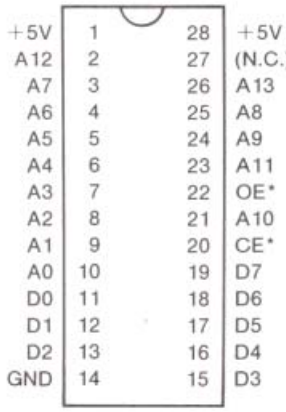


Figure 11-12. *The 2316 ROM Pinouts*

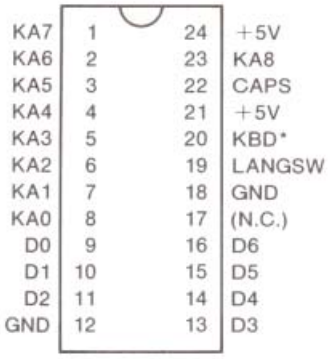
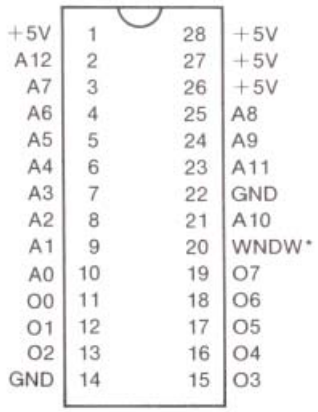


Figure 11-13. *The 2364 ROM Pinouts*



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 to $DFFF. The address space from $C300 to $C3FF and much of $C800 to $CFFF contains the enhanced video firmware.

These ROM address allocations are approximately true (some space sharing takes place):

- ROM addresses $C000 to $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 entry point to mouse interface support.
- ROM addresses $C500 to $C5FF are reserved.
- ROM address $C600 is 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 to $F7FF contain the Applesoft BASIC interpreter; addresses $F800 through $FFFF contain the Monitor firmware.

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.

11.6.2 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 described in section 11.4.2 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 $\phi 0$, and the display circuits read data only during $\phi 1$.

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 it 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 sixty times per second.

The dynamic RAM devices used in the Apple IIc also need a kind of refreshment, because the data is stored in the form of electric charges which diminish with time and must be replenished every so often. The Apple IIc is designed so that refreshing the display also refreshes the dynamic RAMs. The next few paragraphs explain how this is done.

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**. Since 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).

Figure 11-14. *The 64K RAM Pinouts*

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 in section 11.9.1, 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 eight 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 two milliseconds (see Table 11-12). This more than satisfies the refresh requirements of the dynamic RAMs.

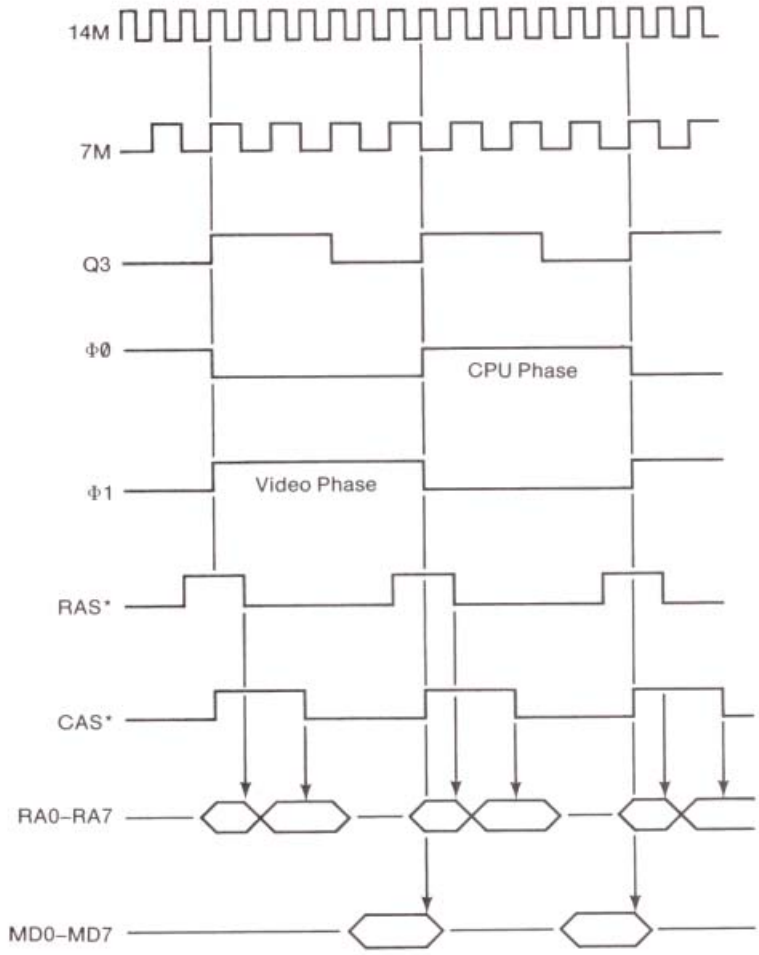
Table 11-12. *RAM Address Multiplexing*

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

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 is strobed by the falling edge of $\phi 0$, and display data is strobed by the falling edge of $\phi 1$, as shown in Figure 11-15.

Figure 11-15. RAM Timing Signals



The RAM timing looks complicated because the RAM address is multiplexed, as described in the previous section. 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 labeled 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 Name	Description
ϕ0	Clock phase 0 (CPU phase)
ϕ1	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

11.7 The Keyboard

The Apple IIc's keyboard is a matrix of keyswitches 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.

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, along with 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 11-16. *Keyboard Circuit Diagram*

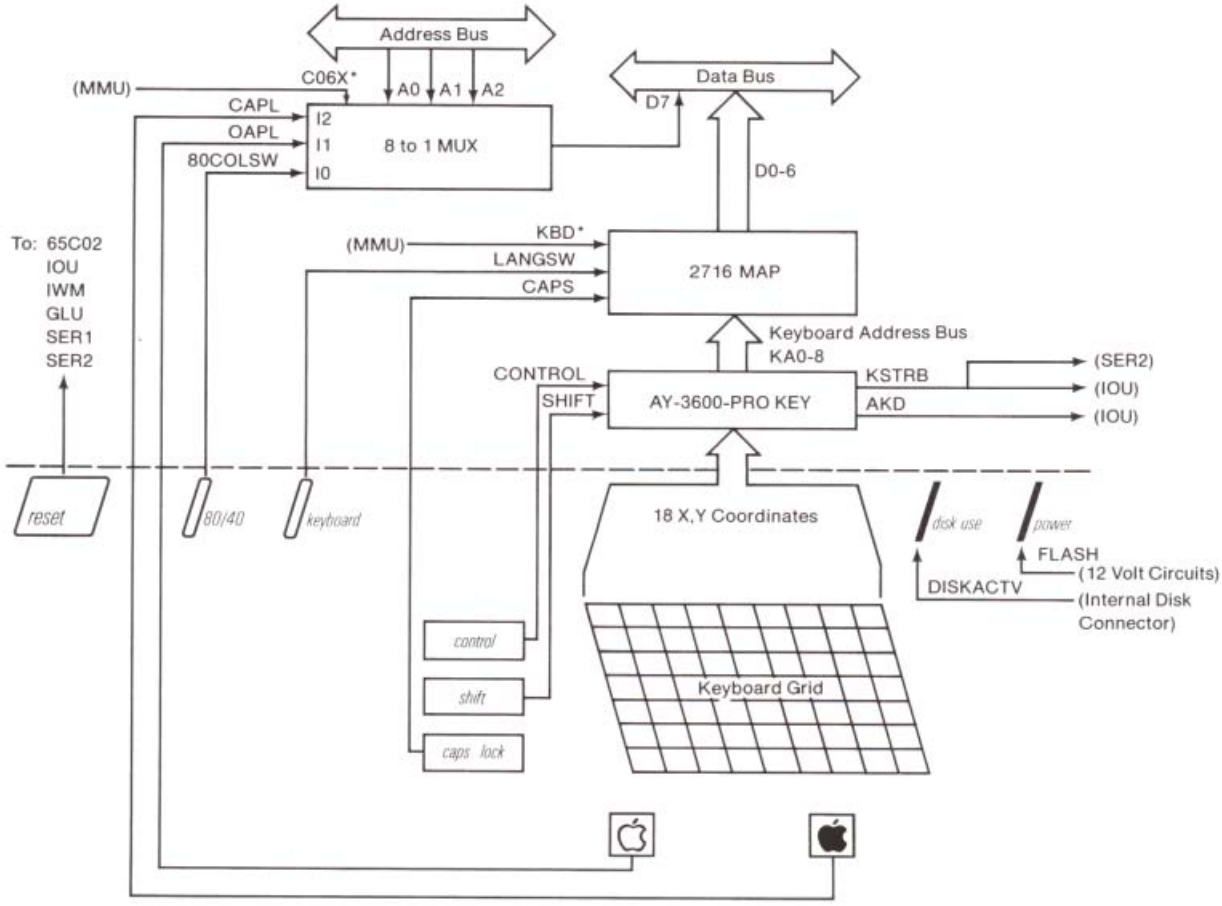
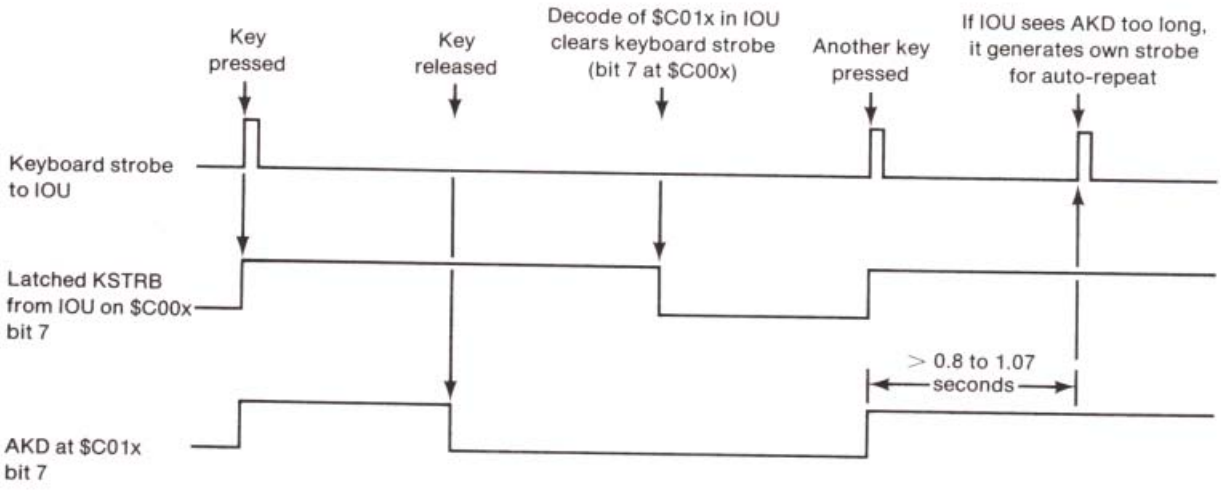


Figure 11-17 illustrates the events that occur when a key is pressed, when the keypress is detected by a program, and when a key is pressed and held for more than about a second.

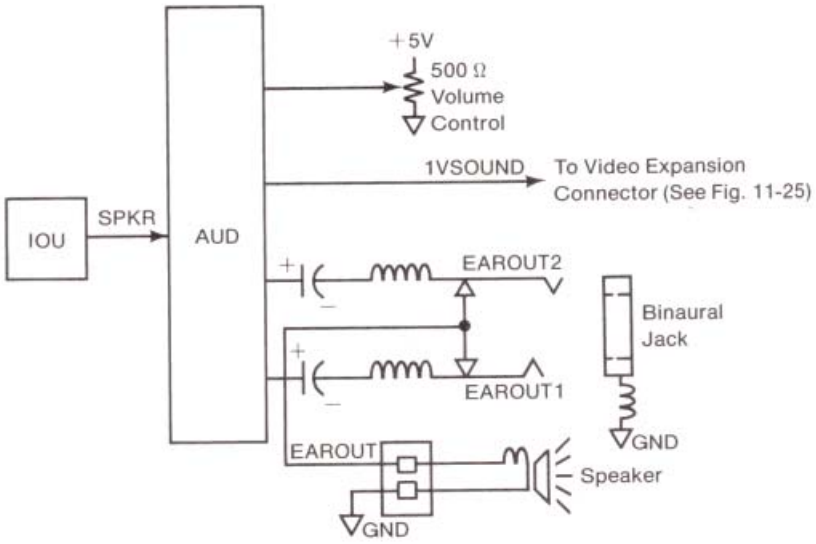
Figure 11-17. *Keyboard Signals*



11.8 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*



11.8.1 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.

11.8.2 Output Jack

Next to the volume control, along the lower-left side of the Apple IIc case, there is a 3.5 mm stereo miniphone jack. Although speaker output is monaural, the jack accommodates stereo headphone plugs (as well as monaural, of course), providing sound to both channels. Inserting a headphone plug disconnects the internal Apple IIc speaker.

11.9 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 is 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 IIcs 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). 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.

11.9.1 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 normally from 0 to 64, then start over at 0. Whenever this happens, HPE* forces another count with H0 through H5 held at zero, thus extending the total count to 65.

The IOU uses the forty horizontal count values from 25 through 64 in generating the low-order part of the display data address, as described in section 11.9.3. 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 zero. 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.)

11.9.2 Display Memory Addressing

As described in section 5.7, 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 is 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 $400 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 it 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 forty 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 in section 11.9.4.

11.9.3 Display Address Mapping

Consider the simplest display on the Apple IIc, the 40-column text mode. To address forty columns requires six bits, and to address twenty-four rows requires another five bits, for a total of eleven address bits. Addressing the display this way would involve 2048 (two to the eleventh 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 in section 11.6.2.

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 called 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 one appears as the low-order bit of the addend. The carry bit generated with the sum is not used.

Figure 11-19. *Display Address Transformation*

			V3	Carry in
H5*	V3	H4	H3	Augend
V4	H5*	V4	1	Addend
S3	S2	S1	S0	Sum

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 zero and the horizontal count is 24: H0, H1, H2, and H5 are zeroes, and H3 and H4 are ones. The value of the sum is zero, so the memory location for the first character on the display is the first location in the display page, as you might expect.

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 zero to four 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 zero again, and V3 changes to one. 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.

Figure 11-20. 40-Column Text Display Memory. Memory locations marked with a double asterisk (**) are screen holes, described in section 2.5.1.

	128 Bytes			
	40 Bytes	40 Bytes	40 Bytes	8 Bytes
$400	Row 0	Row 8	Row 16	**
$480	Row 1	Row 9	Row 17	**
$500	Row 2	Row 10	Row 18	**
$580	Row 3	Row 11	Row 19	**
$600	Row 4	Row 12	Row 20	**
$680	Row 5	Row 13	Row 21	**
$700	Row 6	Row 14	Row 22	**
$780	Row 7	Row 15	Row 23	**

Figure 11-20 shows how groups of three forty-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.

*Table 11-14. Display Memory Addressing. **For these address bits, see text and Table 11-15.*

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

Address bits marked with double asterisks (**) are different for different modes: see Table 11-15 and section 11.9.4.

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.

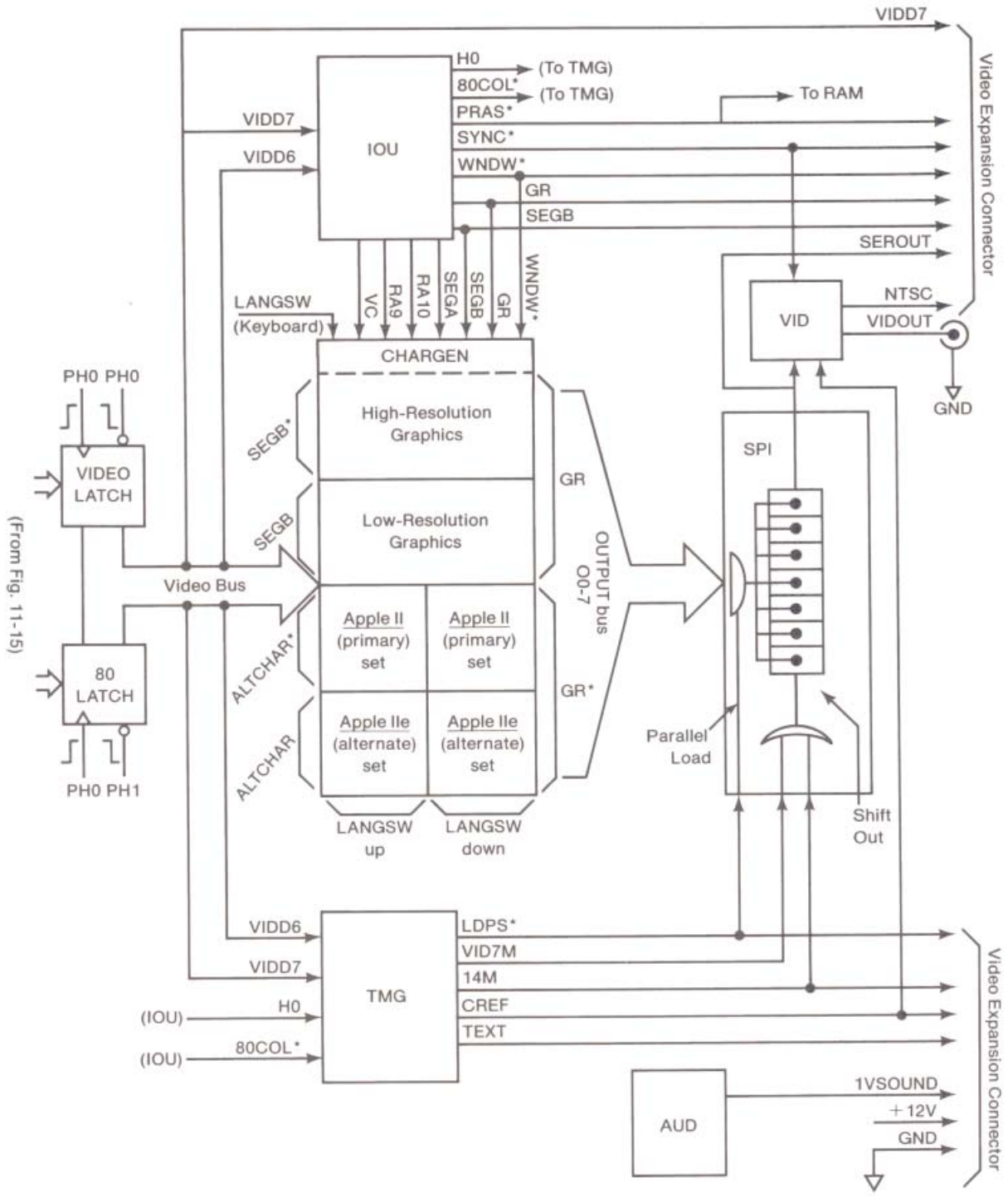
Table 11-15. Memory Address Bits for Display Modes. (. means logical AND; ' means logical NOT.)

	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

11.9.4 Video Display Modes

The different display modes all use the address-mapping scheme described in section 11.9.3, but they use different-sized memory areas in different locations. Section 11.9.4 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 $400 and $800. 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 zero. 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 $\phi 0$), but their outputs are sent to the character generator alternately by the falling edge of $\phi 0$ and $\phi 1$. In 80-column mode, LDPS* loads data from the character generator into the shift register twice during each microsecond, once during $\phi 0$ and once during $\phi 1$, and VID7M remains low, enabling the clock continuously at 14M (Figure 11-23).

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 one of sixteen 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 fourteen bits from each eight-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 eight bits; the last two 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 ones and zeros gets displayed as a line of color. The high-resolution graphics subroutines produce the appropriate bit patterns by masking the data bits with alternating ones and zeros.

To produce different colors, the bit patterns must have different phase relationships to the 3.58 MHz color signal. If alternating ones and zeros produce a certain color, say green, then reversing the pattern to zeros and ones 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, 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, seven 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 appears in columns 0-6, 14-20, and so on, up to columns 547-552. Data from main memory appears 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 will be 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 $\phi 0$ 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 $\phi 0$ 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.

Figure 11-22. *7 MHz Video Timing Signals (40-Column, Low-Resolution and High-Resolution Display)*

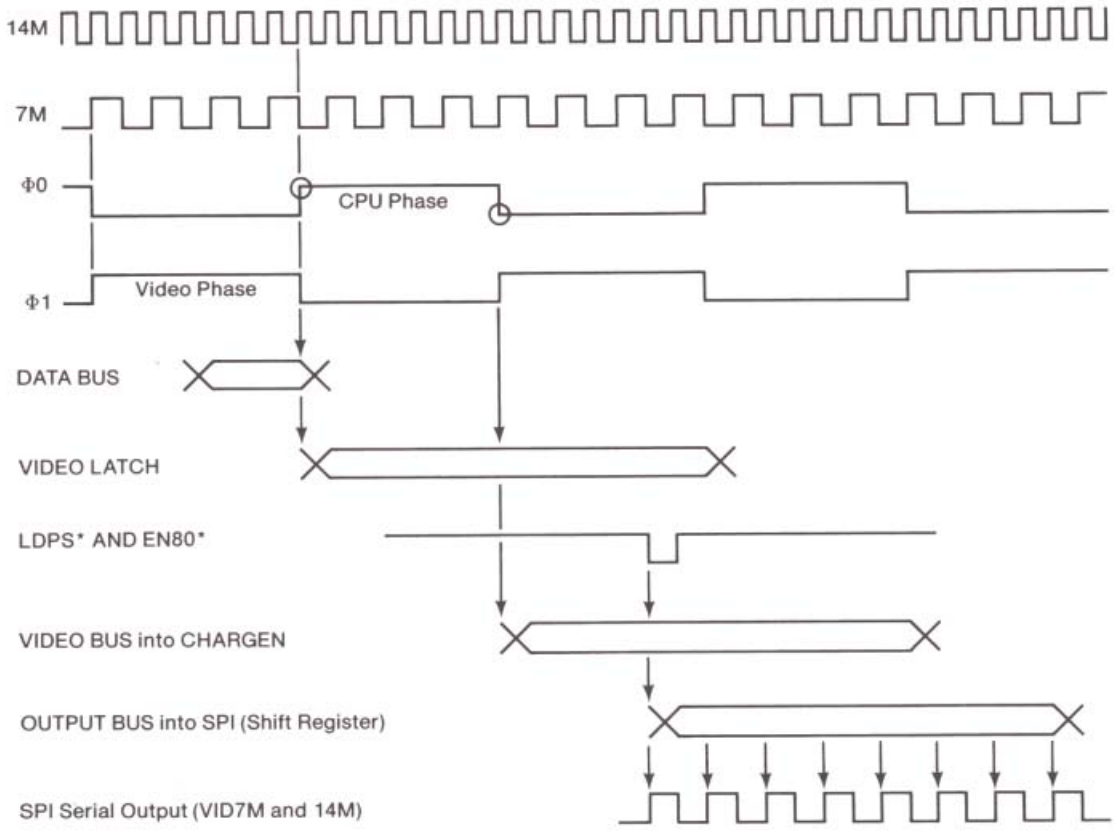
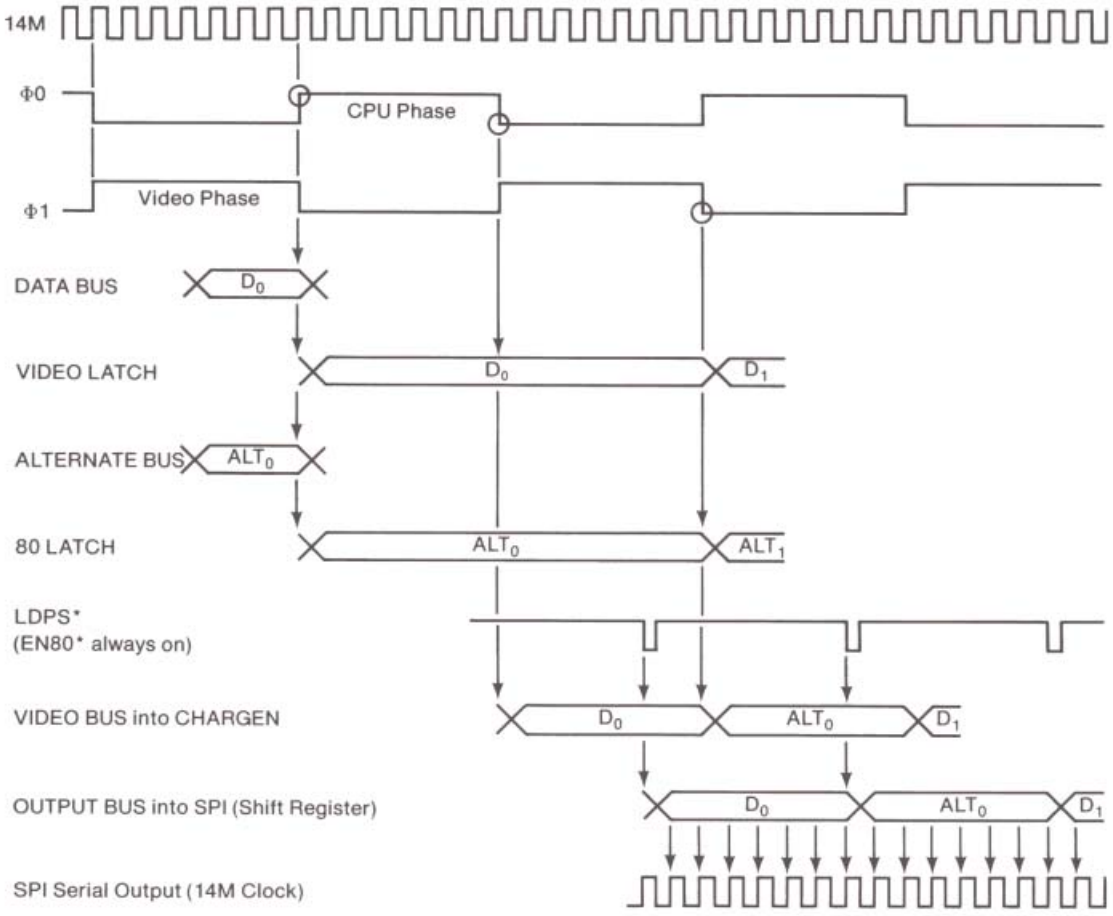


Figure 11-23. 14 MHz Video Timing Signals (80-Column and Double-High-Resolution Display)



11.9.5 Video Output Signals

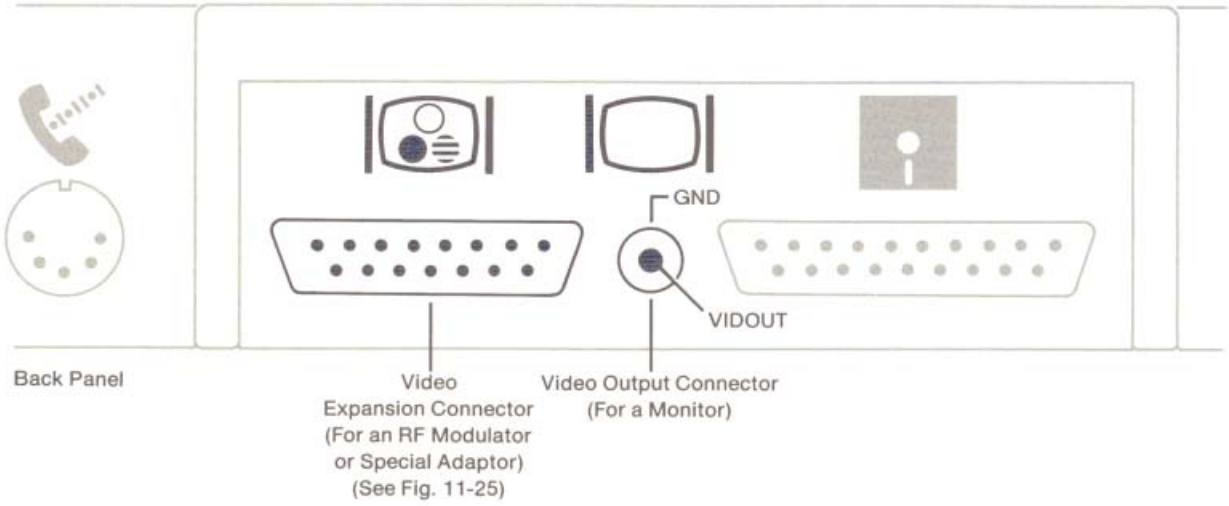
VID is a video-amplifier hybrid circuit.

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 phono jack on the back of the Apple IIc
- at the video expansion connector (pin 12) on the back panel. (Table 11-17).

Figure 11-24. *Video Output Back Panel Connectors*



Monitor Output

The sleeve of the phono jack 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.

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.

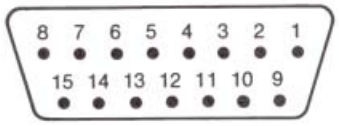
In Table 11-17, the column labeled *Deriv* indicates what clock signals the video signals are derived from. LDPS, CREF and PRAS have a maximum delay of 30 ns from the appropriate 14 MHz rising edge. SEROUT is clocked out of a 74LS166 by the rising edge of 14M and has a maximum delay of 35 ns. VIDD7 is driven from a 74LS374 and has a maximum delay of 28 ns from the rising and (if 80-column) falling edges of $\phi 1$.

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 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.

Warning
Several of these signals, such as 14 MHz, must be buffered within about four 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.

Figure 11-25. The Video Expansion Connector Pinouts



Pin	Signal	Pin	Signal
1	TEXT	9	PRAS*
2	14MHz	10	GR
3	SYNC*	11	SEROUT*
4	SEGB	12	NTSC
5	1VSOUND	13	GND
6	LDPS*	14	VIDD7
7	WNDW*	15	CREF
8	+12V		

Table 11-17. The Video Expansion Connector Signals

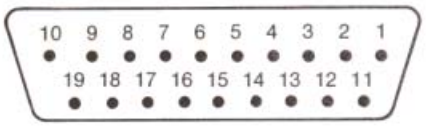
Pin	Deriv	Name	Description
1	ϕ0	TEXT	Video text signal from TMG; set to inverse of GR, except in double-high-resolution mode
2		14MHz	14 MHz master timing signal from the system oscillator
3	Q3	SYNC*	Display horizontal and vertical synchronization signal from IOU pin 39
4	PRAS	SEGB	Display vertical counter 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	14MHz	LDPS*	Video shift-register load enable from pin 12 of TMG
7	PRAS	WNDW*	Active area display blanking; includes both horizontal and vertical blanking
8		+12 V	Regulated +12 volts DC.; can drive 350 mA
9	14MHz	PRAS*	RAM row-address strobe from TMG pin 19
10	PRAS	GR	Graphics mode enable from IOU pin 2
11	14MHz	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	ϕ0	VIDD7	From 74LS374 video latch; causes half-dot shift if high
15	14MHz	CREF	Color reference signal from TMG pin 3; 3.58 MHz

Warning
Use caution—The maximum allowable current drain of +12 V 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.

11.10 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 in section 11.5.5.

Figure 11-26. *Disk Drive Connector*



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		

Warning
The power available at this connector is for a Disk II 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.

*Table 11-18. Disk Drive Connector Signals. **Refer to the warning preceding this table.*

Connector Pin Number	Name	Description
1,2,3,4	GND	Ground reference and supply
6	+5 V	+5 volt supply**
7,8	+12	+12 volt supply**
9	EXTINT*	External interrupt
10	WRPROT	Write-protect input
11-14	PH0-4	Motor phase 0-4 output
15	WRREQ*	Write Request
17	DR1*	Drive 1 select
18	RDDATA	Read data input
19	WRDATA	Write data output

11.11 Serial I/O

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.

The RS-232 signals are defined in the Glossary.

Figure 11-28 is a detail block diagram of the 6551 ACIA. The registers are described in sections 11.11.1 through 11.11.4.

Figure 11-27. Serial Port Circuits

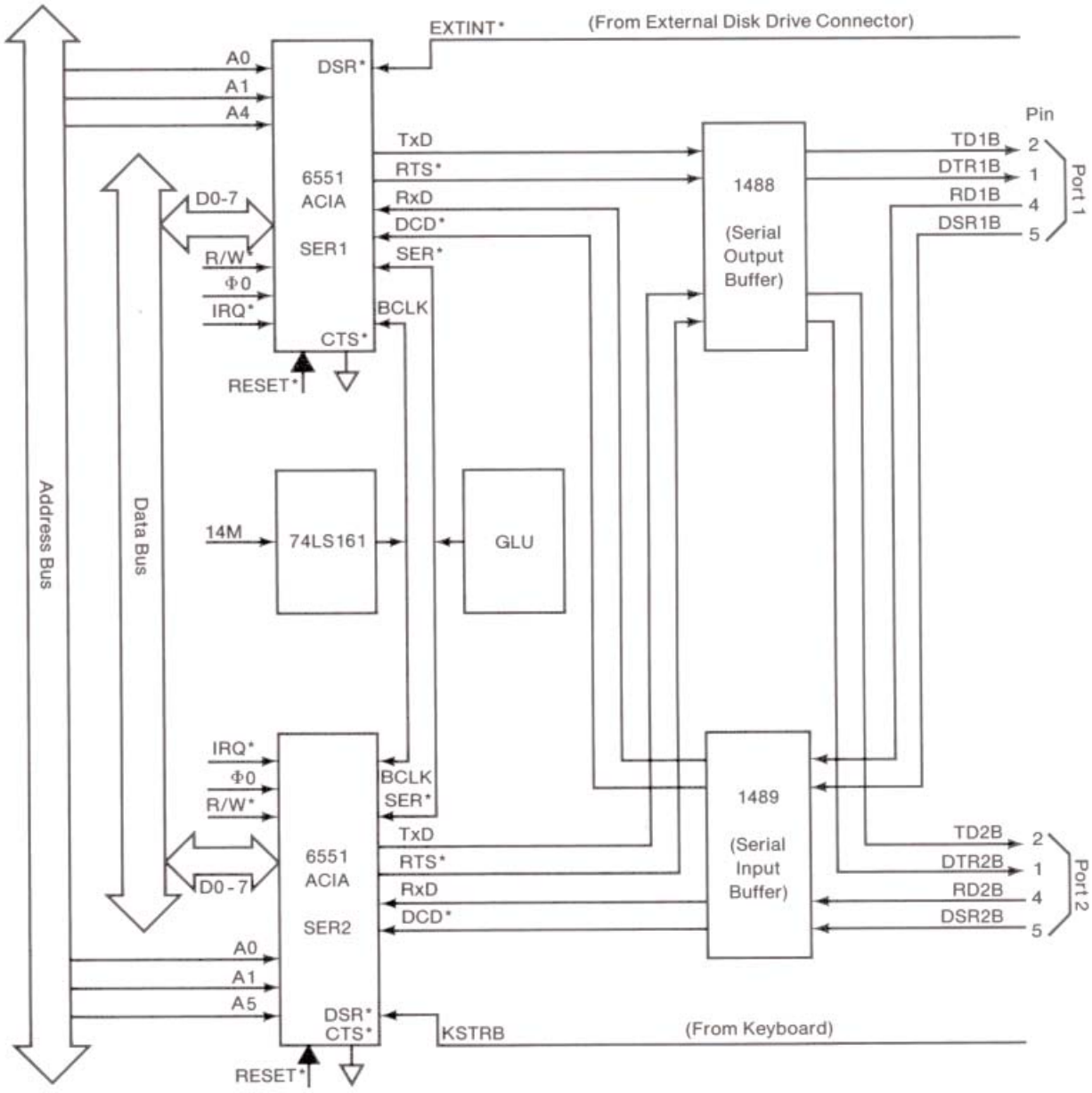
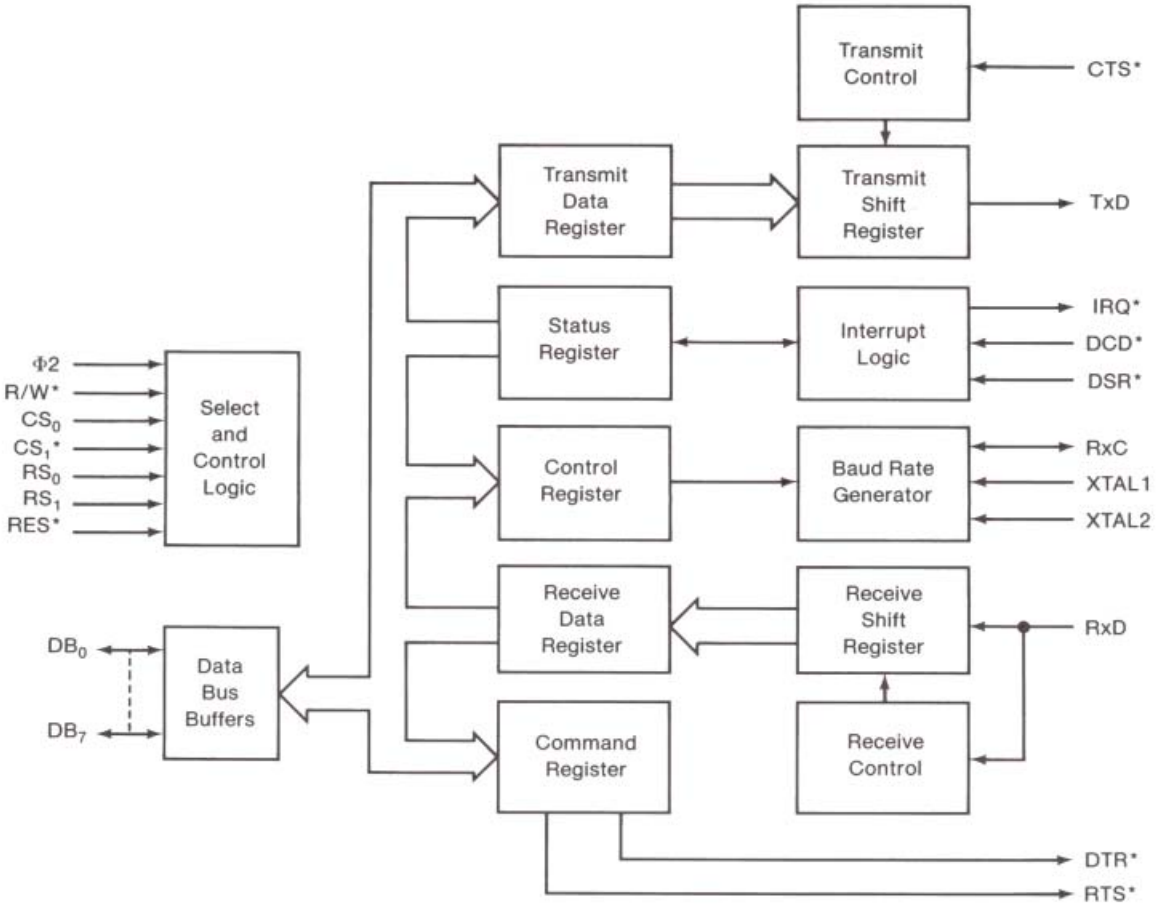


Figure 11-28. *6551 ACIA Block Diagram. Copyright 1978, Synertek Inc. Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.*



The 6551 pin assignments are shown in Figure 11-29 and described in Table 11-19. Note that the two 6551s 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 +5 V through a 3.3 Kohm pullup resistor.

Figure 11-29. The 6551 Pinouts

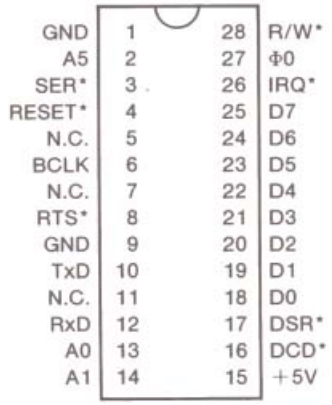
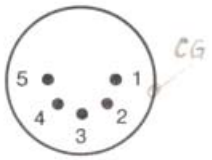


Table 11-19. The 6551 Signal Descriptions

Pin Number	Name	Description
1	GND	Power and signal common ground
2	A4	Address line 4 to select serial port 1
	A5	Address line 5 to select serial port 2
3	SER*	Serial device select from GLU
4	RESET*	Resets both serial ports
5	-	No connection
6	BCLK	Baud rate clock from GLU
7	-	No connection
8	RTS*	Request to Send output
9	CTS*	Clear to Send input
10	TXD	Transmit Data output
11	-	No connection
12	RXD	Receive Data input
13,14	A0,A1	Address lines 0 and 1
15	+5 V	+5 volt supply
16	DSR	Data Set Ready input
17	EXTINT*	External interrupt (port 1 ACIA)
	KSTRB	Keyboard strobe input (port 2 ACIA; Appendix E)
18-25	D0-D7	Eight-bit data bus
26	IRQ*	Interrupt Request input
27	PH0	Phase 0 clock pulse
28	R/W*	Read/write select input

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.

Figure 11-30. *Serial Port Connectors*



Pin	Port 1	Port 2
1	DTR1B	DTR2B
2	TD1B	TD2B
3	GND	GND
4	RD1B	RD2B
5	DSR1B	DSR2B

Table 11-20. *Serial Port Connector Signals*

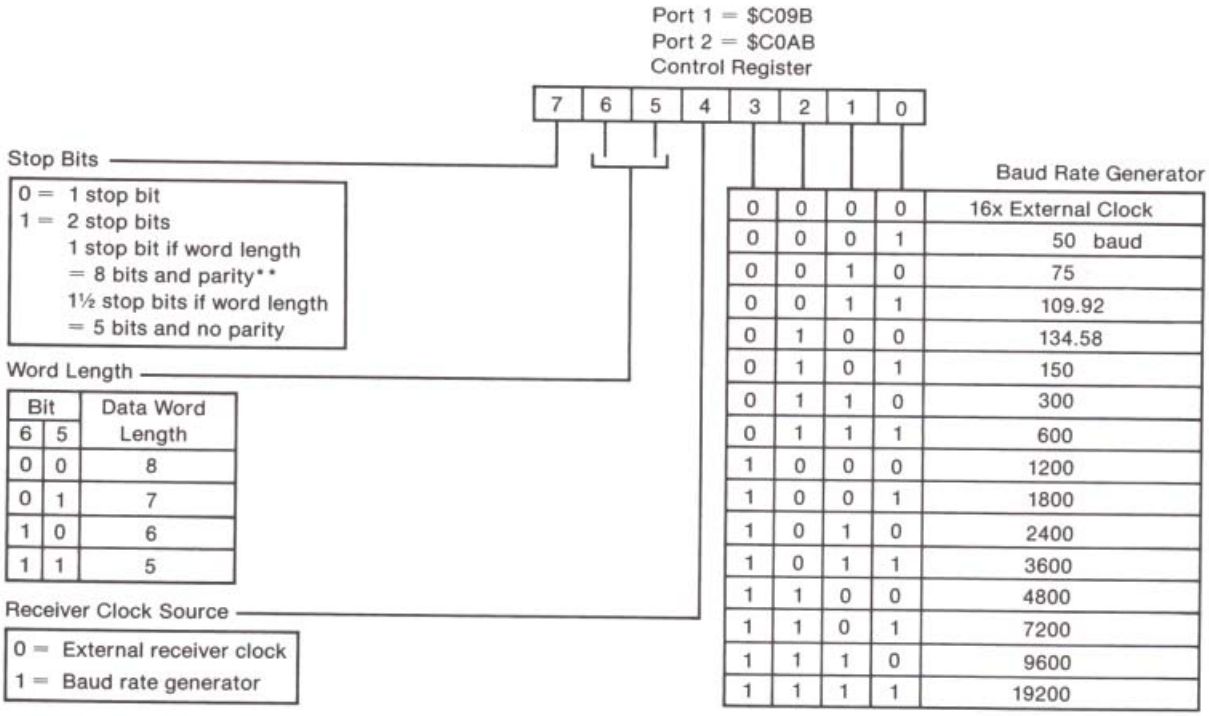
Pin Number	Name	Description
1	DTR1B	Data Terminal Ready output
	DTR2B	
2	TD1B	Transmit Data output
	TD2B	
3	GND	Power and signal common
4	RD1B	Read Data input
	RD2B	
5	DSR1B	Data Set Ready input
	DSR2B	

11.11.1 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 2% lower than the nominal rate. (The EIA standard allows plus or minus 2% variation.) If *an* Apple IIc serial port is used with a modem that is 2% above the nominal rate, framing errors can occur, especially at 1200 baud and above, when using eight 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, Synertek Inc.*
Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.



Bit 6	Bit 5	Data Word Length
0	0	8
0	1	7
1	0	6
1	1	5

3	2	1	0	Baud Rate Generator
0	0	0	0	16x External Clock
0	0	0	1	50 baud
0	0	1	0	75
0	0	1	1	109.92
0	1	0	0	134.58
0	1	0	1	150
0	1	1	0	300
0	1	1	1	600
1	0	0	0	1200
1	0	0	1	1800
1	0	1	0	2400
1	0	1	1	3600
1	1	0	0	4800
1	1	0	1	7200
1	1	1	0	9600
1	1	1	1	19200

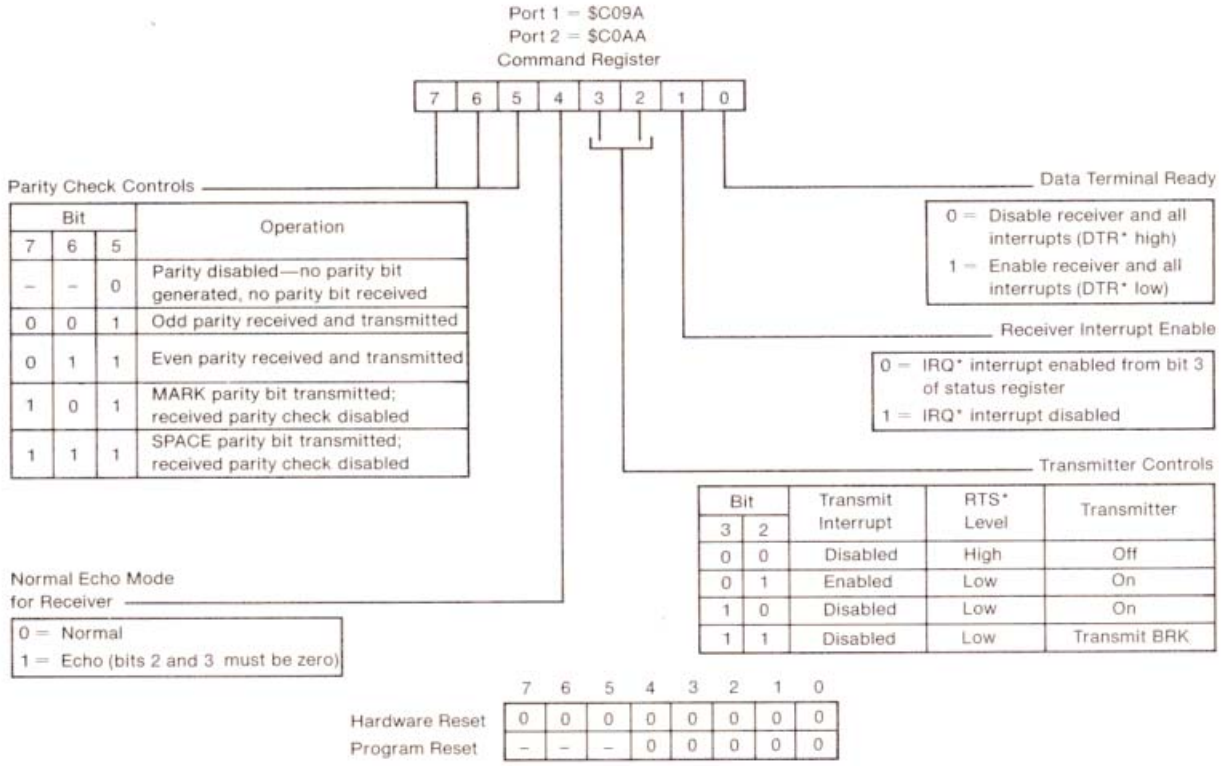
**This allows for 9-bit transmission (8 data plus parity).

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	–	–	–	–	–	–	–	–

11.11.2 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.

Figure 11-32. *ACIA Command Register. Copyright 1978, Synertek Inc. Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.*



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

11.11.3 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.

Figure 11-33. *ACIA Status Register. Copyright 1978, Synertek Inc. Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.*

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	–	–	–	–	–	–	–	–

11.11.4 ACIA Transmit/Receive Register

Each ACIA uses the same address—$C098 for serial port 1, $C0A8 for serial port 2—as temporary data 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.

11.12 Mouse Input

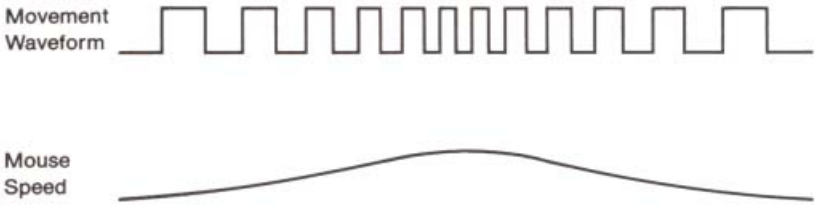
The mouse is a hand-held X-Y pointing device that can be rolled along a flat surface. It has an attached pushbutton. This section describes how mouse movement and direction can be detected and interpreted.

A mouse has a ball inside its housing that protrudes a small distance so that its turning corresponds to mouse movements across a tabletop. 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).

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.

Figure 11-34. Sample Mouse Waveform



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).

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 11-35. Mouse Movement and Direction Waveforms

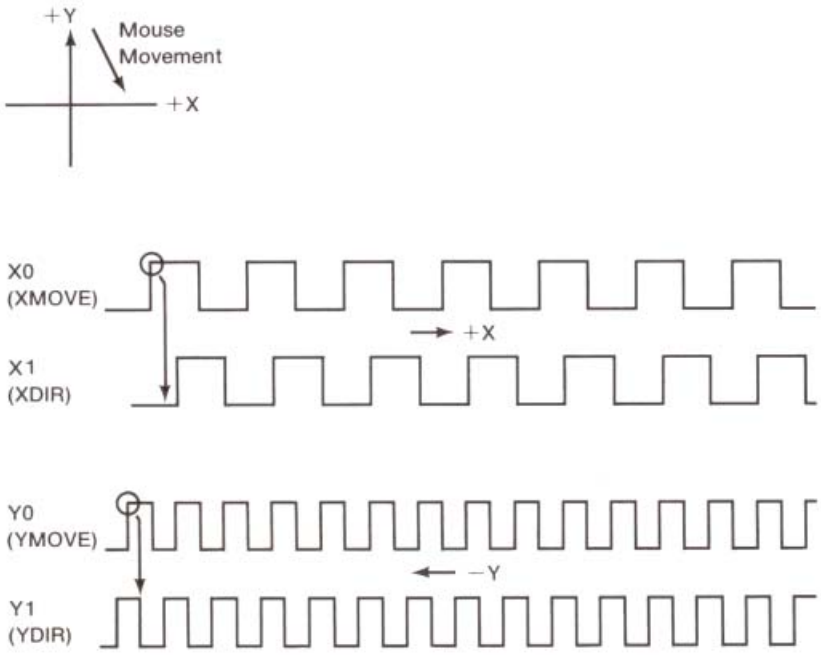
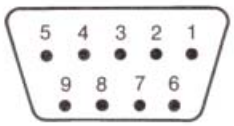


Figure 11-36 shows the pin assignments for the mouse DB-9 connector on the back panel. Table 11-21 gives the signal names and descriptions.

Figure 11-36. *Mouse Connector*



Pin	Signal
1	MOUSEID*
2	+5V
3	GND
4	XDIR
5	XMOVE
6	(N.C.)
7	MSW*
8	YDIR
9	YMOVE

Table 11-21. *Mouse Connector*

DB-9 Pin Number	Signal Name	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	X1	Mouse X-direction indicator
5	X0	Mouse X-movement interrupt
6		Mouse button
7	MSW*	Mouse button
8	Y1	Mouse Y-direction indicator
9	Y0	Mouse Y movement interrupt

Figure 11-37 shows the mouse and hand control circuitry with the mouse circuits highlighted. 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*

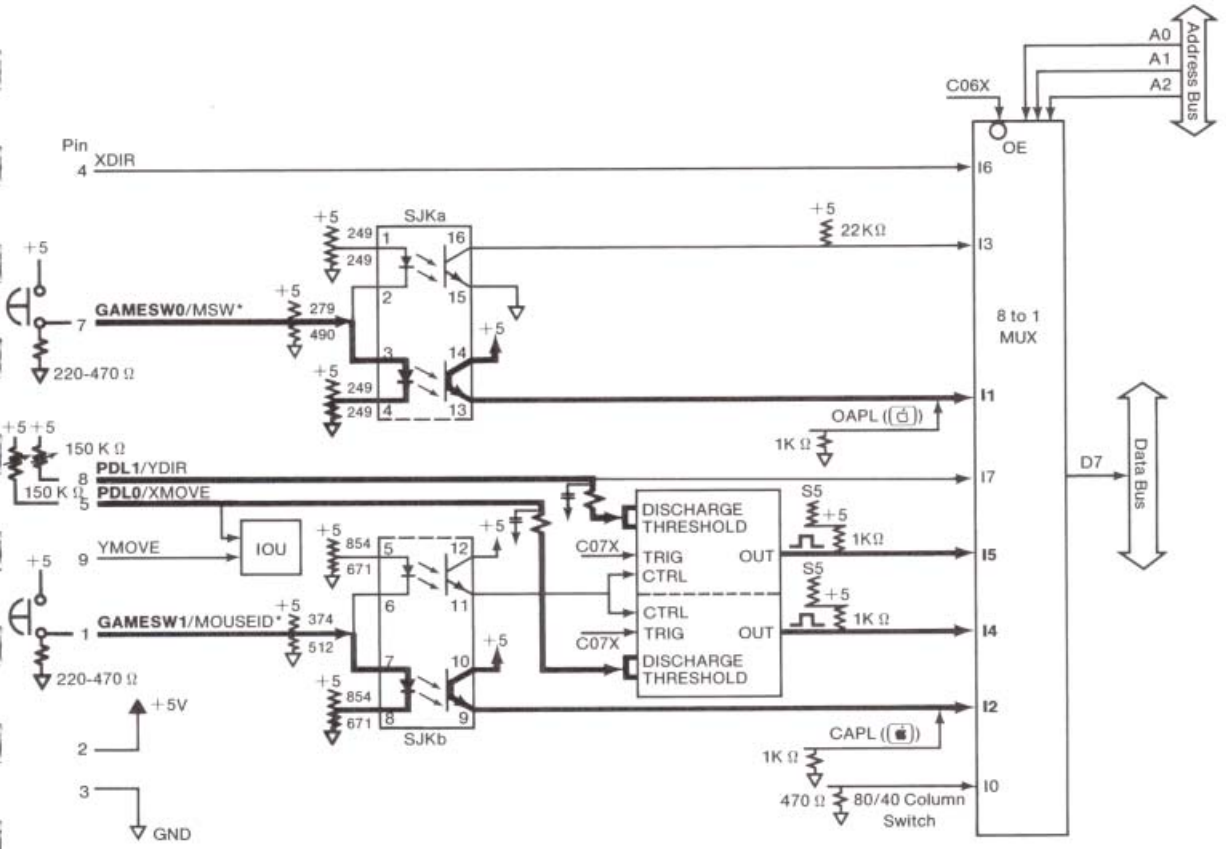
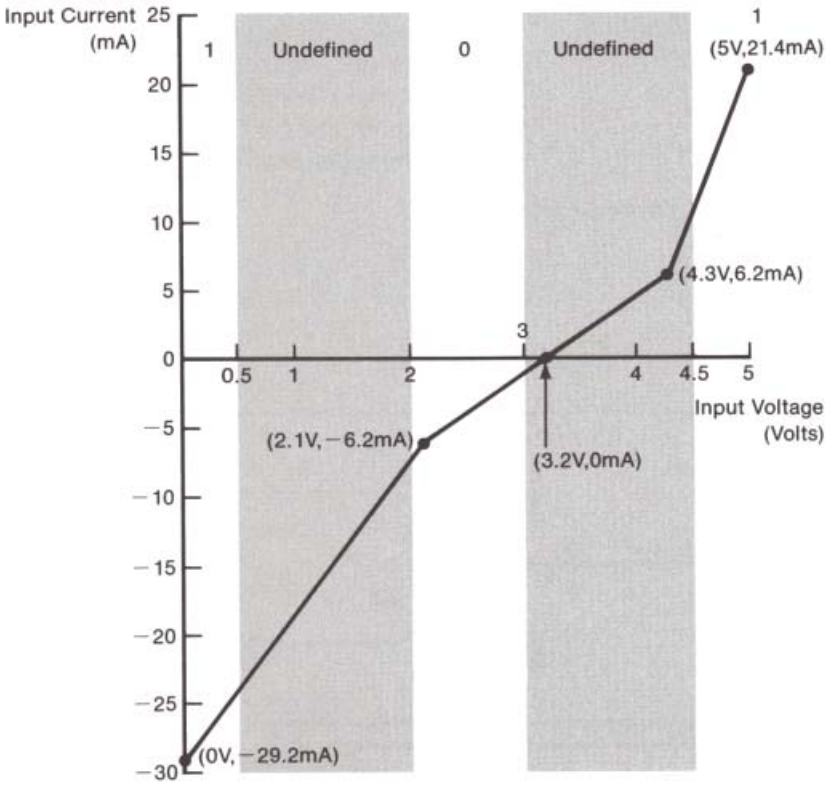
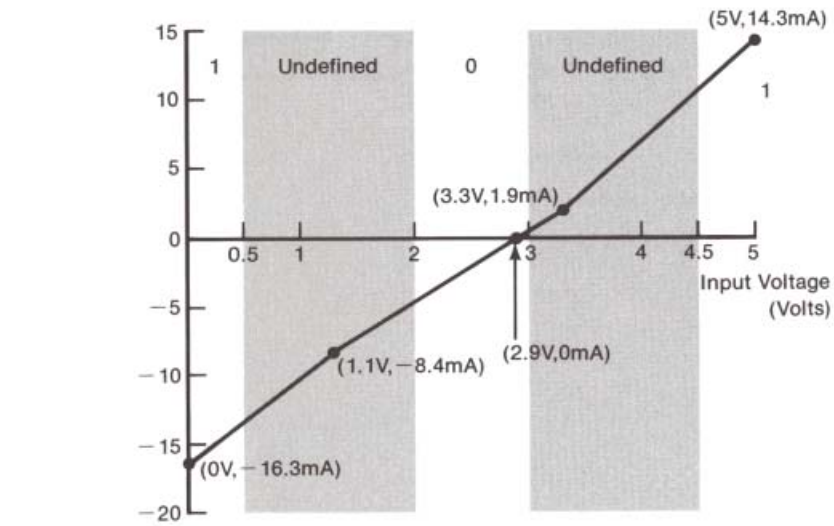


Figure 11-38. Mouse Button Signals





11.13 Hand Controller Input

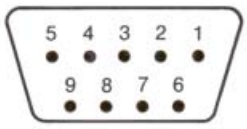
Several input signals that are individually controlled via soft switches are collectively referred to as the **hand controller (game) signals**. These signals arrive in the Apple IIc via the same DB-9 connector as the one used for the mouse (section 11.12), but the Apple IIc interprets these signals differently.

The DB-9 connector pin assignments and signal descriptions, as used for hand control input, appear in Figure 11-39 and Table 11-22.

Even though they are normally used for hand controls, these signals can be used for other simple I/O applications. There are two one-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 one-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 control circuitry with the hand control circuits highlighted. Figure 11-42 illustrates the values of the hand-control switch inputs when the switch is open or closed.

Figure 11-39. *Hand Controller Connector*



Pin	Signal
1	GAMESW1
2	+5V
3	GND
4	Not used for hand controls
5	PDL0
6	(N.C.)
7	GAMESW0
8	PDL1
9	Not used for hand controls

Table 11-22. Hand Control Connector Signals

Connector Pin Number	Signal Name	Description
1	GAMESW1	Switch input 1 (sometimes called **paddle button 1**)
2	+5V	+5 V power supply. Total current drain from this pin must not exceed 100 mA.
3	GND	System ground
4,9	-	Not used for hand controls
5,8	PDL0 and PDL1	Hand control inputs. Each of these must be connected to a 150 Kohm 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

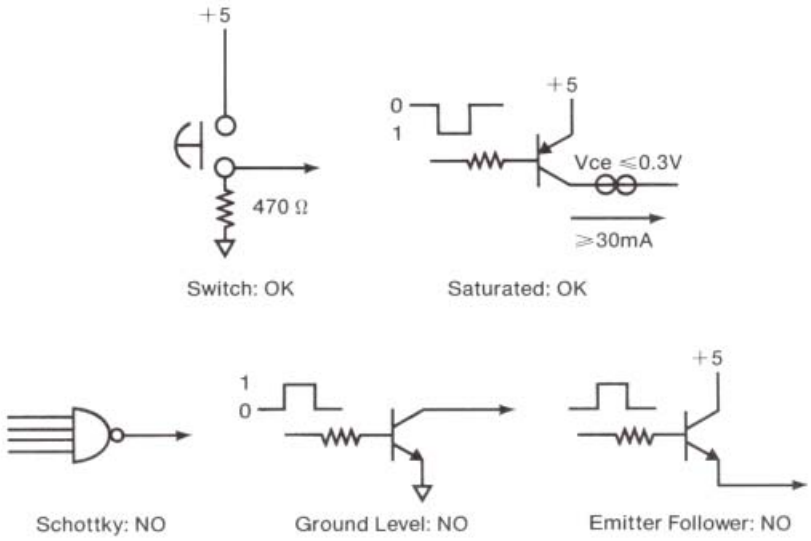


Figure 11-41. Hand Control Circuits

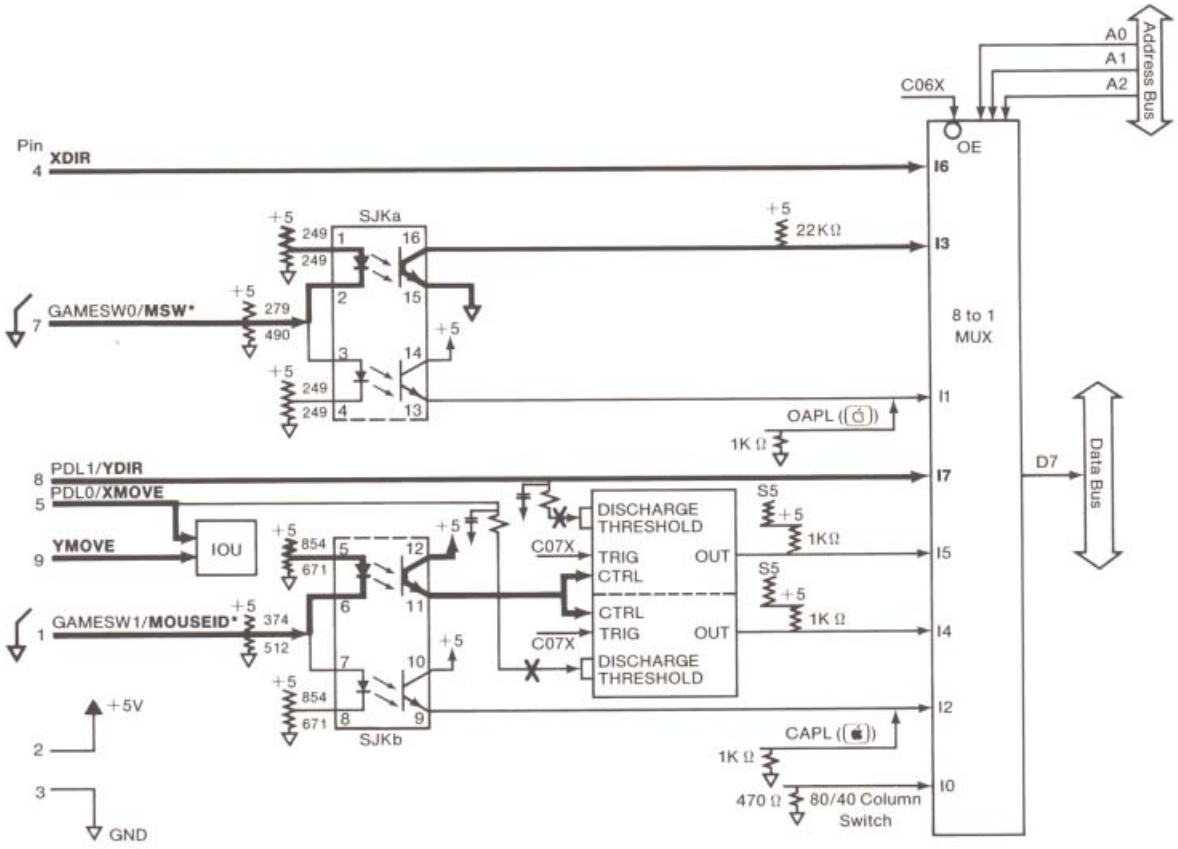
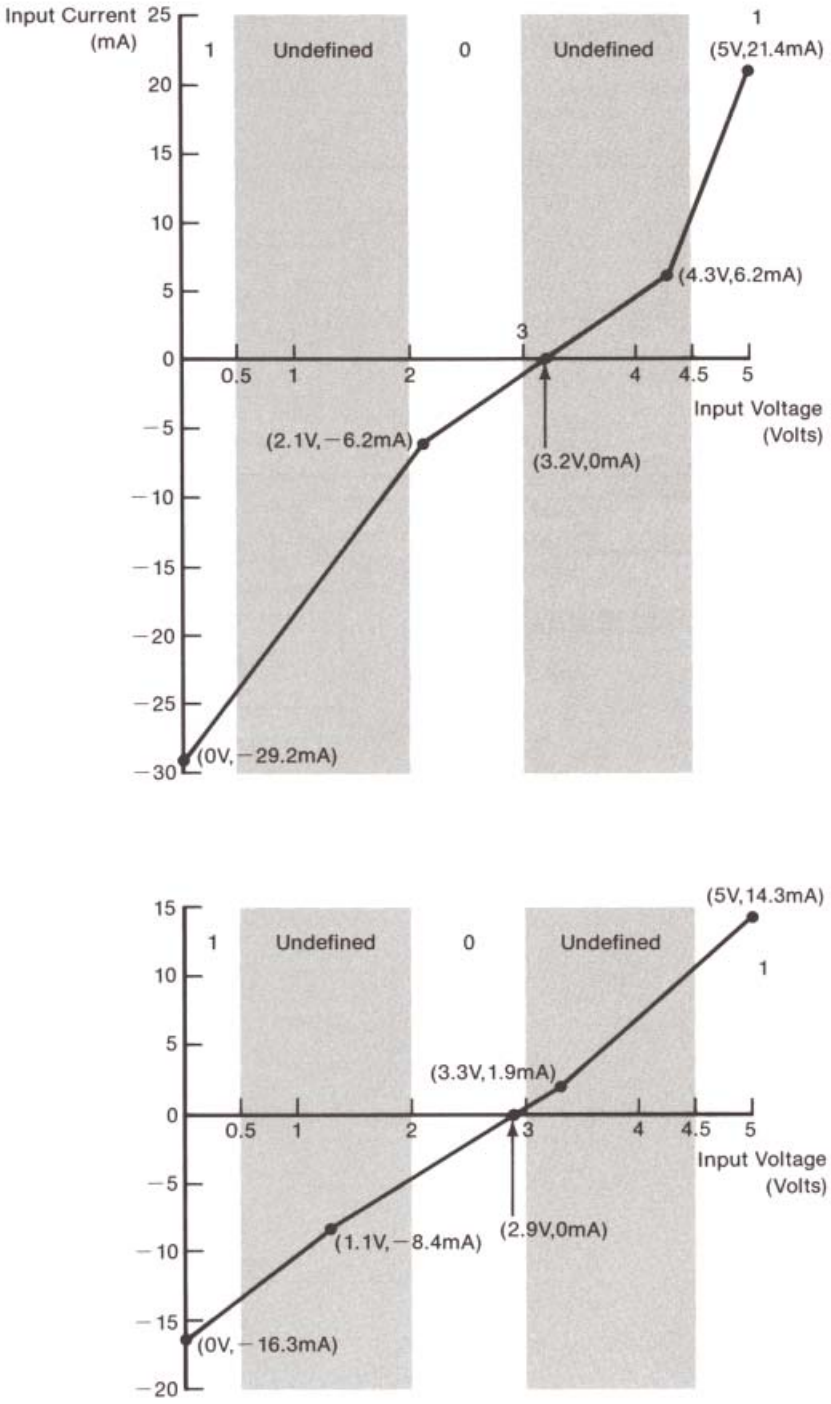


Figure 11-42. *Hand Control Signals*



The hand-control 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 Kohms connected between one of these inputs and the +5 V supply controls the charging time of one of the two 0.022 microfarad capacitors.

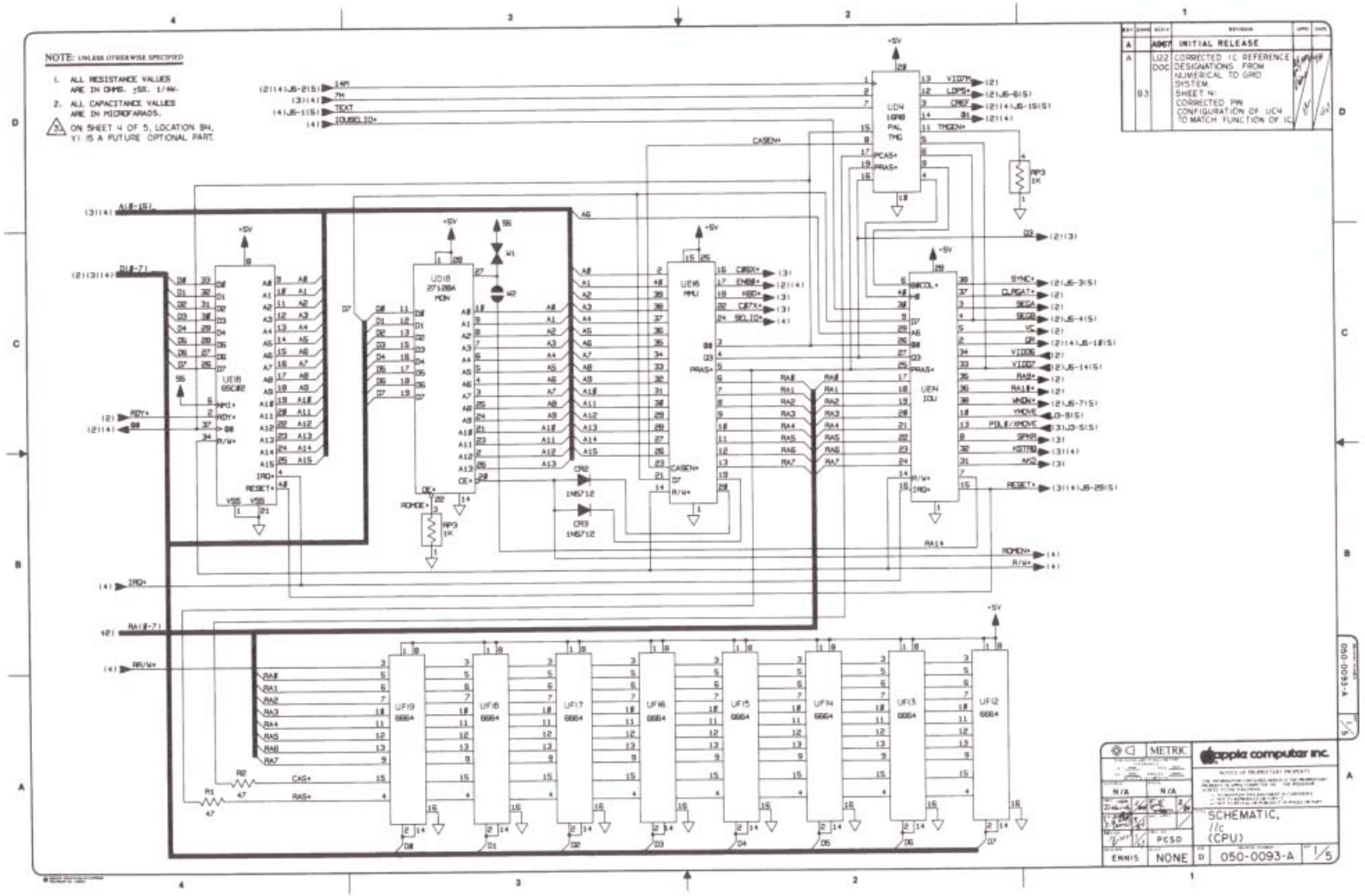
When the voltage on the capacitor passes a certain threshold, the output of the NE556 changes back to 0 (low). Programs can determine the setting of a variable resistor by resetting the timers and then counting time until the selected timer input changes from high to low. The resulting count is proportional to the resistance.

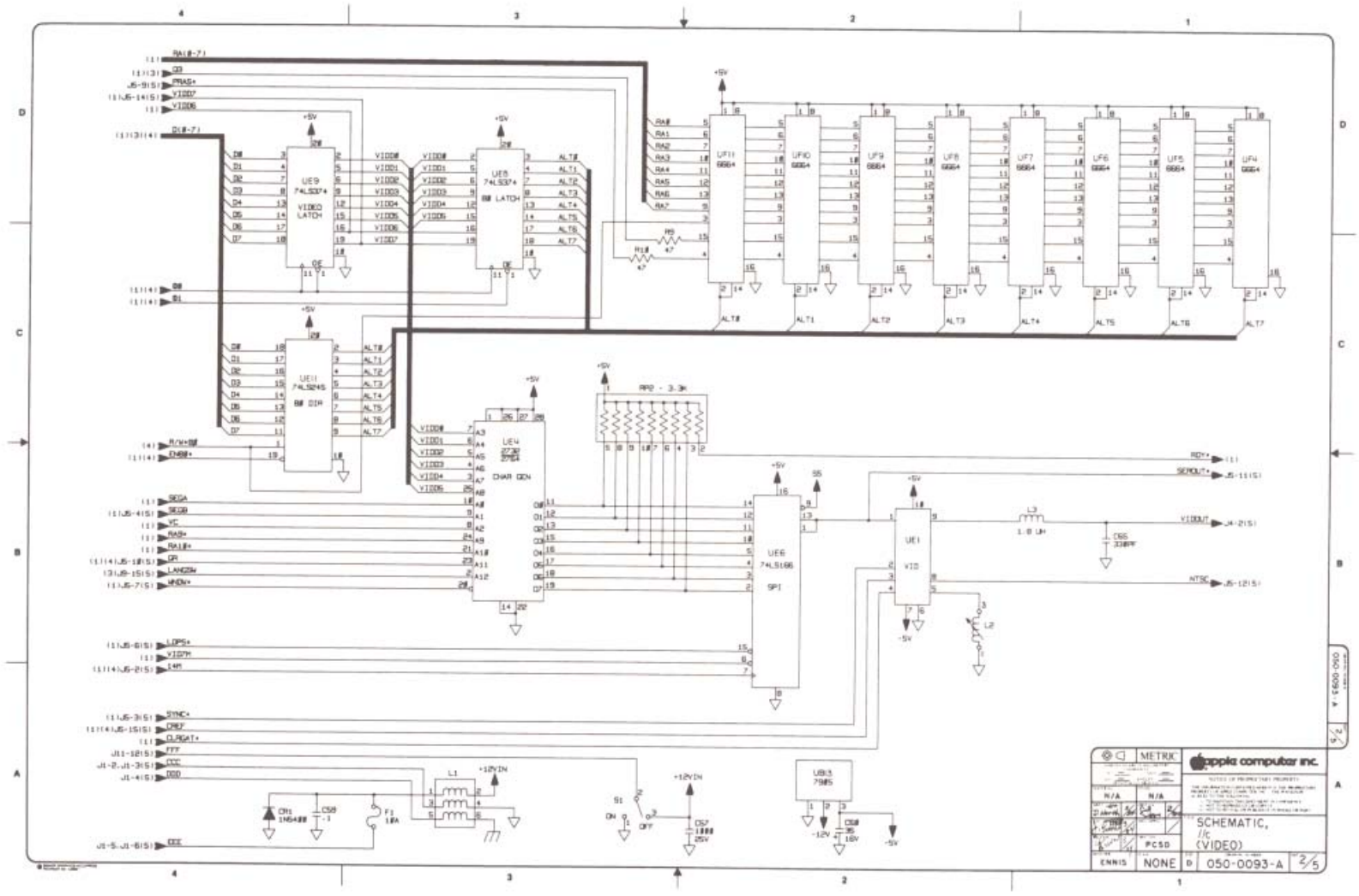
Warning

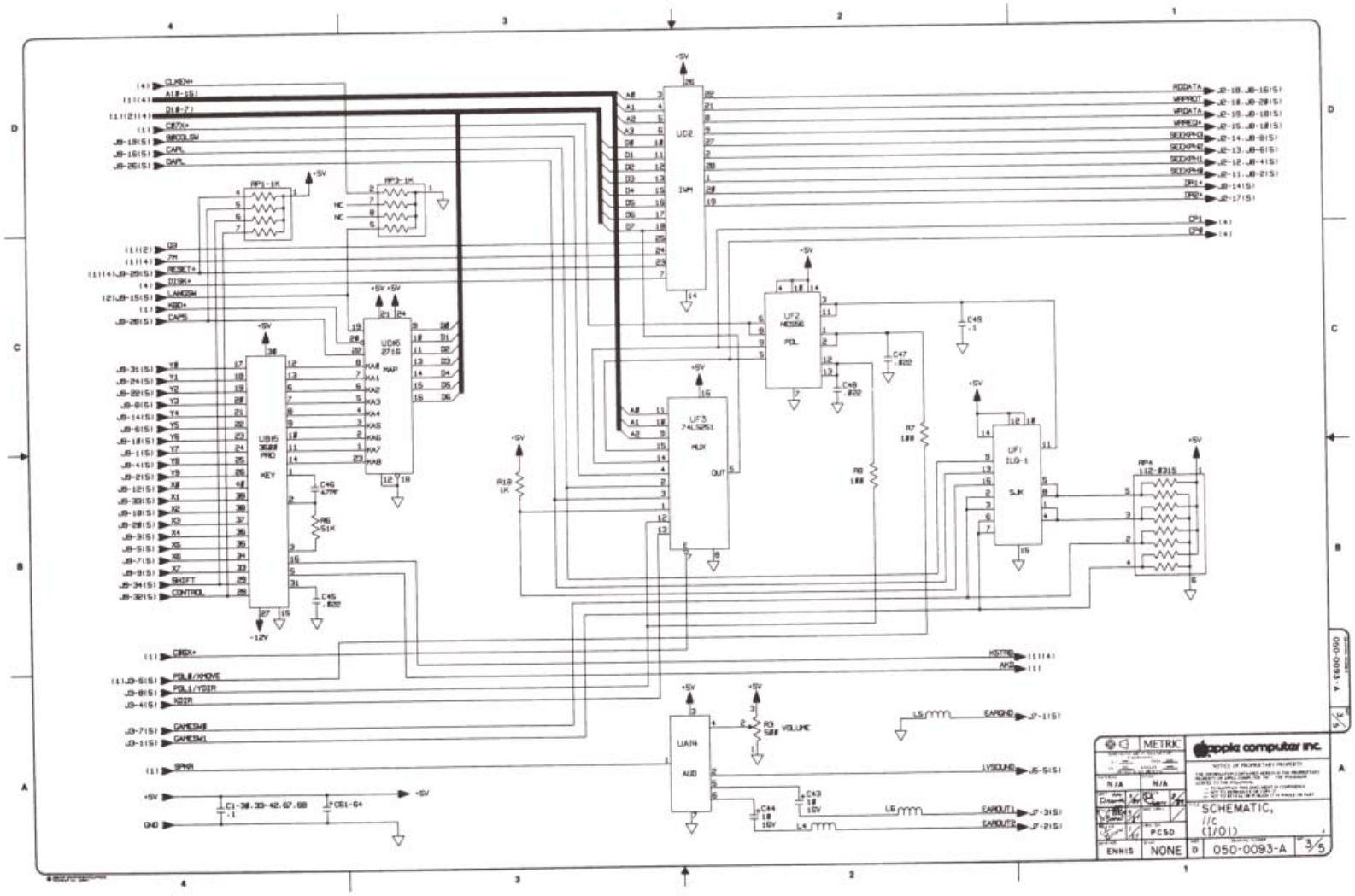
The only way to ensure correct paddle values is to make sure the output of the paddle you intend to read is low before you trigger the timer. Triggering the timer starts the charging cycle for the capacitor in each paddle circuit; the cycle for one may not be completed by the time you have read the other. 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.

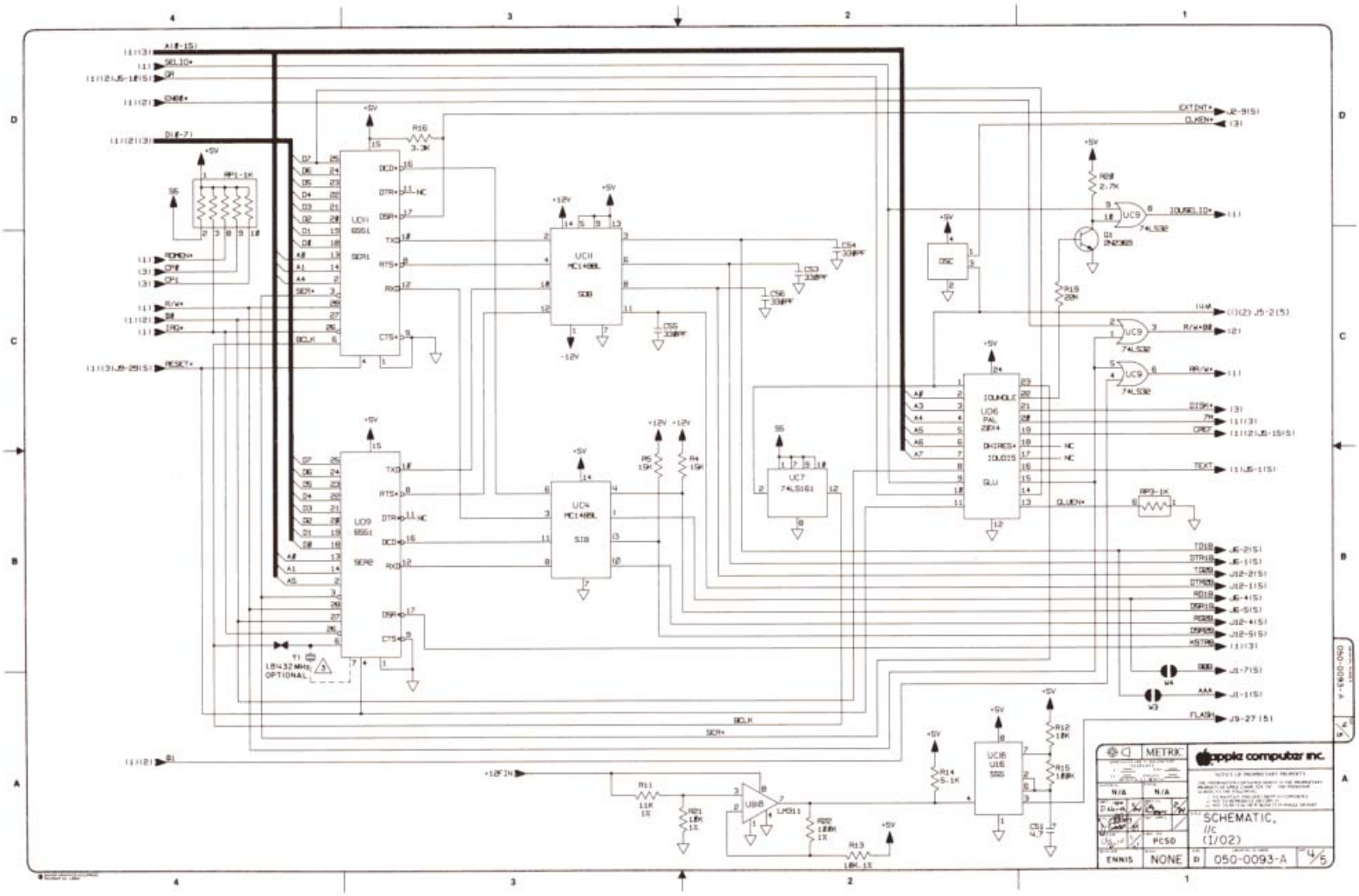
11.14 Schematic Diagrams

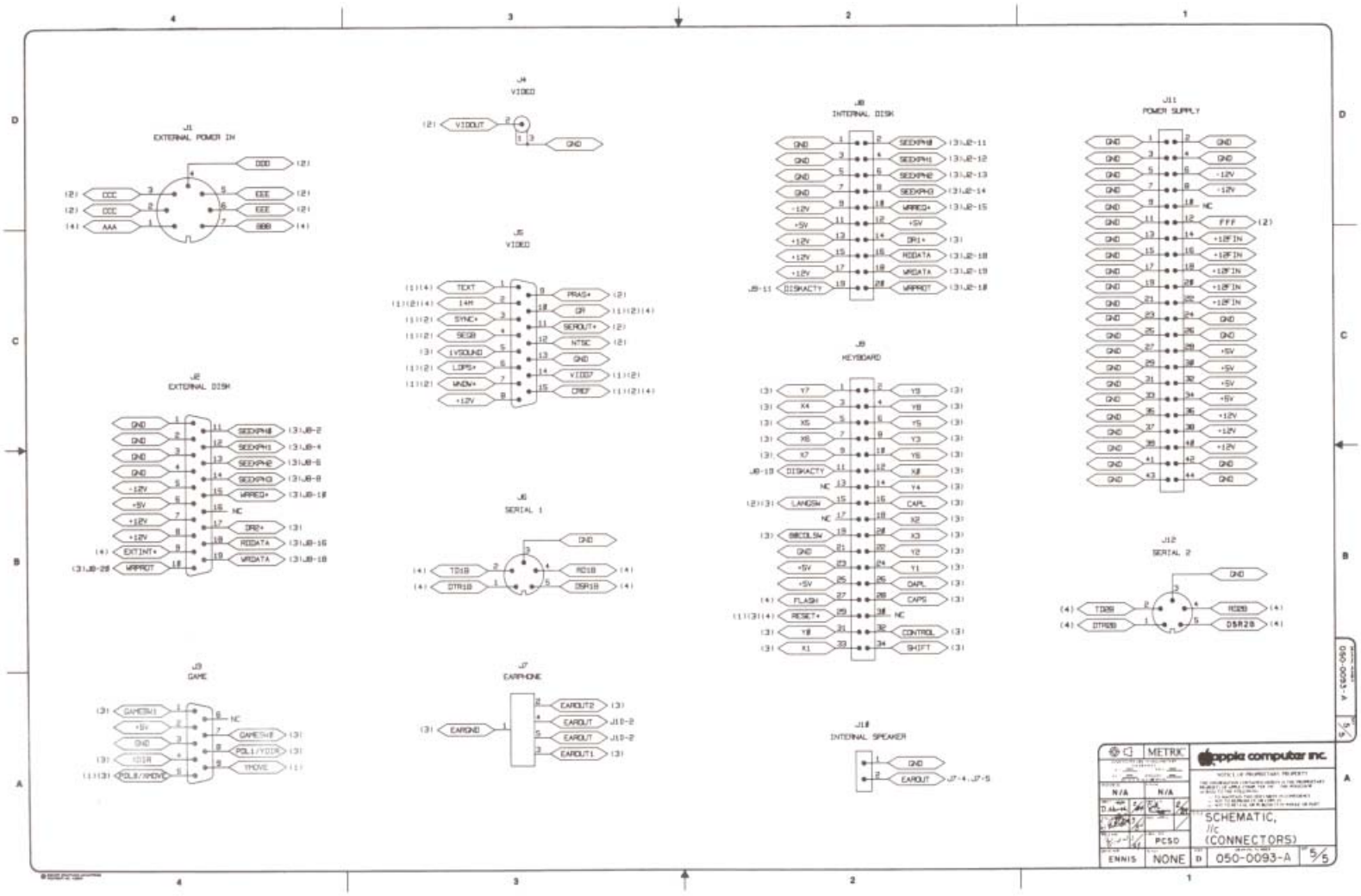
The following pages contain schematic diagrams for the Apple IIc.


NOTE: UNLESS OTHERWISE SPECIFIED
1. ALL RESISTANCE VALUES ARE IN OHMS, ±5%, 1/4W.
2. ALL CAPACITANCE VALUES ARE IN MICROFARADS.
3. ON SHEET 4 OF 5, LOCATION B4, Y1 IS A FUTURE OPTIONAL PART.
A3 A967 INITIAL RELEASE
A A22 DOC CORRECTED IC REFERENCE DESIGNATIONS FROM NUMERICAL TO GRID SYSTEM.
B3 SHEET 4: CORRECTED PIN CONFIGURATION OF UC4 TO MATCH FUNCTION OF IC
UC4 16R8 PAL TMG
UE8 6502
UD8 27128A ROM
UE6 MMU
UE4 IOU
UF19 UF18 UF17 UF16 UF15 UF14 UF13 UF12 6664
METRIC
apple computer inc.
SCHEMATIC, IIc (CPU)
ENNIS
NONE
D
050-0093-A
1/5


SCHEMATIC,
/c
(VIDEO)
apple computer inc.
METRIC
PCSD
NONE
D
050-0093-A
2/5
ENNIS


apple computer inc.
SCHEMATIC, //c (1/01)
050-0093-A
3/5
ENNIS
NONE
D
PCSD
N/A
METRIC


SCHEMATIC, IIc (1/02)
apple computer inc.
050-0093-A
PCSD
ENNIS
NONE
METRIC


J1
EXTERNAL POWER IN
J2
EXTERNAL DISK
J3
GAME
J4
VIDEO
J5
VIDEO
J6
SERIAL 1
J7
EARPHONE
J8
INTERNAL DISK
J9
KEYBOARD
J10
INTERNAL SPEAKER
J11
POWER SUPPLY
J12
SERIAL 2
apple computer inc.
SCHEMATIC, I/c (CONNECTORS)
050-0093-A
5/5
ENNIS
NONE
PCSD

Index

References to entries in Volume 2 are in square brackets [ ].

Cast of Characters

A

D

E

F

J

K

L

M

Q

R

S

T

U

V

W

X

Y

Z

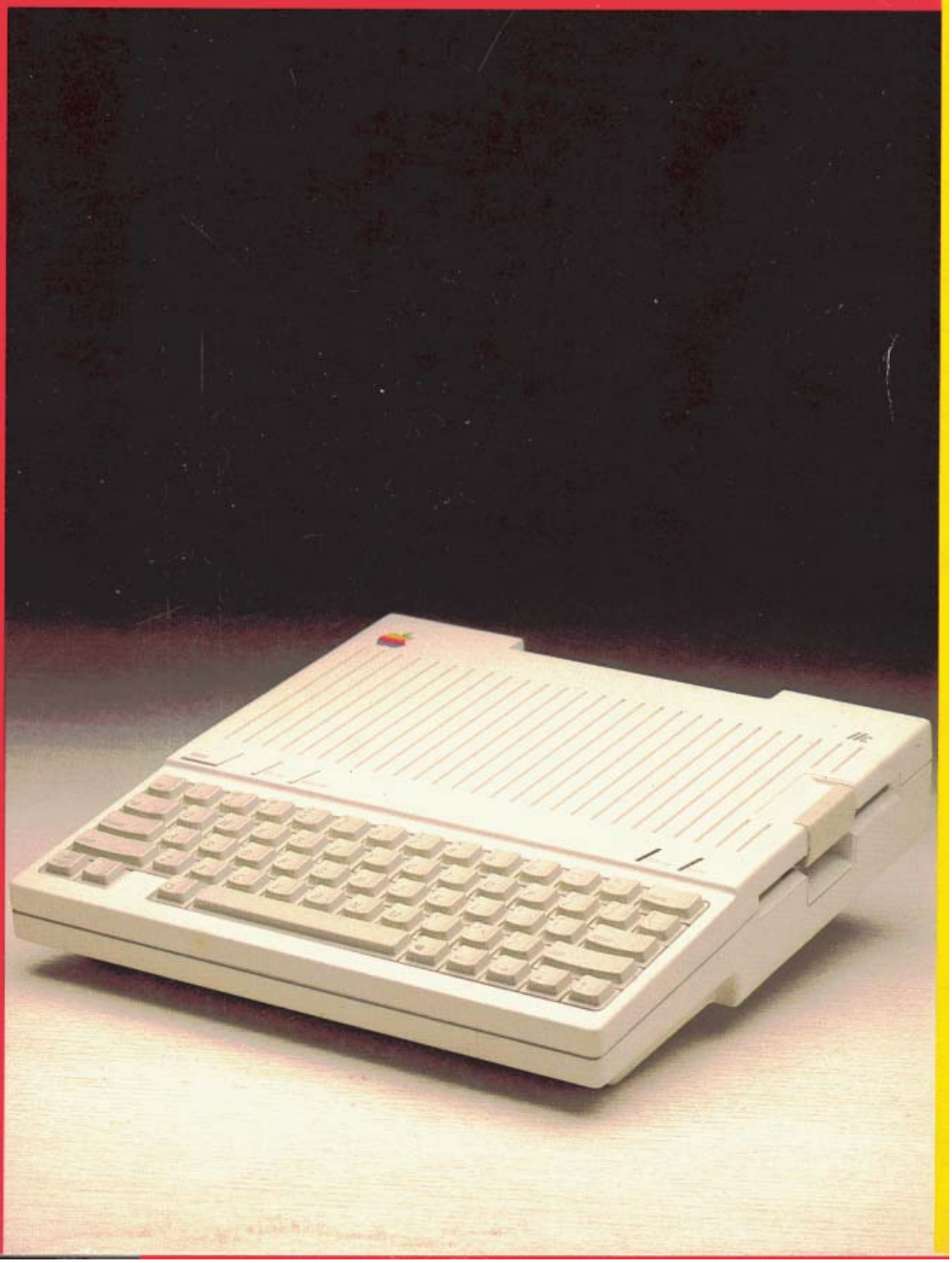
Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576

030-0814-A

The Apple IIc

The Apple IIc Reference Manual

Volume 2



Customer Satisfaction

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the documentation or media at no charge to you during the 90-day period after you purchased the product.

In addition, if Apple releases a corrective update to a software product during the 90-day period after you purchased the software, Apple will replace the applicable disks and documentation with the revised version at no charge to you during the six months after the date of purchase.

In some countries the replacement period may be different; check with your authorized Apple dealer. Return any item to be replaced with proof of purchase to Apple or an authorized Apple dealer.

Limitation on Warranties and Liability

Even though Apple has tested the software described in the manual and reviewed its contents, neither Apple nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in this manual, their quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is," and you the purchaser are assuming the entire risk as to their quality and performance. In no event will Apple or its software suppliers be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering or reproducing these programs or data. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Copyright

Product Revisions

Apple cannot guarantee that you will receive notice of a revision to the software described in the manual, even if you have returned a registration card received with the product. You should periodically check with your authorized Apple dealer.

Warning

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

Reorder Apple Product #A2L4030

The Apple IIc

Apple IIc Reference Manual
Volume 2

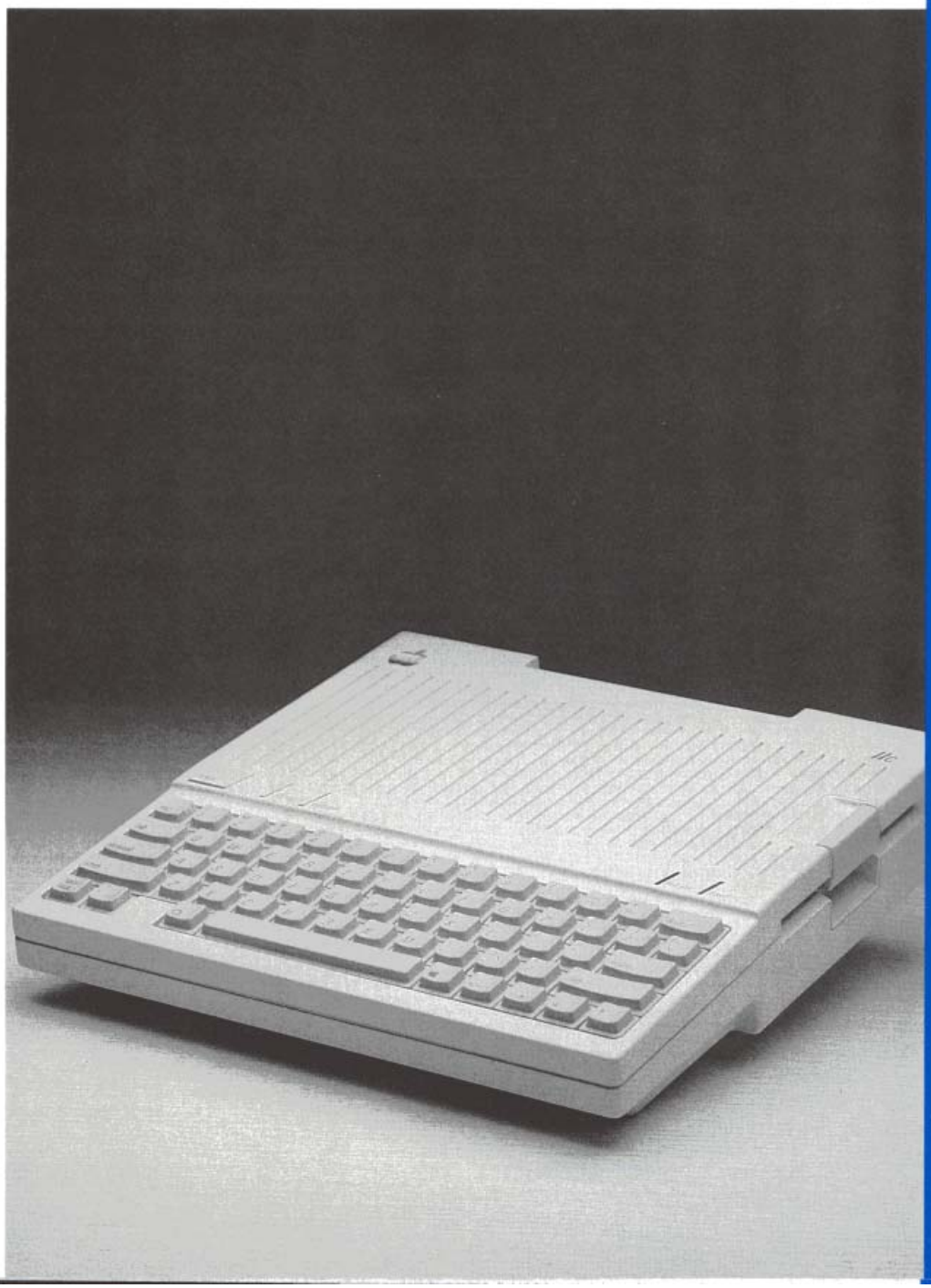


Table of Contents

Appendix F Apple II Series Differences 61

List of Figures and Tables

Appendix A The 65C02 Microprocessor

Appendix B Memory Map

Appendix C Important Firmware Locations

Appendix E Interrupts

Appendix F Apple II Series Differences

Appendix G USA and International Models

Appendix H Conversion Tables

Preface

This volume, Volume 2 of the *Apple IIc Reference Manual*, contains nine appendixes, a bibliography, and a glossary.

Appendix A contains a description of the differences between the 6502 and the 65C02 microprocessors, plus a reprint of the manufacturer's data sheet for the 65C02 microprocessor.

Appendixes B and C contain tables of the important RAM, ROM, and hardware addresses in the Apple IIc. The reader can use these tables to find locations by address, the index to find them by label, the firmware listings to find them as defined and used, and the chapters to find them described in the context of their function.

Appendix B is a memory map of the Apple IIc, including detailed tables of page zero, page three, the screen holes, and the hardware page.

Appendix C lists the *published* firmware entry points, arranged by address, and indicates where in the manual they are described. The list includes I/O firmware (pages $C1 through $CF) and Monitor firmware (pages $F0 through $FF). For Applesoft interpreter firmware (pages $D0 through $EF), refer to the *Applesoft BASIC Programmer's Reference Manual*, Volumes 1 and 2.

Appendix D discusses what operating systems and languages run on the Apple IIc, and what features they do and do not use.

Appendix E describes how to use the Apple IIc's interrupt handling capabilities.

Appendix F contains an overview of the differences among the Apple II series computers.

Appendix G contains the keyboard layouts, code conversion tables, and external power supply characteristics of USA and international models of the Apple IIc.

Appendix H contains reference tables for code and number base conversion.

Appendix I contains a listing of the source code for the Monitor, enhanced video firmware, and input/output firmware contained in the Apple IIc. The listings do not include the built-in Applesoft interpreter, which is discussed in the *Applesoft BASIC Programmer's Reference Manual.*

The Bibliography lists articles and books containing additional information about the Apple IIc and related products.

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

The 65C02 Microprocessor

appendix a

This appendix contains a description of the differences between the 6502 and the 65C02 microprocessor. It also contains the data sheet for the NCR 65C02 microprocessor.

In the data sheet tables, execution times are specified in number of cycles. One cycle time 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 instructions present on the 6502.

A.1 Differences Between 6502 and 65C02

The data sheet lists the new instructions and addressing modes of the 65C02. This section supplements that information by listing the instructions whose execution times or results have changed.

A.1.1 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 instructions whose number of instruction execution cycles on the 65C02 is different from the 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

A.1.2 Differing Instruction Results

It is important to note that 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 contained a zero.

Also note that 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 ($2FF) gets ADL from location $2FF on both processors. But on the 65C02, ADH comes from $300; on the 6502, ADH comes from $200.

A.2 Data Sheet

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.

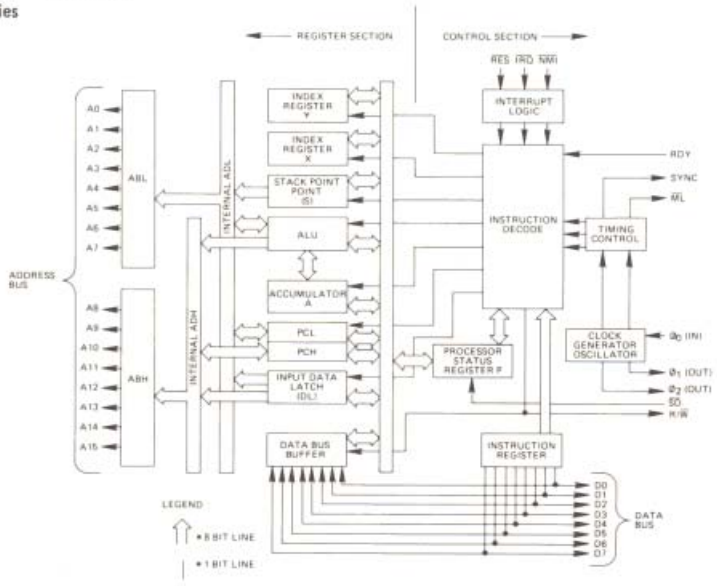
■ PIN CONFIGURATION

■ 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 $Ø_2$ 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, $\overline{IRQ}$, $\overline{SO}$, $\overline{NMI}$ and $\overline{RES}$)

* Specifications are subject to change without notice.

■ NCR65C02 BLOCK DIAGRAM



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 (V_{IN} = 0, T_A = 25°C, f = 1MHz)	C				pF
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

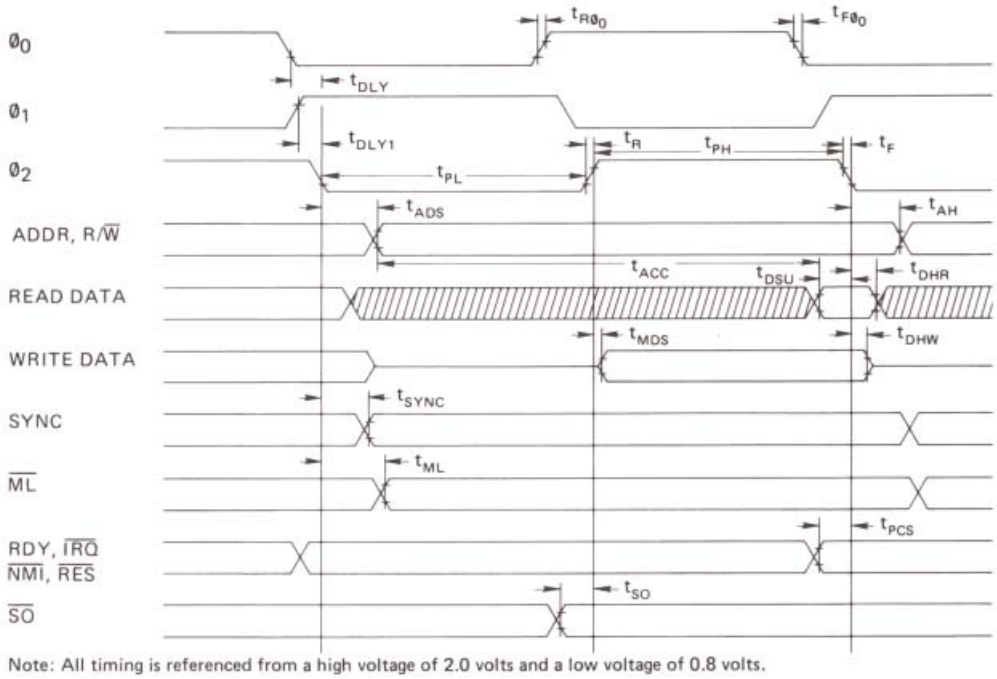
Function	NMOS 6502 Microprocessor	NCR65C02 Microprocessor
Indexed addressing across page boundary.	Extra read of invalid address.	Extra read of last instruction byte.
Execution of invalid op codes.	Some terminate only by reset. Results are undefined.	All are NOPs (reserved for future use). Op Code — 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.

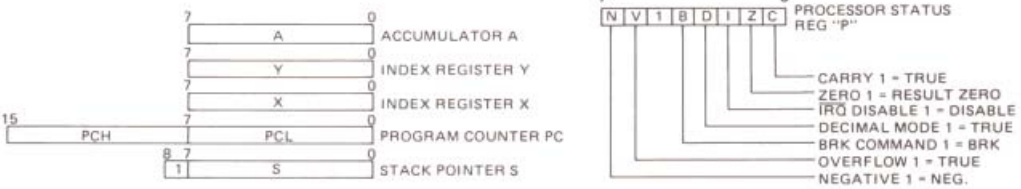
▪ NEW INSTRUCTION MNEMONICS

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 ($\emptyset_0$, $\emptyset_1$, and $\emptyset_2$)
$\emptyset_0$ 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 $\emptyset_2$ clock output is in phase with $\emptyset_0$. The $\emptyset_1$ output pin is 180° out of phase with $\emptyset_0$. (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 $\emptyset_2$ operation; if the interrupt flag in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during $\emptyset_1$. 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 $\emptyset_2$; the current instruction is completed and the interrupt sequence begins during $\emptyset_1$. 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 ($\emptyset_1$), will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two ($\emptyset_2$) 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 $\emptyset_1$.

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 $\emptyset_1$ of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the $\emptyset_1$ 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

		IMME-DIATE			ABSO-LUTE			ZERO PAGE			ACCUM			IM-PLIED			(IND, X)			(IND), Y			ZPG, X			ZPG, Y			ABS, X			ABS, Y			RELA-TIVE			(ABS)			ABS (IND, X)			(ZPG)			PROCESSOR STATUS CODES		
																																													7 6 5 4 3 2 1 0		
MNE	OPERATION	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	N V . B D I Z C		MNE
ADC	A + M + C → A	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2				7D	4	3	79	4	3																ADC
AND	A ∧ M → A	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2				3D	4	3	39	4	3																AND
ASL	C ← 7 ← 0 ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2				1E	6	3																			ASL
BCC	Branch if C=0																																	90	2	2													BCC
BCS	Branch if C=1																																	B0	2	2													BCS
BEQ	Branch if Z=1																																	F0	2	2													BEQ
BIT	A ∧ M	89	2	2	2C	4	3	24	3	2													34	4	2				3C	4	3																			BIT
BMI	Branch if N=1																																	30	2	2													BMI
BNE	Branch if Z=0																																	D0	2	2													BNE
BPL	Branch if N=0																																	10	2	2													BPL
BRA	Branch Always																																	80	2	2													BRA
BRK	Break													00	7	1																																1 . 1 .		BRK
BVC	Branch if V=0																																	50	2	2													BVC
BVS	Branch if V=1																																	70	2	2													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	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2				DD	4	3	D9	4	3																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				CE	6	3	C6	5	2													D6	6	2				DE	6	3																			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																EOR
INA	A + 1 → A										1A	2	1																																	N . . . . . Z .		INA
INC	M + 1 → M				EE	6	3	E6	5	2													F6	6	2				FE	6	3																			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	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2				BD	4	3	B9	4	3																LDA
LDX	M → X	A2	2	2	AE	4	3	A6	3	2																B6	4	2				BE	4	3															LDX
LDY	M → Y	A0	2	2	AC	4	3	A4	3	2													B4	4	2				BC	4	3																			LDY
LSR	0 → 7 → 0 → C				4E	6	3	46	5	2	4A	2	1										56	6	2				5E	6	3																			LSR
NOP	PC + 1 → PC													EA	2	1																																		NOP
ORA	A ∨ M → A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2				1D	4	3	19	4	3																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 ←				2E	6	3	26	5	2	2A	2	1										36	6	2				3E	6	3																			ROL
ROR	→ C → 7 → 0 →				6E	6	3	66	5	2	6A	2	1										76	6	2				7E	6	3																			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	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2				FD	4	3	F9	4	3																SBC
SEC	1 → C													38	2	1																																		SEC
SED	1 → D													F8	2	1																																		SED
SEI	1 → I													78	2	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																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				1C	6	3	14	5	2																																					. . . . . Z .		TRB
TSB	A ∨ M → M				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
V Or
⊻ Exclusive or

n No. Cycles
No. Bytes
M_6 Memory bit 6
M_7 Memory bit 7

Memory Map

This appendix lists all important RAM and hardware locations in address order and describes them briefly. It also provides cross-references to the section of the manual where they are described further. 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 32767) the complementary decimal value for use in Apple Integer BASIC.

Appendix H explains the general rules and tables for converting numbers from one of these forms to another. For memory map diagrams, refer to Chapter 2. Figure 2-2 is an overall memory map, Figure 2-3 is a map of bank-switched memory, and Figure 2-11 is a map of the 48K memory space.

B.1 Page Zero

For Monitor zero page usage, refer to the firmware listings. For zero page use by the languages and operating systems, refer to the appropriate reference manuals.

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.
- 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. Zero Page Use

Hex	Dec	Used by
$00	0	A
$01	1	A
$02	2	A
$03	3	A
$04	4	A
$05	5	A
$06	6	
$07	7	
$08	8	
$09	9	
$0A	10	A
$0B	11	A
$0C	12	A
$0D	13	A
$0E	14	A
$0F	15	A
$10	16	A
$11	17	A
$12	18	A
$13	19	A
$14	20	A
$15	21	A
$16	22	A
$17	23	A
$18	24	A
$19	25	
$1A	26	
$1B	27	
$1C	28	
$1D	29	
$1E	30	
$1F	31	
$20	32	M
$21	33	M
$22	34	M
$23	35	M
$24	36	M
$25	37	M
$26	38	M D
$27	39	M D
$28	40	M D
$29	41	M D
$2A	42	M D
$2B	43	M D
$2C	44	M D
$2D	45	M D
$2E	46	M D
$2F	47	M D
$30	48	M
$31	49	M
$32	50	M
$33	51	M
$34	52	M
$35	53	M D
$36	54	M D
$37	55	M D
$38	56	M D
$39	57	M D
$3A	58	M P
$3B	59	M P
$3C	60	M P
$3D	61	M P
$3E	62	M D P
$3F	63	M D P
$40	64	M D (P)
$41	65	M D (P)
$42	66	M D (P)
$43	67	M D (P)
$44	68	M D (P)
$45	69	M D (P)
$46	70	M D (P)
$47	71	M D (P)
$48	72	M D (P)
$49	73	M (P)
$4A	74	I D (P)
$4B	75	I D (P)
$4C	76	I D (P)
$4D	77	I D (P)
$4E	78	M (P)
$4F	79	M
$50	80	M A
$51	81	M A
$52	82	M A
$53	83	M A
$54	84	M A
$55	85	M A I
$56	86	A I
$57	87	A I
$58	88	A I
$59	89	A I
$5A	90	A I
$5B	91	A I
$5C	92	A I
$5D	93	A I
$5E	94	A I
$5F	95	A I

Table B-1—Continued. *Zero Page 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	
$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	

Table B-1—Continued. Zero Page Use

Hex	Dec	Used by			Hex	Dec	Used by
$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	

B.2 Page Three

Most of page 3 is available for small machine-language programs or any other use your program might put it to. The built-in Monitor uses the top sixteen addresses of page 3, as shown in Table B-2; the XFER routine (section 2.5.3) uses locations $3ED and $3EE. If you are using DOS or ProDOS, it also uses the 32 locations $3D0 through $3EF.

Table B-2. *Page 3 Use*

Hex	Dec	Section	Use
$3F0 $3F1	1008 1009	2.6.4	Address of BRK request handler (normally $59, $FA)
$3F2 $3F3 $3F4	1010 1011 1012	2.6.4 & 10.1 2.6.4	Reset vector Power-up byte (see text)
$3F5 $3F6 $3F7	1013 1014 1015		Jump instruction to Applesoft &-command handler (initially $4C, $58, $FF)
$3F8 $3F9 $3FA	1016 1017 1018	10.6.4	Jump instruction to user CONTROL-Y command handler
$3FB $3FC $3FD	1019 1020 1021		Jump instruction to NMI interrupt handler (not used by Apple IIc)
$3FE $3FF	1022 1023	2.6.4	Address of user IRQ interrupt handler

B.3 Screen Holes

One result of the way the Apple IIc hardware maps display memory on the screen is that groups of eight memory addresses are left over in sixteen areas of the text and low-resolution display pages—eight areas in main RAM and eight in auxiliary RAM. The firmware uses for these 128 bytes are shown in Tables B-3 and B-4, with cross-references to the section numbers where they are described.

Table B-3. *Main Memory Screen Hole Allocations*

Hex	Dec	Section	Description
$478	1144	9.1.5	Mouse port: low byte of clamping minimum
$479	1145	7.5	Reserved for serial port 1
$47A	1146	8.5	Reserved for serial port 2
$47B	1147		Reserved
$47C	1148	9.1.5	Low byte of X coordinate
$47D	1149		Reserved for mouse port
$47E	1150		Reserved
$47F	1151		Reserved
$4F8	1272	9.1.5	Mouse port: low byte of clamping maximum
$4F9	1273	7.5,E.6.3	Reserved for serial port 1
$4FA	1274	8.5,E.6.2	Reserved for serial port 2
$4FB	1275		Reserved
$4FC	1276	9.1.5	Low byte of Y coordinate
$4FD	1277		Reserved for mouse port
$4FE	1278		Reserved
$4FF	1279	E.6.4	Reserved
$578	1400	9.1.5	Mouse port: high byte of clamping minimum
$579	1401	7.5	Port 1 printer width (1-255; 0 = unlimited)
$57A	1402	8.5	Port 2 line length (1-255; 0 = unlimited)
$57B	1403		Cursor horizontal position (80-column display)
$57C	1404	9.1.5	High byte of X coordinate
$57D	1405		Reserved for mouse port
$57E	1406		Reserved
$57F	1407	E.6.4	Reserved
$5F8	1528	9.1.5	Mouse port: high byte of clamping maximum
$5F9	1529	7.5,E.6.3	Port 1 temporary storage location
$5FA	1530	8.5,E.6.2	Port 2 temporary storage location
$5FB	1531		Reserved
$5FC	1532	9.1.5	High byte of Y coordinate
$5FD	1533		Reserved for mouse port
$5FE	1534		Reserved
$5FF	1535	E.6.2	Reserved

Table B-3—Continued. *Main Memory Screen Hole Allocations*

Hex	Dec	Section	Description
$678	1656		Reserved
$679	1657	7.5	Indicates when port 1 firmware is parsing a command
$67A	1658	8.5	Indicates when port 2 firmware is parsing a command
$67B	1659		Reserved
$67C	1660	9.1.5	Mouse port: reserved
$67D	1661		Reserved for mouse port
$67E	1662		Reserved
$67F	1663	E.6.4	Reserved
$6F8	1784		Reserved
$6F9	1785	7.5	Current port 1 command character
$6FA	1786	8.5	Current port 2 command character
$6FB	1787		Reserved
$6FC	1788	9.1.5	Mouse port: reserved
$6FD	1789		Reserved for mouse port
$6FE	1790		Reserved
$6FF	1791	E.6.2	Reserved
$778	1912		DEVNO: $n0 = current active port number x 16
$779	1913	7.5	Port 1 flags for echo and auto line feed
$77A	1914	8.5	Port 2 flags for echo and auto line feed
$77B	1915		Reserved
$77C	1916	9.1.5,E.6.1	Mouse port status byte
$77D	1917		Reserved for mouse port
$77E	1918		Reserved
$77F	1919		Reserved
$7F8	2040		MSLOT: owner of $C800-$CFFF ($C3, video)
$7F9	2041	7.5	Port 1 current printer column
$7FA	2042	8.5	Port 2 current line position
$7FB	2043		Reserved
$7FC	2044	9.1.5	Mouse port mode byte
$7FD	2045		Reserved for mouse port
$7FE	2046		Reserved
$7FF	2047		Reserved

Table B-4. Auxiliary Memory Screen Hole Allocations

Hex	Dec	Section	Description
$478	1144	7.5	Initial port 1 ACIA Control Register values ($9E)
$479	1145	7.5	Initial port 1 ACIA Command Register values ($0B)
$47A	1146	7.5	Initial port 1 characteristics flags ($40)
$47B	1147	7.5	Initial port 1 printer width ($50)
$47C	1148	8.5	Initial port 2 ACIA Control Register values ($16)
$47D	1149	8.5	Initial port 2 ACIA Command Register values ($0B)
$47E	1150	8.5	Initial port 2 characteristics flags ($01)
$47F	1151	8.5	Initial port 2 line length ($00)
$4F8 through $4FF	1272 through 1279		Reserved
$578 through $57F	1400 through 1407		Reserved
$5F8 through $5FF	1528 through 1535		Reserved
$678 through $67F	1656 through 1663		Reserved
$6F8 through $6FF	1784 through 1791		Reserved
$778 through $77F	1912 through 1919		Reserved
$7F8 through $7FF	2040 through 2047		Reserved

B.4 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 means the sixteen 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 Through $C03F*

RW	Hex	Dec	Neg Dec	Label	Section	Use
R	$C00x			KSTRB	4.1	Read keyboard data (bits 0-6) and strobe (bit 7)
W	$C000	49152	-16384	80STORE	5.6†	Off: PAGE2 switches Page 1 and 2
W	$C001	49153	-16383	80STORE	5.6†	On: PAGE2 switches Page 1 and 1X
W	$C002	49154	-16382	RAMRD	2.5.2	Off: read main 48K RAM
W	$C003	49155	-16381	RAMRD	2.5.2	On: read auxiliary 48K RAM
W	$C004	49156	-16380	RAMWRT	2.5.2	Off: write in main 48K RAM
W	$C005	49157	-16379	RAMWRT	2.5.2	On: write in auxiliary 48K RAM
W	$C006	49158	-16378			Reserved
W	$C007	49159	-16377			Reserved
W	$C008	49160	-16376	ALTZP	2.4.2	Off: use main P0, P1, bank-switched RAM
W	$C009	49161	-16375	ALTZP	2.4.2	On: use auxiliary P0, P1, bank-switched RAM
W	$C00A	49162	-16374			Reserved
W	$C00B	49163	-16373			Reserved
W	$C00C	49164	-16372	80COL	5.6	Off: 40-column display
W	$C00D	49165	-16371	80COL	5.6	On: 80-column display
W	$C00E	49166	-16270	ALTCHAR	5.6	Off: display primary character set
W	$C00F	49167	-16369	ALTCHAR	5.6	On: display alternate character set
W	$C01x				4.1	Clear keyboard strobe ($C00x bit 7)
R7	$C010	49168	-16368	AKD	4.1	See if any key now down; clear strobe
R7	$C011	49169	-16367	RDBNK2	2.4.2	See if using $D000 bank 2 (or 1)
R7	$C012	49170	-16366	RDLCRAM	2.4.2	See if reading RAM (or ROM).
R7	$C013	49171	-16365	RDRAMRD	2.5.2	See if reading auxiliary 48K RAM (or main)
R7	$C014	49172	-16364	RDRAMWRT	2.5.2	See if writing auxiliary 48K RAM (or main)
R	$C015	49173	-16363	RSTXINT	9.1.3	Reset mouse X0 interrupt.
R7	$C016	49174	-16362	RDALTZP	2.4.2	See if auxiliary P0, P1 and bank-switched RAM
R	$C017	49175	-16361	RSTYINT	9.1.3	Reset mouse Y interrupt
R7	$C018	49176	-16360	RD80STORE	5.6†	See if 80STORE on (or off)
R7	$C019	49177	-16359	RSTVBL	9.1.3	See if VBLINT off (1); reset it
R7	$C01A	49178	-16358	RDTEXT	5.6	See if text (or graphics)
R7	$C01B	49179	-16357	RDMIX	5.6	See if mixed mode switch on
R7	$C01C	49180	-16356	RDPAGE2	5.6†	See if page 2/1X selected (or 1)
R7	$C01D	49181	-16355	RDHIRES	5.6†	See if high-resolution switch on
R7	$C01E	49182	-16354	RDALTCHAR	5.6	See if alternate character set (or primary)
R7	$C01F	49183	-16353	RD80COL	5.6	See if 80-column hardware on
N	$C020 through $C02F	49184 through 49199	-16352 through -16337			Reserved (read and write)
W	$C030	49200	-16336			Reserved
R	$C030	49200	-16336		4.2.1	Toggle speaker
N	$C031 through $C03F	49201 through 49215	-16335 through -16321			Reserved (read and write)

† Also section 2.5.4

Table B-6. *Addresses $C040 Through $C05F*

RW	Hex	Dec	Neg Dec	Label	Section	Use
R7	$C040	49216	-16320	RDXYMSK	9.1.3	See if X0/Y0 mask set
R7	$C041	49217	-16319	RDVBLMSK	9.1.3	See if VBL mask set
R7	$C042	49218	-16318	RDX0EDGE	9.1.3	See if interrupt on falling X0 edge
R7	$C043	49219	-16317	RDY0EDGE	9.1.3	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	9.1.3	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	5.6	Off: graphics display
R/W	$C051	49233	-16303	TEXT	5.6	On: text display
R/W	$C052	49234	-16302	MIXED	5.6	Off: text or graphics only
R/W	$C053	49235	-16301	MIXED	5.6	On: combination text and graphics
R/W	$C054	49236	-16300	PAGE2	5.6†	Off: use page 1
R/W	$C055	49237	-16299	PAGE2	5.6†	On: display page 2 (80STORE off); store to page 1X (80STORE on)
R/W	$C056	49238	-16298	HIRES	5.6†	Off: low-resolution
R/W	$C057	49239	-16297	HIRES	5.6†	On: high-resolution; double if 80COL and DHIRES on
N	$C058	49240	-16296			Reserved if IOUDIS on ($C07E bit 7 = 1)
R/W				DISXY	9.1.3	Disable (mask) mouse X0/Y0 interrupts
N	$C059	49241	-16295			Reserved if IOUDIS on
R/W				ENBXY	9.1.3	Enable (allow) mouse X0/Y0 interrupts
N	$C05A	49242	-16294			Reserved if IOUDIS on
R/W				DISVBL	9.1.3	Disable (mask) VBL interrupts
N	$C05B	49243	-16293			Reserved if IOUDIS on
R/W				ENVBL	9.1.3	Enable (allow) VBL interrupts
N	$C05C	49244	-16292			Reserved if IOUDIS on
R/W				X0EDGE	9.1.3	Interrupt on rising edge of X0
N	$C05D	49245	-16291			Reserved if IOUDIS on
R/W				X0EDGE	9.1.3	Interrupt on falling edge of X0
R/W	$C05E	49246	-16290	DHIRES	5.6	If IOUDIS on: set double-high-resolution
R/W				Y0EDGE	9.1.3	If IOUDIS off: interrupt on rising Y0
R/W	$C05F	49247	-16289	DHIRES	5.6	If IOUDIS on: clear double-high-resolution
R/W				Y0EDGE	9.1.3	If IOUDIS off: interrupt on falling Y0

† Also section 2.5.4.

Table B-7. Addresses $C060 Through $C07F

RW	Hex	Dec	Neg Dec	Label	Section	Use
W	$C06x					Reserved (write)
R7	$C060	49248	-16288	RD80SW	4.1	See if 80/40 switch down (= 40)
R7	$C061	49249	-16287	RDBTN0	9.1.3†	See if switch 0 or (⌂) pressed
R7	$C062	49250	-16286	RDBTN1	9.2†	See if switch 1 or (◆) pressed
R7	$C063	49251	-16285	RD63	9.1,9.2	See if mouse button not pressed
R7	$C064	49252	-16284	PDL0	9.2	See if hand control button 0 pressed
R7	$C065	49253	-16283	PDL1	9.2	See if hand control button 1 pressed
R7	$C066	49254	-16282	MOUX1	9.1.3	See if mouse X1 (direction) is high
R7	$C067	49255	-16281	MOUY1	9.1.3	See if mouse Y1 (direction) is high
N	$C068 through	49256	-16280			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	9.2	Designated trigger or reset location
N	$C071 through	49265	-16271			Reserved
N	$C07D	49277	-16259			
R7	$C07E	49278	-16258	RDIOUDIS		See if IOUDIS on; trigger paddle timer; reset VBLINT
W				IOUDIS	5.6,9.1.3	On: enable access to DHIRES switch; disable $C058-$C05F IOU access
R7	$C07F	49279	-16257	RDDHIRES	5.6,9.1.3	See if DHIRES on
W				IOUDIS	5.6	Off: disable access to DHIRES switch; enable $C058-$C05F IOU access

† Also section 4.1.

Table B-8. Addresses $C080 Through $C0AF

RW	Hex	Dec	Neg Dec	Label	Section	Use
R	$C080	49280	-16256		2.4.2	Read RAM; no write; use $D000 bank 2
RR	$C081	49281	-16255		2.4.2	Read ROM, write RAM; use $D000 bank 2
R	$C082	49282	-16254		2.4.2	Read ROM; no write; use $D000 bank 2
RR	$C083	49283	-16253		2.4.2	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		2.4.2	Read RAM; no write; use $D000 bank 1
RR	$C089	49289	-16247		2.4.2	Read ROM, write RAM; use $D000 bank 1
R	$C08A	49290	-16246		2.4.2	Read ROM; no write; use $D000 bank 1
RR	$C08B	49291	-16245		2.4.2	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		7.3, 11.11	Port 1 ACIA Transmit/receive register
R/W	$C099	49305	-16231		7.3, 11.11	Port 1 ACIA Status register
R/W	$C09A	49306	-16230		7.3, 11.11, Appendix E	Port 1 ACIA Command register
R/W	$C09B	49307	-16229		7.3, 11.11	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		8.3, 11.11	Port 2 ACIA Transmit/receive register
R/W	$C0A9	49321	-16215		8.3, 11.11	Port 2 ACIA Status register
R/W	$C0AA	49322	-16214		8.3, 11.11, Appendix E	Port 2 ACIA Command register
R/W	$C0AB	49323	-16213		8.3, 11.11	Port 2 ACIA Control register
N	$C0AC through	49324	-16212			Reserved
N	$C0AF	49327	-16209			

Table B-9. *Addresses $C0B0 Through $C0FF*

RW	Hex	Dec	Neg Dec	Label	Section	Use
N	$C0B0	49328	-16208			Reserved
	through					
N	$C0BF	49343	-16193			
N	$C0C0	49344	-16192			Reserved
	through					
N	$C0CF	49359	-16177			
N	$C0D0	49360	-16176			Reserved
	through					
N	$C0DF	49375	-16161			
N	$C0E0	49376	-16160			Reserved
	through					
N	$C0EF	49391	-16145			
N	$C0F0	49392	-16144			Reserved
	through					
N	$C0FF	49407	-16129			

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 (section 3.4.2).

C.1 The Tables

Appendix H contains tables and examples of the derivation of each form of address from either of the other forms.

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 of the location. These labels are listed only to assist you in finding the named location in the firmware listings, or in remembering 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),* indicators (labeled *indic)* specifying whether there are optional routines, vector addresses (labeled *vector),* and an RTS instruction location.

The column labeled *Section* contains the number of the section that describes the item. If there is no description except in a table in this appendix, a section number is not listed.

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.

C.2 Port Addresses

Table C-1. *Serial Port 1 Addresses*

Hex	Dec	Neg Dec	Label	Type	Section	Description
$C100	49408	-16128		entry	3.1.1	Main port 1 entry point
$C105	49413	-16123		ident	3.4.2	ID byte ($38)
$C107	49415	-16121		ident	3.4.2	ID byte ($18)
$C10B	49419	-16117		ident	3.4.2	Firmware card signature ($01)
$C10C	49420	-16116		ident	3.4.2	Super Serial Card ID ($31)
$C10D	49421	-16115		offset	7.4	Low-order PINIT address
$C10E	49422	-16114		offset	7.4	Low-order PREAD address
$C10F	49423	-16113		offset	7.4	Low-order PWRITE address
$C110	49424	-16112		offset	7.4	Low-order PSTATUS address
$C111	49425	-16111		indic	3.4.2	Non-zero: no optional routines

Table C-2. *Serial Port 2 Addresses*

Hex	Dec	Neg Dec	Label	Type	Section	Description
$C200	49664	-15872		entry	3.1.1	Main port 2 entry point
$C205	49669	-15867		ident	3.4.2	ID byte ($38)
$C207	49671	-15865		ident	3.4.2	ID byte ($18)
$C20B	49675	-15861		ident	3.4.2	Firmware card ID ($01)
$C20C	49676	-15860		ident	3.4.2	Super Serial Card ID ($31)
$C20D	49677	-15859		offset	8.4	Low-order PINIT address
$C20E	49678	-15858		offset	8.4	Low-order PREAD address
$C20F	49679	-15857		offset	8.4	Low-order PWRITE address
$C210	49680	-15856		offset	8.4	Low-order PSTATUS address
$C211	49681	-15855		indic	3.4.2	Non-zero: no optional routines

Table C-3. *Video Firmware Addresses*

Hex	Dec	Neg Dec	Label	Type	Section	Description
$C300	49920	-15616		entry	3.1.1	Main video entry point (output only)
$C305	49925	-15611	C3KEYIN	ident	3.4.2	ID byte ($38)
$C307	49927	-15609	C3COUT1	ident	3.4.2	ID byte ($18)
$C30B	49931	-15605		ident	3.4.2	Firmware card signature ($01)
$C30C	49932	-15604		ident	3.4.2	80-column card ID ($88)
$C30D	49933	-15603		offset	5.9	Low-order PINIT address
$C30E	49934	-15602		offset	5.9	Low-order PREAD address
$C30F	49935	-15601		offset	5.9	Low-order PWRITE address
$C310	49936	-15600		offset	5.9	Low-order PSTATUS address
$C311	49937	-15599	MOVEAUX	entry	2.5.3	Routine for main/auxiliary control swapping (Also called AUXMOVE)

Table C-4. *Mouse Port Addresses*

Hex	Dec	Neg Dec	Label	Type	Section	Description
$C400	50176	-15360		entry		Main mouse entry point
$C405	50181	-15355		ident	3.4.2	ID byte ($38)
$C407	50183	-15353		ident	3.4.2	ID byte ($18)
$C40B	50187	-15349		ident	3.4.2	Firmware card signature ($01)
$C40C	50188	-15348		type	3.4.2	X-Y pointing device ID ($20)
$C40D	50189	-15347		offset	9.1.4	Low-order PINIT address
$C40E	50190	-15346		offset	9.1.4	Low-order PREAD address
$C40F	50191	-15345		offset	9.1.4	Low-order PWRITE address
$C410	50192	-15344		offset	9.1.4	Low-order PSTATUS address
$C411	50193	-15343		indic	3.4.2	Optional routines follow ($00)
$C412	50194	-15342	SETMOUSE	offset	9.1.4	Low-order SETMOUSE address
$C413	50195	-15341	SERVEMOUSE	offset	9.1.4	Low-order SERVEMOUSE address
$C414	50196	-15340	READMOUSE	offset	9.1.4	Low-order READMOUSE address
$C415	50197	-15339	CLEARMOUSE	offset	9.1.4	Low-order CLEARMOUSE address
$C416	50198	-15338	POSMOUSE	offset	9.1.4	Low-order POSMOUSE address
$C417	50199	-15337	CLAMPMOUSE	offset	9.1.4	Low-order CLAMPMOUSE address
$C418	50200	-15336	HOMEMOUSE	offset	9.1.4	Low-order HOMEMOUSE address
$C419	50201	-15335	INITMOUSE	offset	9.1.4	Low-order INITMOUSE address

C.3 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	Section	Description
$C600	50688	-14848		entry	6.1	Disk drive firmware entry point
$C700	50944	-14592		entry	6.2	External disk startup routine
$C803	51203	-14333	NEWIRQ	entry	E.1	IRQ handling routine

C.4 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.

C.5 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	Section	Description
$F800	63488	-2048	PLOT	entry	5.8	Plots a low-resolution block
$F819	63513	-2023	HLINE	entry	5.8	Draws low-resolution horizontal line
$F828	63528	-2008	VLINE	entry	5.8	Draws low-resolution vertical line
$F832	63538	-1998	CLRSCR	entry	5.8	Clears low-resolution screen
$F836	63542	-1994	CLRTOP	entry	5.8	Clears top 40 low-resolution lines
$F864	63588	-1948	SETCOL	entry	5.8	Sets low-resolution color (Table 5-4)
$F871	63601	-1935	SCRN	entry	5.8	Reads color of low-resolution block
$F941	63809	-1727	PRNTAX	entry	5.8	Displays (A) and (X) in hex
$F94A	63818	-1718	PRBL2	entry	5.8	Sends (X) blanks to output
$FA47	63845	-1691	NEWBRK	entry	E.2	Apple IIc break handler
$FA62	64098	-1438	RESET	entry	2.6	Hardware reset routine
$FB1E	64286	-1250	PREAD	entry	9.2	Reads hand control position
$FB6F	64367	-1169	SETPWRC	entry	2.6.4	Routine to create power-up byte
$FBB3	64435	-1101		ident	F.1.2	Machine identification byte
$FBC0	64448	-1088		ident	F.1.2	Machine identification byte
$FBDD	64477	-1059	BELL1	entry	4.2.2	Sends 1 kHz beep to speaker
$FC42	64578	-958	CLREOP	entry	5.8	Clears from cursor to bottom
$FC58	64600	-936	HOME	entry	5.8	Clears; cursor to upper left
$FC9C	64668	-868	CLREOL	entry	5.8	Clears from cursor to end of line
$FC9E	64670	-866	CLEOLZ	entry	5.8	Clears from BASL to end of line
$FCA8	64680	-856	WAIT	entry		Delays for time specified by (A)
$FD0C	64780	-756	RDKEY	entry	3.2.1	Displays cursor, jumps to (KSW)
$FD1B	64795	-741	KEYIN	entry	3.2.2	Waits for keypress, reads key
$FD35	64821	-715	RDCHAR	entry	4.1.2	Gets input, interprets ESC codes
$FD67	64871	-665	GETLNZ	entry	4.1.2	Sends CR to output, goes to GETLN
$FD6A	64874	-662	GETLN	entry	3.2.3	Displays prompt, gets input line
$FD6F	64879	-657	GETLN1	entry	4.1.2	No prompt; gets input line
$FD8B	64907	-629	CROUT1	entry	5.8	Clears to end of line, calls CROUT
$FD8E	64910	-626	CROUT	entry	5.8	Sends CR to output
$FDDA	64986	-550	PRBYTE	entry	5.8	Sends (A) to output
$FDE3	64995	-541	PRHEX	entry	5.8	Displays low nibble of (A) in hex
$FDED	65005	-531	COUT	entry	3.3.1	Jumps to (CSW)
$FDF0	65008	-528	COUT1	entry	3.3.2	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	5.8	Sends ERR to output; beeps
$FF3A	65338	-198	BELL	entry	4.2.2	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	10.1	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)

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 full account. For more information, refer to the manuals that are provided with each product.

D.1 Operating Systems

This section discusses the operating systems that the Apple IIc does and does not work with.

D.1.1 ProDOS

ProDOS is the preferred disk operating system for the Apple IIc. It supports startup from the external disk drive, interrupts, and all other hardware and firmware features of the Apple IIc.

D.1.2 DOS

The Apple IIc works with DOS 3.3. Its disk drive support hardware and firmware can also access DOS 3.2 disks by using the *BASICS* disk. However, neither version of DOS takes full advantage of the features of the Apple IIc. DOS support is provided only for the sake of Apple II series compatibility.

D.1.3 Pascal Operating System

Version 1.2 of the Pascal Operating System uses 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 input/output firmware entry points are rigidly defined (rather than being accessed via a table), and the firmware does not support these entry points.

D.1.4 CP/M

CP/M, and any other operating system that requires an interface card, will not work on the Apple IIc.

D.2 Languages

For further information about these languages, refer to the manuals that came with them.

This section discusses special techniques to use, and characteristics to be aware of, when using Apple programming languages with the Apple IIc. It is also a guide to using this reference manual with these languages.

D.2.1 Applesoft BASIC

Use the appendixes to make or find decimal conversions. Appendix H has tables and examples to help you convert numbers between hexadecimal, decimal, and negative (complementary) decimal. All the addresses listed in Appendixes B and C—screen holes, hardware addresses, firmware entry points, and so on—are given in all three numeric forms.

The focus of the chapters in this manual is 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. 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.

D.2.2 Integer BASIC

Unless you load a version of DOS into your Apple IIc, you will not have Integer BASIC available inside the machine. ProDOS does not support Integer BASIC.

D.2.3 Pascal Language

The Pascal language works on the Apple IIc under versions 1.1 and 1.2 of the Pascal Operating System. However, for best performance, use Pascal version 1.2.

D.2.4 FORTRAN

FORTRAN works under version 1.1 of the Pascal Operating System which, as explained in section D.1.3, 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.

Interrupts

This appendix presents a unified account of 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, rather than using the built-in interrupt-handling firmware, compatibility with possible future Apple II series computers or revisions cannot be guaranteed.

E.1 Introduction

This section orients you to interrupts and their effects on the Apple IIc hardware.

E.1.1 What Is an Interrupt?

On a computer, an interrupt is a signal that abruptly causes the computer to stop what it is currently doing and immediately attend to an important time-dependent task. For example, the Apple IIc mouse sends an interrupt to the computer every time it moves. This is necessary because unless the mouse is read shortly after it moves, the signal indicating its direction is lost.

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 had been doing, those conditions can be restored.

E.1.2 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. Thus both DOS and the Monitor work with interrupts on the Apple IIc.

If, however, you want software that uses interrupts to work on the Apple IIe and the Apple II Plus, you must use either ProDOS, Apple's new enhanced disk operating system, or Pascal 1.2. Both operating systems have full interrupt support built in.

Interrupts are effective only if they are enabled most of the time. Interrupts that occur while interrupts are disabled cannot be detected. Due to the critical timing of disk read and write operations, Pascal, DOS, and ProDOS turn off interrupts while accessing the disk. Thus it is important to remember that while a disk drive is being accessed, all sources of interrupts 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.

E.1.3 Interrupt Handling on the 65C02

From the point of view of the 65C02, there are three possible causes of interrupts.

1. If 65C02 interrupts are not masked (that is, the CLI instruction has been used), the IRQ line on the microprocessor can be pulled low. This is the standard technique by which a device indicates that it needs immediate attention.
2. The processor executes a break (BRK, opcode $00) instruction.
3. A non-maskable interrupt (NMI) occurs. Because the NMI line in the Apple IIc's 65C02 is not used, this never happens.

Options 1 and 2 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 IRQ, finish executing the current instruction. (If BRK, current instruction is already finished.)
2. Push high byte of program counter onto stack.
3. Push low byte of program counter onto stack.
4. Push program status byte onto stack.
5. Jump to address stored in $FFFE, $FFFF, that is, JMP ($FFFE).

The different sources of interrupt signals are discussed below.

E.1.4 The Interrupt Vector at $FFFE

In the Apple IIc computer 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. Due to the complexity of interrupts in the Apple IIc, it is recommended that you use it rather than writing your own interrupt handling routine.

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.

E.2 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.

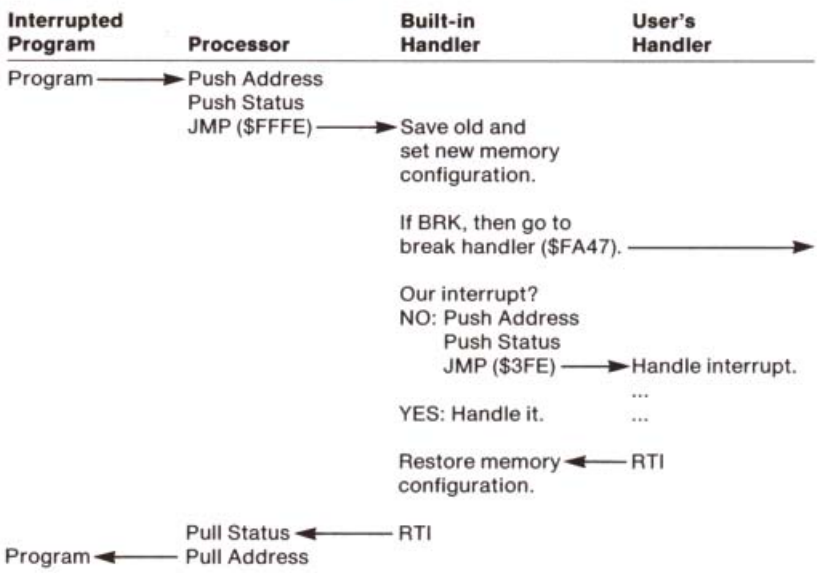
Next the handler checks to see if the interrupt was caused by a break instruction, and if it was, handles it as described in section E.4.

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 $3FE and $3FF 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. Figure E-1 illustrates this whole process. Each of the steps is explained in detail in the sections that follow.

Figure E-1. *Interrupt-Handling Sequence*



E.2.1 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 (see section E.2.2).

Note: Because main memory is switched in, all memory addresses used later in this appendix are in main memory unless otherwise specified.

E.2.2 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: $100 for the main stack pointer when the auxiliary stack is active, and $101 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 $100, 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 $101, and the main stack pointer is retrieved from $100. Then the main stack is switched in for use. After the interrupt has been handled, the stack pointer is restored to its original value.

E.3 User's Interrupt Handler at $3FE

Screen hole locations can be set up to indicate that the user's interrupt handler should be called when certain interrupts occur. To use such a routine, place its address at $3FE and $3FF 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 using an RTI instruction.

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 in section E.4.

In general there is no guaranteed 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 2 switched in for reading and writing).

E.4 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 one). 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 $3F0 and $3F1.

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 1 switched in
Bit 1	=	1	if bank-switched $D000 page 2 switched in
Bit 0	=	0	

E.5 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 classes of interrupt sources: those associated with use of the mouse, and those associated with the two 6551 ACIA circuits (the chips that control serial communication).

The interrupts that are associated with the mouse are these:

- An interrupt can be generated when the mouse is moved in the horizontal (X) direction.
- An interrupt can be generated when the mouse is moved in the vertical (Y) direction.
- An interrupt can be generated every 1/60 second. This is called the **vertical blanking** (VBL) interrupt, and is synchronized with a signal used for the video display.
- Using the firmware, interrupts can be generated when the mouse button is pressed.

The interrupts that are associated with the ACIAs are these:

- An interrupt can be generated when a key is pressed. The firmware can use this interrupt to buffer keystrokes, or it can pass the interrupt on to the user.
- An interrupt can be generated by a device attached to the external disk drive port. The firmware can pass this interrupt on to the user.
- An interrupt can be generated when 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.
- An interrupt can be generated when 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.
- An interrupt can be generated when 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.
- An interrupt is generated when the keyboard strobe is cleared. The firmware absorbs this interrupt.

E.6 Firmware Handling of Interrupts

The following sections discuss the various sources of interrupts and how they should be used in conjunction with the built-in interrupt handler.

E.6.1 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:

- Mouse movement in the X direction
- Mouse movement in the Y direction
- Change of state of the button
- Rising edge of the vertical blanking signal.

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 $3FE, 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 $77C 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 $77C 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:

$47C	Low byte of X coordinate
$4FC	Low byte of Y coordinate
$57C	High byte of X coordinate
$5FC	High byte of Y coordinate

$77C 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.

As already mentioned, 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.

E.6.2 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 8 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, ⌘-CONTROL-X, clears the buffer.

Using Keyboard Buffering Firmware

See Chapter 8.

Keyboard buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself.

1. Disable processor interrupts (SEI).
2. Set location $5FA to $80. This tells the firmware to buffer keystrokes without calling the user's interrupt handler.
3. Set locations $5FF and $6FF 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 $F. For example:

```
LDA  $C0AA      ;read port 2 ACIA command register
ORA  #$0F       ;set low nibble to $F
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 $5FA 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 $F. For example:

```
LDA  $C0AA     ;read port 2 ACIA command register
ORA  #$0F      ;set low nibble to $F
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 $4FA. 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 $4FA to 0.

E.6.3 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 $5F9 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 $F. For example:

```
LDA  $C09A     ;read port 1 ACIA command register
ORA  #$0F      ;set low nibble to $F
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 $4F9. 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 $4F9 to 0.

E.6.4 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 is 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:

1. Disable processor interrupts (SEI).
2. Set location $4FF to $C1 to buffer port 1, or to $C2 to buffer port 2.
3. Set locations $57F and $67F to 0. 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 $D. For example:

```
LDA  $C09A     ;read port 1 ACIA command register
AND  $F0       ;clear low nibble
ORA  #$0D      ;set low nibble to $D
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).

See Chapter 8.

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.

1. Disable processor interrupts (SEI).
2. Set location $4FF 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 $D. For example:

```
LDA  $C09A     ;read port 1 ACIA command register
AND  $F0       ;clear low nibble
ORA  #$0D      ;set low nibble to $D
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 ($4F9 for port 1; $4FA 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 ($4F9 for port 1; $4FA 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 remains 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.

E.7 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.

E.7.1 Using Mouse Interrupts Without the Firmware

To use mouse interrupts without the firmware, as mentioned above, you must set your own interrupt vectors. If $D000-$FFFF ROM is ever switched in, the built-in interrupt handler will absorb the mouse interrupts. Tables E-1 and E-2 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-1. 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-2. 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 is guaranteed to be valid for at least 40 microseconds. Average duration is at least 200 microseconds. This means you should read the direction as soon as possible.

E.7.2 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 (section E.6.1).

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 ACIA's 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:	Data Register	=	$C098
	Status Register	=	$C099
	Command Register	=	$C09A
	Control Register	=	$C09B
Port 2:	Data Register	=	$C0A8
	Status Register	=	$C0A9
	Command Register	=	$C0AA
	Control Register	=	$C0AB

Apple II Series Differences

This appendix compares the Apple IIc to the Apple IIe, Apple II Plus, and Apple II.

F.1 Overview

This appendix does not contain an exhaustive list of differences. However, it does include those differences most likely to affect the accuracy of programs, displays, and instructions created for end users of two or more models from the Apple II Series.

As an 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.

Note: The following *equations* are merely an overview of what each model of Apple II Series is with respect to its predecessor. These equations are in terms of functional equivalence, not strict equality. For example,

Apple IIe = Apple II Plus + Apple Language Card

does not mean there is an actual language card or slot—just that the one machine functions as if it were the other with such a card (with its connector) in a slot.

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
			+	extra 64K RAM)
			-	slot 0
Apple IIc	=	IIe	+	extended 80-column text card
			+	80/40 switch
			+	keyboard switch
			+	disk 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

F.1.1 Type of CPU

The CPU in the Apple II and II Plus is the 6502. The Apple IIe uses a 6502A. The Apple IIc uses 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 instruction cycle times that differ.

F.1.2 Machine Identification

Identification of Apple II series computers is as follows:

Machine	$FBB3	$FBC0	$FB1E
Apple II	$38		
Apple II Plus	$EA		
Apple IIe	$06	$EA	
Apple IIc	$06	$00	
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.

With regard to ProDOS, its MACHID byte, at location $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 a 1 if the computer is not one of these machines. In addition, for an Apple IIc, bits 7 and 6 are 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.

F.2 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.

F.2.1 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, following a procedure described in the *Apple II Reference Manual*.

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 Card.

F.2.2 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 15.75 K (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 (Apple)-(CONTROL)-(RESET) causes the self-test firmware to take over the slot ROM address spaces.

The Apple IIc ROM also uses the 15.75 K 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 pre-empt any port firmware space.

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.

F.2.3 Peripheral-Card Memory Spaces

Each Apple IIc port has up to sixteen 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.

F.2.4 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 most hardware page uses. However, for many characteristics, the Apple IIe and IIc work one way, while the Apple II and II Plus work another.

$C000 to $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 to $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 to $C02F

On the Apple II, II Plus, and IIe, reading any address $C02x toggles the cassette output signal. On the Apple IIc, both reading from and writing to these locations are reserved.

$C030 to $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 these addresses is explicitly reserved.

$C040 to $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 interrupts (XINT and YINT).

$C050 to $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, 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 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 to $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 to $C07F

On the Apple II, II Plus, and IIe, reading from or writing to any address of the form $C07x triggered 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 to $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 to $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.

F.2.5 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.

F.3 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.

F.3.1 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.

F.3.2 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 built-in hardware and firmware that are functional equivalents of slots with cards in them (and back-panel connectors). These are called **ports** on the Apple IIc.

F.3.3 Interrupts

Interrupts on the Apple IIc are described in Appendix E.

The Apple IIc is the first computer in the Apple II Series to have built-in interrupt-handling capabilities.

F.4 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.

F.4.1 Keys

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 (Ó) and a (Apple) key; the Apple II and II Plus do not have these two keys.

The captions on several keys—(ESC), (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.

F.4.2 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 (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 productions 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*.

F.5 Speaker

The Apple IIc has two speaker features that the three previous models do not have. They are a two-channel, but monaural, mini-phone jack for headphones—which disconnects the internal speaker when something is plugged into it—and a volume control.

F.6 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.

F.6.1 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 compatible with some software, you have to switch the Apple IIc keyboard to uppercase by pressing (CAPS LOCK).

F.6.2 MouseText

MouseText characters (Chapter 5) are available on every Apple IIc, and on any Apple IIe that has had its ROMs appropriately upgraded, if necessary.

F.6.3 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.

F.6.4 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.

F.7 Disk I/O

The Apple II, II Plus, and IIe can support up to six (four is the recommended maximum) disk drives 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.

F.8 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.

F.8.1 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 [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 several functions that Apple IIc serial port commands don't support: masking incoming line feed after carriage return; 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; ignoring keyboard input, and so on.

Following a CONTROL-I nnnN command, the Apple IIc automatically generates carriage return after nnn characters; with the Super Serial Card, you need to turn this on with CONTROL-I C.

F.8.2 Serial I/O Buffers

The communication port firmware uses auxiliary memory page 8 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 8. However, such information will be destroyed when the communication port is activated.

F.9 Mouse and Hand Controls

The DB-9 back-panel connector on the Apple IIc is used for both the mouse and hand controls. On the Apple IIc, the DB-9 connector supports hand controls only. On the Apple IIe, the mouse must use the connector on the interface card.

F.9.1 Mouse Input

The Apple IIc provides built-in firmware support for a mouse connected to the DB-9 mouse and hand control 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 than it is on the Apple IIe.

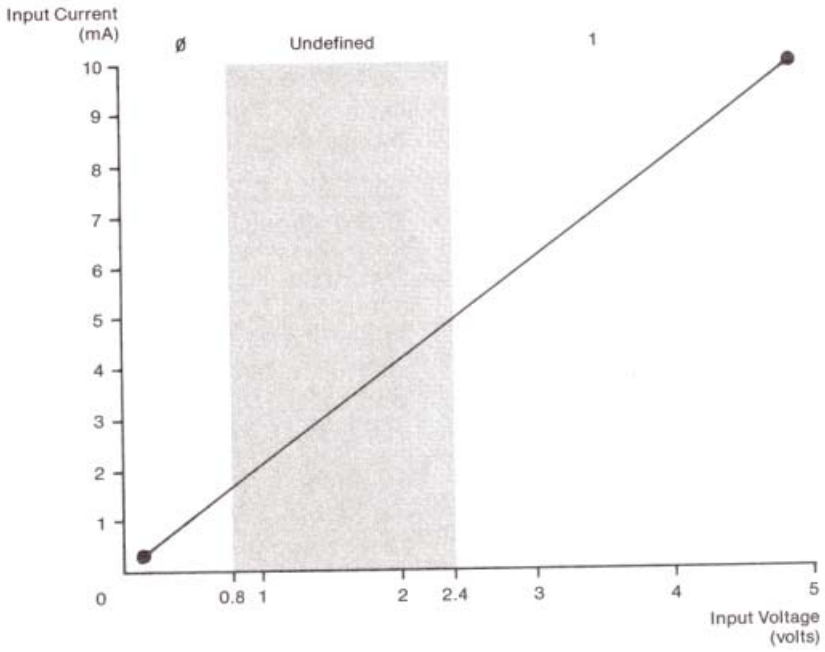
F.9.2 Hand Control Input and Output

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 voltage-current curve for hand controls differs for the Apple IIc compared with 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 control signals using the same circuits.

Figure F-1. *Apple II, II Plus, and IIe Hand Control Signals*



The paddle timing circuit on the Apple II Plus is slightly different than the one on the Apple IIe and IIc. On the Apple IIe and IIc the 100 ohm fixed resistor is betwen 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 conected directly to the discharge lead, and the fixed resistor is in series with the paddle resistor.

F.10 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.

F.11 Hardware

Besides the different microprocessors used in various models in the Apple II series (section F.1.1), there are important differences in power specifications and custom chips.

F.11.1 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.

F.11.2 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 (IWM). The Apple IIc has two hybrid units (AUD and VID) for audio and video amplification.

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 there is a composite table of the ASCII codes and the characters associated with them on all the models discussed.

G.1 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-6) 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, from the fact that 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-7. 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.

Note: 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 follows 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, there are arrow symbols on TAB, CAPS-LOCK, RETURN, and the two SHIFT keys (as shown in Figure G-3).

G.1.1 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)*

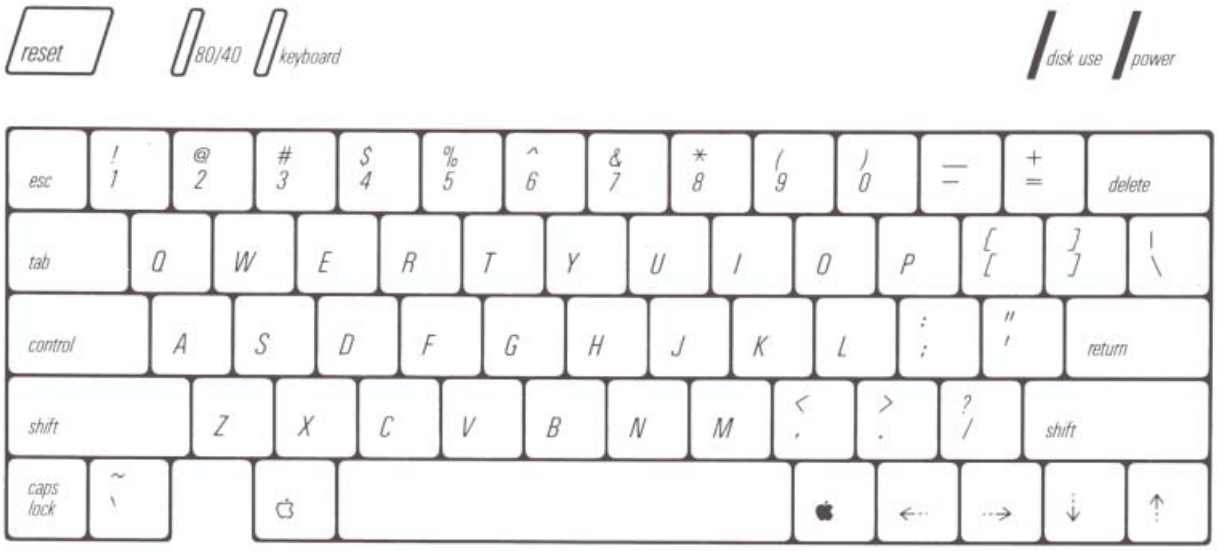


Table G-1. *Keys and ASCII Codes. Codes are shown here in hexadecimal; to find the decimal equivalents, use Table G-7.*

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

Table G-1—Continued. *Keys and ASCII Codes. Codes are shown here in hexadecimal; to find the decimal equivalents, use Table G-7.*

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

G.1.2 USA Simplified (Dvorak) Keyboard

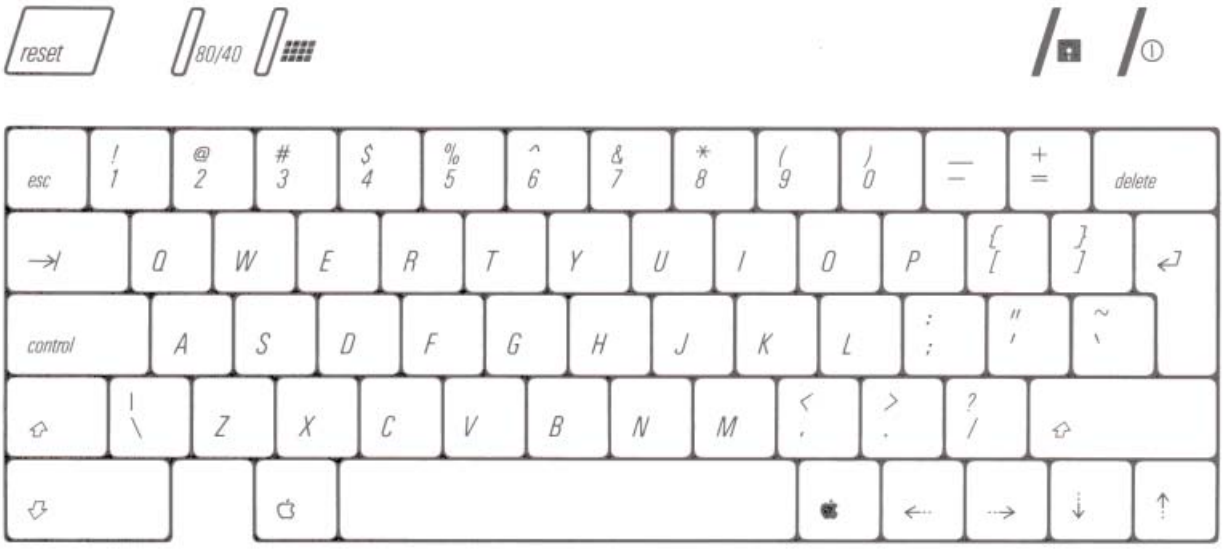
Figure G-2 shows the Dvorak layout of the USA keyboard. Characters are paired up on keys in exactly the same way as on the USA Standard keyboard; only individual key positions are changed. In fact, you can change the keycap arrangement to match Figure G-2, lock the keyboard switch in its down position, 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)*

G.1.3 ISO Layout of USA Keyboard

Figure G-3 shows the layout of the keyboard 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)*



G.1.4 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)*

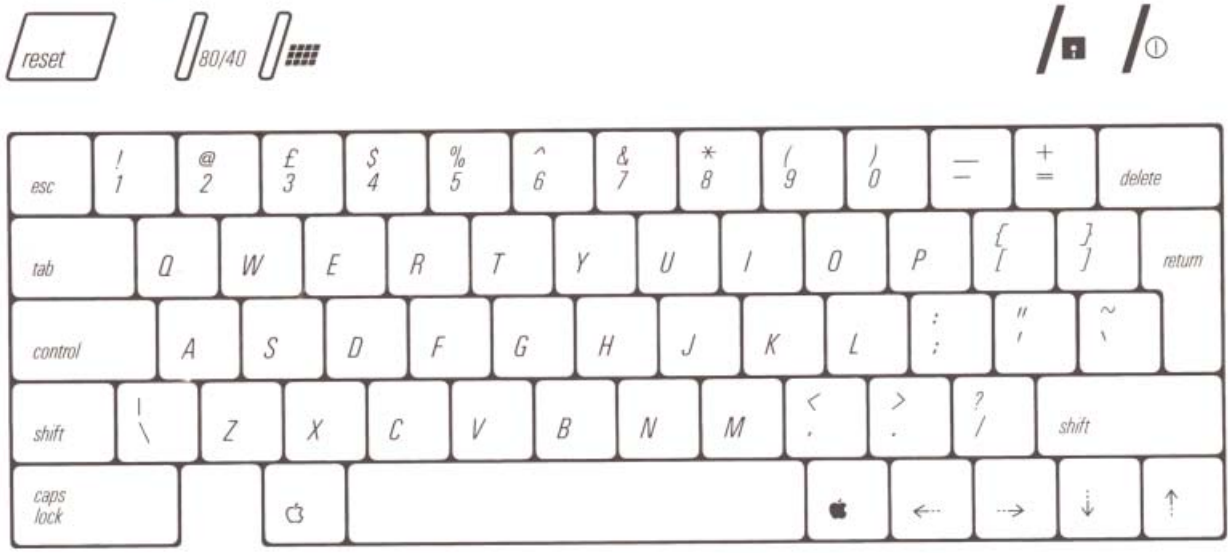


Table G-2. *English Keyboard Code Differences From Table G-1*

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
3£	33	3	33	3	23	£	23	#

G.1.5 French and Canadian Keyboards

With the keyboard switch up, the French model of the Apple IIc keyboard layout is as shown in Figure G-3, and the Canadian is as shown in Figure G-1. On both models, keystrokes produce the ASCII codes shown in Table G-1.

Note: 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.

With the keyboard switch down, the French model keyboard layout is as shown in Figure G-5, and the Canadian model keyboard layout is as shown in Table G-3. The changes in ASCII code production (from that in Table G-1) are shown in Table G-3.

Figure G-5. *French Keyboard (Keyboard Switch Down)*

Figure G-6. Canadian Keyboard (Keyboard Switch Down)

Table G-3. French and Canadian Keyboard Code Differences From Table G-1

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
&1	26	&	26	&	31	1	31	1
é2	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
è7	7D	è	7D	è	37	7	37	7
!8	21	!	21	!	38	8	38	8
ç9	5C	ç	1C	FS	39	9	1C	FS
à0	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	/

G.1.6 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-4.

Figure G-7. *German Keyboard (Keyboard Switch Down)*

Table G-4. *German Keyboard Code Differences From Table G-1*

Key	Key Alone Hex	Key Alone Char	CONTROL + Key Hex	CONTROL + Key Char	SHIFT + Key Hex	SHIFT + Key Char	Both + Key Hex	Both + Key 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	:

G.1.7 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-5.

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 shown in Table G-5 or Table G-7.

Figure G-8. *Italian Keyboard (Keyboard Switch Down)*

Table G-5. *Italian Keyboard Code Differences From Table G-1*

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	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
ç5	5C	ç	1C	FS	35	5	1C	FS
è6	7D	è	7D	è	36	6	36	6
)7	29	)	29	)	37	7	37	7
£8	23	£	23	£	38	8	38	8
à9	7B	à	7B	à	39	9	39	9
é0	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	!

G.1.8 Western Spanish Keyboard

With the keyboard switch up, the Western (that is, American) Spanish model of the 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-6.

Figure G-9. *Western Spanish Keyboard (Keyboard Switch Down)*

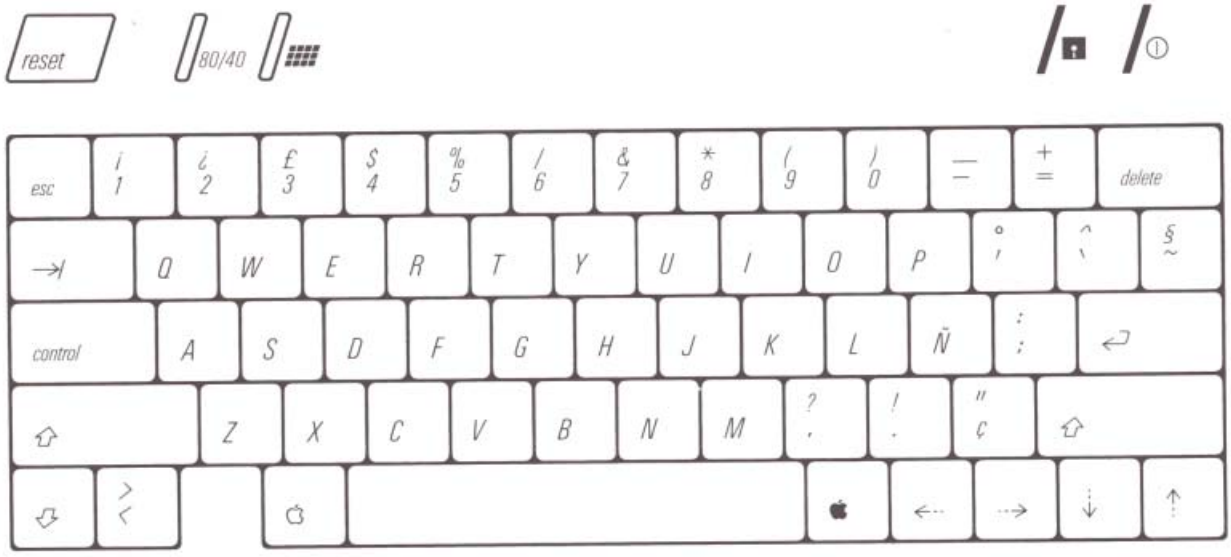


Table G-6. *Western Spanish Keyboard Code Differences From Table G-1*

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
2"	32	2	32	2	22	"	22	"
3£	33	3	33	3	23	£	23	£
6&	36	6	00	NUL	26	&	00	NUL
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	=
'?	27	'	27	'	3F	?	3F	?
`¿	60	`	60	`	5D	¿	5D	¿
- ^	7E	-	1E	RS	5E	^	1E	RS
+*	2B	+	1B	ESC	2A	*	1B	ESC
Ñ	7C	Ñ	1C	FS	5C	Ñ	1C	FS
çi	7D	ç	7D	ç	5B	¡	5B	¡
°§	7B	°	00	NUL	40	§	00	NUL
<>	3C	<	1E	RS	3E	>	1E	RS
,;	2C	,	2C	,	3B	;	3B	;
.:	2E	.	2E	.	3A	:	3A	:

G.2 ASCII Character Sets

Table G-7 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, a circled number in the main table refers to a column in the lower part of the table.

Table G-7. 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
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	4B	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	87	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

•	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)	£	$	@	[	\	]	^	`	{	|	}	~
Deutsch	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
Français	£	$	à	°	ç	§	^	`	é	ù	è	¨
Italiano	£	$	§	°	ç	é	^	ù	à	ò	è	ì
Español	£	$	§	¡	Ñ	¿	^	`	°	ñ	ç	~

G.3 Certifications

In the countries where they are applicable, these certifications replace the USA FCC Class B notice printed on the inside front cover of this manual. The safety instructions apply to all countries.

G.3.1 Radio Interference

This product is designed to comply with specification VDE 0871/6.78, Radio Frequency Interference Suppression of Radio Frequency Equipment, Level B.

G.3.2 Product Safety

This product is designed to meet the requirements of safety standard IEC 380, Safety of Electrically Energized Office Machines.

G.3.3 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.

G.4 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 V DC (nominal)
Supply current	1.2 A (nominal)

Conversion Tables

This appendix briefly discusses bits and bytes and what they can represent. It also contains conversion tables for hexadecimal to decimal and negative decimal, for low-resolution display dot patterns, display color values, 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.

H.1 Bits and Bytes

This section discusses the relationships between bit values and their position within a byte. The following 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.

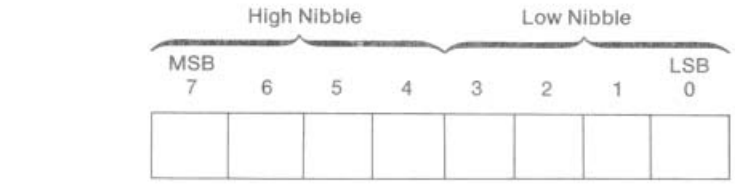
Table H-1. *What a Bit Can Represent*

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
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* .

- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of four bits.
- Four bits comprise a nibble (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 and (because our decimal system has only ten of the sixteen digits we need) A through F.
- Eight bits (two nibbles) make a byte (Figure H-1).

Figure H-1. Bits, Nibbles, and Bytes



Hexadecimal	$80	$40	$20	$10	$08	$04	$02	$01
Decimal	128	64	32	16	8	4	2	1

Binary	Hexadecimal	Decimal
0000	$0	0
0001	$1	1
0010	$2	2
0011	$3	3
0100	$4	4
0101	$5	5
0110	$6	6
0111	$7	7
1000	$8	8
1001	$9	9
1010	$A	10
1011	$B	11
1100	$C	12
1101	$D	13
1110	$E	14
1111	$F	15

- 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 eight-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-13 list some of the ways bytes are commonly interpreted.

- Two bytes make a word. The sixteen bits of a word can represent any one of 256 x 256 or 65536 different values.
- The 65C02 uses a 16-bit word to represent memory locations. It can therefore distinguish among 65536 (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.

H.2 Hexadecimal and Decimal

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

Table H-2. *Hexadecimal/Decimal Conversion*

Digit	$x000	$0x00	$00x0	$000x
F	61440	3840	240	15
E	57344	3584	224	14
D	53248	3328	208	13
C	49152	3072	192	12
B	45056	2816	176	11
A	40960	2560	160	10
9	36864	2304	144	9
8	32768	2048	128	8
7	28672	1792	112	7
6	24576	1536	96	6
5	20480	1280	80	5
4	16384	1024	64	4
3	12288	768	48	3
2	8192	512	32	2
1	4096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up.

Examples:

```
$3C = ?

$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 zero left. Add up the hexadecimal numbers.

Example:

```
16215 = $ ?

16215 - 12288 = 3927      12288 = $7000
 3927 -  3840 =   87       3840 = $ F00
   87 -    80 =    7         80 = $  50
             7                7 = $   7
                                ----------
                           16215 = $7F57
```

H.3 Hexadecimal and Negative Decimal

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

Table H-3. Decimal 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 (zeros included). Then add their values (ignoring their signs for a moment). The resulting number, with a minus sign in front of it, is the desired negative decimal number.

Example:

```
$C010 = - ?

$C000:   -12288
$ 000:   -  3840
$   10:  -   224
$    0:  -    16
-------------
$C010    -16368
```

To convert a negative-decimal number directly to a positive decimal number, add it to 65536. (This addition ends up looking like subtraction.)

Example:

```
-151 = + ?

65536 + (-151) = 65536 - 151 = 65385
```

To convert a negative-decimal number to a hexadecimal number, first convert it to a positive decimal number, then use Table H-2.

H.4 Graphics Bits and Pieces

Table H-4 is a quick guide to the hexadecimal values corresponding to 7-bit high-resolution patterns on the display screen. Since the bits are displayed in reverse order, it takes some calculation to determine these values. Table H-4 should make it easy.

The *x* represents bit 7. Zeros represent bits that are off; ones bits that are on. Use the first hexadecimal value if bit 7 is to be off, and the second if it is to be on.

For example, to get bit pattern 00101110, use $3A; for 10101110, use $BA.

Table H-4. *Hexadecimal Values for High-Resolution Dot Patterns*

Bit pattern	(x=0)	(x=1)
x0000000	$00	$80
x0000001	$40	$C0
x0000010	$20	$A0
x0000011	$60	$E0
x0000100	$10	$90
x0000101	$50	$D0
x0000110	$30	$B0
x0000111	$70	$F0
x0001000	$08	$88
x0001001	$48	$C8
x0001010	$28	$A8
x0001011	$68	$E8
x0001100	$18	$98
x0001101	$58	$D8
x0001110	$38	$B8
x0001111	$78	$F8
x0010000	$04	$84
x0010001	$44	$C4
x0010010	$24	$A4
x0010011	$64	$E4
x0010100	$14	$94
x0010101	$54	$D4
x0010110	$34	$B4
x0010111	$74	$F4
x0011000	$0C	$8C
x0011001	$4C	$CC
x0011010	$2C	$AC
x0011011	$6C	$EC
x0011100	$1C	$9C
x0011101	$5C	$DC
x0011110	$3C	$BC
x0011111	$7C	$FC

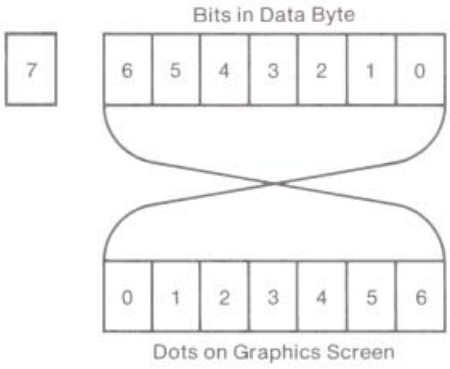


Table H-4—Continued. *Hexadecimal Values for High-Resolution Dot Patterns*

Bit pattern	(x=0)	(x=1)
x0100000	$02	$82
x0100001	$42	$C2
x0100010	$22	$A2
x0100011	$62	$E2
x0100100	$12	$92
x0100101	$52	$D2
x0100110	$32	$B2
x0100111	$72	$F2
x0101000	$0A	$8A
x0101001	$4A	$CA
x0101010	$2A	$AA
x0101011	$6A	$EA
x0101100	$1A	$9A
x0101101	$5A	$DA
x0101110	$3A	$BA
x0101111	$7A	$FA
x0110000	$06	$86
x0110001	$46	$C6
x0110010	$26	$A6
x0110011	$66	$E6
x0110100	$16	$96
x0110101	$56	$D6
x0110110	$36	$B6
x0110111	$76	$F6
x0111000	$0E	$8E
x0111001	$4E	$CE
x0111010	$2E	$AE
x0111011	$6E	$EE
x0111100	$1E	$9E
x0111101	$5E	$DE
x0111110	$3E	$BE
x0111111	$7E	$FE
x1000000	$01	$81
x1000001	$41	$C1
x1000010	$21	$A1
x1000011	$61	$E1
x1000100	$11	$91
x1000101	$51	$D1
x1000110	$31	$B1
x1000111	$71	$F1
x1001000	$09	$89
x1001001	$49	$C9
x1001010	$29	$A9
x1001011	$69	$E9
x1001100	$19	$99
x1001101	$59	$D9
x1001110	$39	$B9
x1001111	$79	$F9

Table H-4—Continued. *Hexadecimal Values for High-Resolution Dot Patterns*

Bit pattern	(x=0)	(x=1)
x1010000	$05	$85
x1010001	$45	$C5
x1010010	$25	$A5
x1010011	$65	$E5
x1010100	$15	$95
x1010101	$55	$D5
x1010110	$35	$B5
x1010111	$75	$F5
x1011000	$0D	$8D
x1011001	$4D	$CD
x1011010	$2D	$AD
x1011011	$6D	$ED
x1011100	$1D	$9D
x1011101	$5D	$DD
x1011110	$3D	$BD
x1011111	$7D	$FD
x1100000	$03	$83
x1100001	$43	$C3
x1100010	$23	$A3
x1100011	$63	$E3
x1100100	$13	$93
x1100101	$53	$D3
x1100110	$33	$B3
x1100111	$73	$F3
x1101000	$0B	$8B
x1101001	$4B	$CB
x1101010	$2B	$AB
x1101011	$6B	$EB
x1101100	$1B	$9B
x1101101	$5B	$DB
x1101110	$3B	$BB
x1101111	$7B	$FB
x1110000	$07	$87
x1110001	$47	$C7
x1110010	$27	$A7
x1110011	$67	$E7
x1110100	$17	$97
x1110101	$57	$D7
x1110110	$37	$B7
x1110111	$77	$F7
x1111000	$0F	$8F
x1111001	$4F	$CF
x1111010	$2F	$AF
x1111011	$6F	$EF
x1111100	$1F	$9F
x1111101	$5F	$DF
x1111110	$3F	$BF
x1111111	$7F	$FF

H.5 Peripheral Identification Numbers

Many Apple products now use Peripheral Identification Numbers (called **PIN numbers**) as shorthand for serial device characteristics. The Apple II Series *System 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.

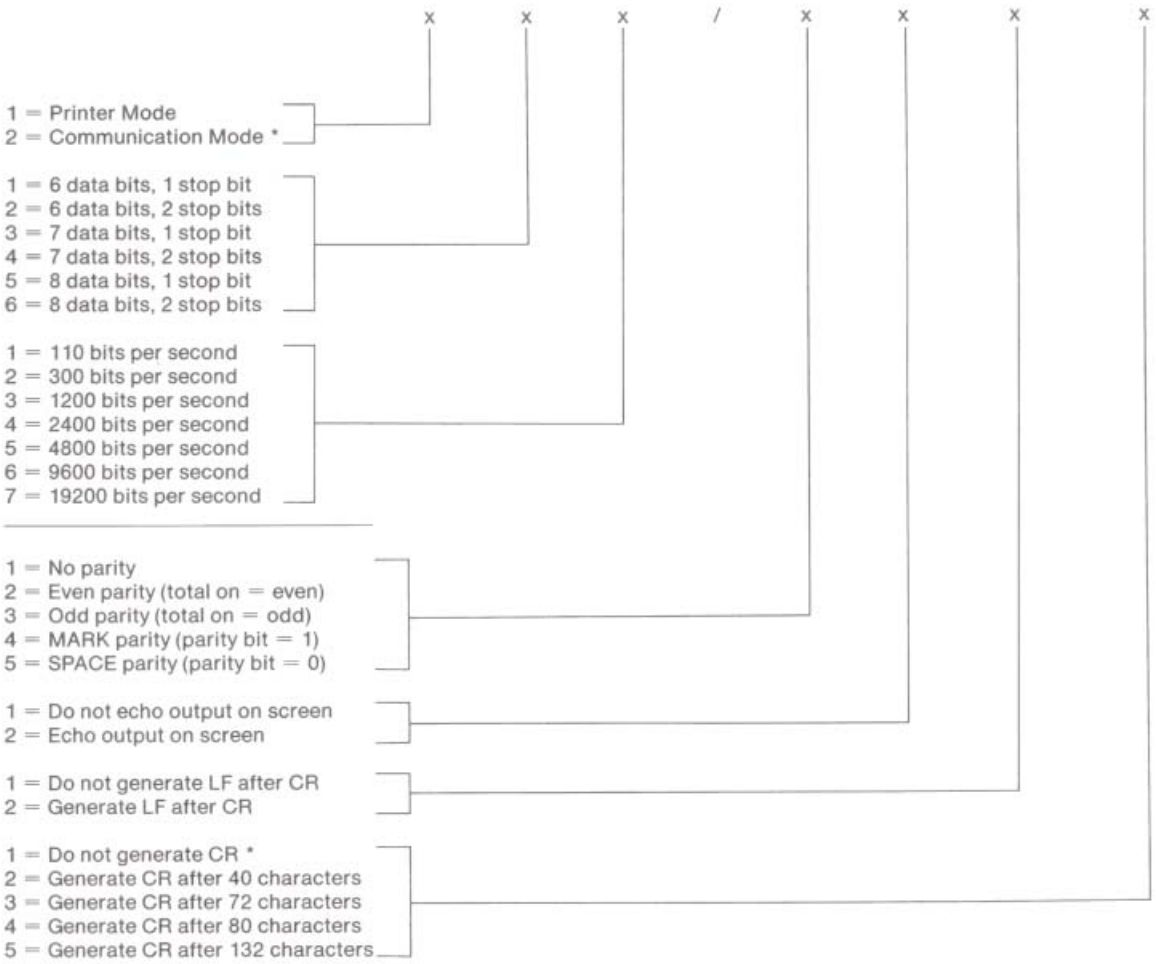
Example:

252/1111 means:

Communication mode
8 data bits, 1 stop bit
300 baud (bits per second)

No parity
Do not echo output to display
No linefeed after carriage return
Do not generate carriage returns

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.

H.6 Eight-Bit Code Conversions

Tables H-6 through H-13 show the entire ASCII character set twice: once with the high bit off, and once with it on. 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 numbers between columns refer to footnotes.
- The columns marked **Pri** and **Alt** indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

  Note that the values $40 through $5F (and $C0 through $DF) in the alternate character set are displayed as MouseText characters (Figure 5-1) if the firmware is set to do so (section 5.2.2), or if the firmware is bypassed.

Note: The primary and alternate displayed character sets in Tables H-6 through H-13 are the result of firmware mapping. The CHARGEN ROM actually contains only one character set. The firmware mapping procedure is described in section 3.36.

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 ←	H	H
0001001	9	$09	HT	Horizontal Tab	CONTROL-I or TAB	I	I
0001010	10	$0A	LF	Line Feed	CONTROL-J or ↓	J	J
0001011	11	$0B	VT	Vertical Tab	CONTROL-K or ↑	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 →	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 ESC	[	[
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	'	Closing Quote		'	'
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		_	

Table H-9. Lowercase Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
1100000	96	$60	`	Opening Quote			`
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

Table H-10. *Control Characters, High Bit On*

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
10000000	128	$80	NUL	Blank (null)	CONTROL-@	@	@
10000001	129	$81	SOH	Start of Header	CONTROL-A	A	A
10000010	130	$82	STX	Start of Text	CONTROL-B	B	B
10000011	131	$83	ETX	End of Text	CONTROL-C	C	C
10000100	132	$84	EOT	End of Transm.	CONTROL-D	D	D
10000101	133	$85	ENQ	Enquiry	CONTROL-E	E	E
10000110	134	$86	ACK	Acknowledge	CONTROL-F	F	F
10000111	135	$87	BEL	Bell	CONTROL-G	G	G
10001000	136	$88	BS	Backspace	CONTROL-H or ←	H	H
10001001	137	$89	HT	Horizontal Tab	CONTROL-I or TAB	I	I
10001010	138	$8A	LF	Line Feed	CONTROL-J or ↓	J	J
10001011	139	$8B	VT	Vertical Tab	CONTROL-K or ↑	K	K
10001100	140	$8C	FF	Form Feed	CONTROL-L	L	L
10001101	141	$8D	CR	Carriage Return	CONTROL-M or RETURN	M	M
10001110	142	$8E	SO	Shift Out	CONTROL-N	N	N
10001111	143	$8F	SI	Shift In	CONTROL-O	O	O
10010000	144	$90	DLE	Data Link Escape	CONTROL-P	P	P
10010001	145	$91	DC1	Device Control 1	CONTROL-Q	Q	Q
10010010	146	$92	DC2	Device Control 2	CONTROL-R	R	R
10010011	147	$93	DC3	Device Control 3	CONTROL-S	S	S
10010100	148	$94	DC4	Device Control 4	CONTROL-T	T	T
10010101	149	$95	NAK	Neg. Acknowledge	CONTROL-U or →	U	U
10010110	150	$96	SYN	Synchronization	CONTROL-V	V	V
10010111	151	$97	ETB	End of Text Blk.	CONTROL-W	W	W
10011000	152	$98	CAN	Cancel	CONTROL-X	X	X
10011001	153	$99	EM	End of Medium	CONTROL-Y	Y	Y
10011010	154	$9A	SUB	Substitute	CONTROL-Z	Z	Z
10011011	155	$9B	ESC	Escape	CONTROL-[or ESC	[	[
10011100	156	$9C	FS	File Separator	CONTROL-\	\	\
10011101	157	$9D	GS	Group Separator	CONTROL-]	]	]
10011110	158	$9E	RS	Record Separator	CONTROL-^	^	^
10011111	159	$9F	US	Unit Separator	CONTROL-_	_	_

Table H-11. Special Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
10100000	160	$A0	SP	Space	SPACE bar		
10100001	161	$A1	!			!	!
10100010	162	$A2	"			"	"
10100011	163	$A3	#			#	#
10100100	164	$A4	$			$	$
10100101	165	$A5	%			%	%
10100110	166	$A6	&			&	&
10100111	167	$A7	'	Closed Quote (acute accent)		'	'
10101000	168	$A8	(			(	(
10101001	169	$A9	)			)	)
10101010	170	$AA	*			*	*
10101011	171	$AB	+			+	+
10101100	172	$AC	,	Comma		,	,
10101101	173	$AD	-	Hyphen		-	-
10101110	174	$AE	.	Period		.	.
10101111	175	$AF	/			/	/
10110000	176	$B0	0			0	0
10110001	177	$B1	1			1	1
10110010	178	$B2	2			2	2
10110011	179	$B3	3			3	3
10110100	180	$B4	4			4	4
10110101	181	$B5	5			5	5
10110110	182	$B6	6			6	6
10110111	183	$B7	7			7	7
10111000	184	$B8	8			8	8
10111001	185	$B9	9			9	9
10111010	186	$BA	:			:	:
10111011	187	$BB	;			;	;
10111100	188	$BC	<			<	<
10111101	189	$BD	=			=	=
10111110	190	$BE	>			>	>
10111111	191	$BF	?			?	?

Table H-12. Uppercase Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
11000000	192	$C0	@			@	@
11000001	193	$C1	A			A	A
11000010	194	$C2	B			B	B
11000011	195	$C3	C			C	C
11000100	196	$C4	D			D	D
11000101	197	$C5	E			E	E
11000110	198	$C6	F			F	F
11000111	199	$C7	G			G	G
11001000	200	$C8	H			H	H
11001001	201	$C9	I			I	I
11001010	202	$CA	J			J	J
11001011	203	$CB	K			K	K
11001100	204	$CC	L			L	L
11001101	205	$CD	M			M	M
11001110	206	$CE	N			N	N
11001111	207	$CF	O			O	O
11010000	208	$D0	P			P	P
11010001	209	$D1	Q			Q	Q
11010010	210	$D2	R			R	R
11010011	211	$D3	S			S	S
11010100	212	$D4	T			T	T
11010101	213	$D5	U			U	U
11010110	214	$D6	V			V	V
11010111	215	$D7	W			W	W
11011000	216	$D8	X			X	X
11011001	217	$D9	Y			Y	Y
11011010	218	$DA	Z			Z	Z
11011011	219	$DB	[	Opening Bracket		[	[
11011100	220	$DC	\	Reverse Slant		\	\
11011101	221	$DD	]	Closing Bracket		]	]
11011110	222	$DE	^	Caret		^	^
11011111	223	$DF	_	Underline		_	_

Table H-13. *Lowercase Characters, High Bit On*

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
11100000	224	$E0	`	Open Quote		`	`
11100001	225	$E1	a			a	a
11100010	226	$E2	b			b	b
11100011	227	$E3	c			c	c
11100100	228	$E4	d			d	d
11100101	229	$E5	e			e	e
11100110	230	$E6	f			f	f
11100111	231	$E7	g			g	g
11101000	232	$E8	h			h	h
11101001	233	$E9	i			i	i
11101010	234	$EA	j			j	j
11101011	235	$EB	k			k	k
11101100	236	$EC	l			l	l
11101101	237	$ED	m			m	m
11101110	238	$EE	n			n	n
11101111	239	$EF	o			o	o
11110000	240	$F0	p			p	p
11110001	241	$F1	q			q	q
11110010	242	$F2	r			r	r
11110011	243	$F3	s			s	s
11110100	244	$F4	t			t	t
11110101	245	$F5	u			u	u
11110110	246	$F6	v			v	v
11110111	247	$F7	w			w	w
11111000	248	$F8	x			x	x
11111001	249	$F9	y			y	y
11111010	250	$FA	z			z	z
11111011	251	$FB	{	Opening Brace		{	{
11111100	252	$FC	|	Vertical Line		|	|
11111101	253	$FD	}	Closing Brace		}	}
11111110	254	$FE	~	Overline (Tilde)			
11111111	255	$FF	DEL	Delete (Rubout)	DELETE	DEL	DEL

Firmware Listings

Appendix I comprises a listing of the source code for the Monitor, enhanced video firmware, and input/output firmware contained in the Apple IIc.

```
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:  002D          39 V2        EQU   $2D          ;temp for lo-res graphics
C100:  002D          40 RMNEM     EQU   $2D          ;temp for mnemonic decoding
C100:  002E          41 MASK      EQU   $2E          ;color mask for lo-res gr.
C100:  002E          42 FORMAT    EQU   $2E          ;temp for opcode decode
C100:  002F          43 LENGTH    EQU   $2F          ;temp for opcode decode
C100:  0030          44 COLOR     EQU   $30          ;color for lo-res graphics
C100:  0031          45 MODE      EQU   $31          ;Monitor mode
C100:  0032          46 INVFLG    EQU   $32          ;normal/inverse(/flash)
C100:  0033          47 PROMPT    EQU   $33          ;prompt character
C100:  0034          48 YSAV      EQU   $34          ;position in Monitor command
C100:  0035          49 YSAV1     EQU   $35          ;temp for Y register
C100:  0036          50 CSWL      EQU   $36          ;character output hook
C100:  0037          51 CSWH      EQU   $37
C100:  0038          52 KSWL      EQU   $38          ;character input hook
C100:  0039          53 KSWH      EQU   $39
C100:  003A          54 PCL       EQU   $3A          ;temp for program counter
C100:  003B          55 PCH       EQU   $3B
C100:  003C          56 A1L       EQU   $3C          ;Monitor temp
C100:  003D          57 A1H       EQU   $3D          ;Monitor temp
C100:  003E          58 A2L       EQU   $3E          ;Monitor temp
C100:  003F          59 A2H       EQU   $3F          ;Monitor temp
```

```
C100:          0040   60 A3L        EQU   $40          ;Monitor temp
C100:          0041   61 A3H        EQU   $41          ;Monitor temp
C100:          0042   62 A4L        EQU   $42          ;Monitor temp
C100:          0043   63 A4H        EQU   $43          ;Monitor temp
C100:          0044   64 A5L        EQU   $44          ;Monitor temp
C100:          0045   65 A5H        EQU   $45          ;Monitor temp
C100:                 66 *
C100:                 67 * Note: In Apple II, //e, both interrupts and BRK destroyed
C100:                 68 * location $45.  Now only BRK destroys $45 (ACC) and it
C100:                 69 * also destroys $44 (MACSTAT).
C100:                 70 *
C100:          0044   71 MACSTAT    EQU   $44          ;Machine state after BRK
C100:          0045   72 ACC        EQU   $45          ;Acc after BRK
C100:                 73 *
C100:          0046   74 XREG       EQU   $46          ;X reg after break
C100:          0047   75 YREG       EQU   $47          ;Y reg after break
C100:          0048   76 STATUS     EQU   $48          ;P reg after break
C100:          0049   77 SPNT       EQU   $49          ;SP after break
C100:          004E   78 RNDL       EQU   $4E          ;random counter low
C100:          004F   79 RNDH       EQU   $4F          ;random counter high
C100:                 80 *
C100:                 81 * Value equates
C100:                 82 *
C100:          0006   83 GOODF8     EQU   $06          ;value of //e, lolly ID byte
C100:          0095   84 PICK       EQU   $95          ;CONTROL-U character
C100:          009B   85 ESC        EQU   $9B          ;what ESC generates
C100:                 86 *
C100:                 87 * Characters read by GETLN are placed in
C100:                 88 * IN, terminated by a carriage return.
C100:                 89 *
C100:          0200   90 IN         EQU   $0200        ;input buffer for GETLN
C100:                 91 *
C100:                 92 * Page 3 vectors
C100:                 93 *
C100:          03F0   94 BRKV       EQU   $03F0        ;vectors here after break
C100:          03F2   95 SOFTEV     EQU   $03F2        ;vector for warm start
C100:          03F4   96 PWREDUP    EQU   $03F4        ;THIS MUST = EOR #$A5 OF SOFTEV+1
C100:          03F5   97 AMPERV     EQU   $03F5        ;APPLESOFT & EXIT VECTOR
C100:          03F8   98 USRADR     EQU   $03F8        ;APPLESOFT USR function vector
C100:          03FB   99 NMI        EQU   $03FB        ;NMI vector
C100:          03FE  100 IRQLOC     EQU   $03FE        ;Maskable interrupt vector
C100:          0400  101 LINE1      EQU   $0400        ;first line of text screen
C100:          07F8  102 MSLOT      EQU   $07F8        ;owner of $C8 space
C100:                103 *
C100:                104 * HARDWARE EQUATES
C100:                105 *
C100:          C000  106 IOADR      EQU   $C000        ;for IN#, PR# vector
C100:          C000  107 KBD        EQU   $C000        ;>127 if keystroke
C100:          C000  108 CLR80COL   EQU   $C000        ;disable 80 column store
C100:          C001  109 SET80COL   EQU   $C001        ;enable 80 column store
C100:          C002  110 RDMAINRAM  EQU   $C002        ;read from main 48K RAM
C100:          C003  111 RDCARDRAM  EQU   $C003        ;read from alt. 48K RAM
C100:          C004  112 WRMAINRAM  EQU   $C004        ;write to main 48K RAM
C100:          C005  113 WRCARDRAM  EQU   $C005        ;write to alt. 48K RAM
C100:          C008  114 SETSTDZP   EQU   $C008        ;use main zero page/stack
C100:          C009  115 SETALTZP   EQU   $C009        ;use alt. zero page/stack
C100:          C00C  116 CLR80VID   EQU   $C00C        ;disable 80 column hardware
C100:          C00D  117 SET80VID   EQU   $C00D        ;enable 80 column hardware
```

```
C100:          C00E  118 CLRALTCHAR EQU  $C00E          ;normal LC, flashing UC
C100:          C00F  119 SETALTCHAR EQU  $C00F          ;normal inverse, LC; no flash
C100:          C010  120 KBDSTRB    EQU  $C010          ;turn off key pressed flag
C100:          C011  121 RDLCBNK2   EQU  $C011          ;>127 if LC bank 2 is in
C100:          C012  122 RDLCRAM    EQU  $C012          ;>127 if LC RAM read enabled
C100:          C013  123 RDRAMRD    EQU  $C013          ;>127 if reading main 48K
C100:          C014  124 RDRAMWRT   EQU  $C014          ;>127 if writing main 48K
C100:          C016  125 RDALTZP    EQU  $C016          ;>127 if Alt ZP and LC switched in
C100:          C018  126 RD80COL    EQU  $C018          ;>127 if 80 column store
C100:          C019  127 RDVBLBAR   EQU  $C019          ;>127 if not VBL
C100:          C01A  128 RDTEXT     EQU  $C01A          ;>127 if text (not graphics)
C100:          C01B  129 RDMIX      EQU  $C01B          ;>127 if mixed mode on
C100:          C01C  130 RDPAGE2    EQU  $C01C          ;>127 if TXTPAGE2 switched in
C100:          C01D  131 RDHIRES    EQU  $C01D          ;>127 if HIRES is on
C100:          C01E  132 ALTCHARSET EQU  $C01E          ;>127 if alternate char set in use
C100:          C01F  133 RD80VID    EQU  $C01F          ;>127 if 80 column hardware in
C100:          C020  134 TAPEOUT    EQU  $C020          ;what is this??
C100:          C030  135 SPKR       EQU  $C030          ;clicks the speaker
C100:          C050  136 TXTCLR     EQU  $C050          ;switch in graphics (not text)
C100:          C051  137 TXTSET     EQU  $C051          ;switch in text (not graphics)
C100:          C052  138 MIXCLR     EQU  $C052          ;clear mixed-mode
C100:          C053  139 MIXSET     EQU  $C053          ;set mixed-mode (4 lines text)
C100:          C054  140 TXTPAGE1   EQU  $C054          ;switch in text page 1
C100:          C055  141 TXTPAGE2   EQU  $C055          ;switch in text page 2
C100:          C056  142 LORES      EQU  $C056          ;low-resolution graphics
C100:          C057  143 HIRES      EQU  $C057          ;high-resolution graphics
C100:          C058  144 CLRAN0     EQU  $C058
C100:          C059  145 SETAN0     EQU  $C059
C100:          C05A  146 CLRAN1     EQU  $C05A
C100:          C05B  147 SETAN1     EQU  $C05B
C100:          C05C  148 CLRAN2     EQU  $C05C
C100:          C05D  149 SETAN2     EQU  $C05D
C100:          C05E  150 CLRAN3     EQU  $C05E
C100:          C05F  151 SETAN3     EQU  $C05F
C100:          C060  152 RD40SW     EQU  $C060          ;>127 if 40/80 switch in 40 pos
C100:          C061  153 BUTN0      EQU  $C061          ;open apple key
C100:          C062  154 BUTN1      EQU  $C062          ;closed apple key
C100:          C064  155 PADDL0     EQU  $C064          ;read paddle 0
C100:          C070  156 PTRIG      EQU  $C070          ;trigger the paddles
C100:          C081  157 ROMIN      EQU  $C081          ;switch in $D000-$FFFF ROM
C100:          C083  158 LCBANK2    EQU  $C083          ;switch in LC bank 2
C100:          C08B  159 LCBANK1    EQU  $C08B          ;switch in LC bank 1
C100:          CFFF  160 CLRROM     EQU  $CFFF          ;switch out $C8 ROMs
C100:          E000  161 BASIC      EQU  $E000          ;BASIC entry point
C100:          E003  162 BASIC2     EQU  $E003          ;BASIC warm entry point
C100:                163 *
C100:          04FB  164 VMODE      EQU  $4F8+3         ;OPERATING MODE
C100:                165 *
C100:                166 * BASIC VMODE BITS
C100:                167 *
C100:                168 * 1....... - BASIC active
C100:                169 * 0....... - Pascal active
C100:                170 * .0......
C100:                171 * .1......
C100:                172 * ..0..... - Print control characters
C100:                173 * ..1..... - Don't print ctrl chars
C100:                174 * ...0.... -
C100:                175 * ...1.... -
```

```
C100:                176 * ....0... - Print control characters
C100:                177 * ....1... - Don't print ctrl chars.
C100:                178 * .....0.. -
C100:                179 * .....1.. -
C100:                180 * ......0. -
C100:                181 * ......1. -
C100:                182 * .......0 - Print mouse characters
C100:                183 * .......1 - Don't print mouse characters
C100:                184 *
C100:          0040  185 M.40       EQU   $40
C100:          0020  186 M.CTL2     EQU   $20          ;Don't print controls
C100:          0008  187 M.CTL      EQU   $08          ;Don't print controls
C100:          0001  188 M.MOUSE    EQU   $01          ;Don't print mouse chars
C100:                189 *
C100:                190 * Pascal Mode Bits
C100:                191 *
C100:                192 * 1.......  - BASIC active
C100:                193 * 0.......  - Pascal active
C100:                194 * .0......
C100:                195 * .1......
C100:                196 * ..0..... -
C100:                197 * ..1..... -
C100:                198 * ...0....  - Cursor always on
C100:                199 * ...1....  - Cursor always off
C100:                200 * ....0...  - GOTOXY n/a
C100:                201 * ....1...  - GOTOXY in progress
C100:                202 * .....0..  - Normal Video
C100:                203 * .....1..  - Inverse Video
C100:                204 * ......0. -
C100:                205 * ......1. -
C100:                206 * .......0  - Print mouse chars
C100:                207 * .......1  - Don't print mouse chars
C100:                208 *
C100:          0080  209 M.PASCAL   EQU   $80          ;Pascal active
C100:          0010  210 M.CURSOR   EQU   $10          ;Don't print cursor
C100:          0008  211 M.GOXY     EQU   $08          ;GOTOXY IN PROGRESS
C100:          0004  212 M.VMODE    EQU   $04
C100:                213 *
C100:          0478  214 ROMSTATE   EQU   $478         ;temp store of ROM state
C100:          04F8  215 TEMP1      EQU   $4F8         ;used by CTLCHAR
C100:          0578  216 TEMPA      EQU   $578         ;used by scroll
C100:          05F8  217 TEMPY      EQU   $5F8         ;used by scroll
C100:                218 *
C100:          047B  219 OLDCH      EQU   $478+3       ;last value of CH
C100:          057B  220 OURCH      EQU   $578+3       ;80-COL CH
C100:          05FB  221 OURCV      EQU   $5F8+3       ;CURSOR VERTICAL
C100:          067B  222 VFACTV     EQU   $678+3       ;Bit7=video firmware inactive
C100:          06FB  223 XCOORD     EQU   $6F8+3       ;X-COORD (GOTOXY)
C100:          077B  224 NXTCUR     EQU   $778+3       ;next cursor to display
C100:          07FB  225 CURSOR     EQU   $7F8+3       ;the current cursor char
C100:                 17           INCLUDE SERIAL     ;Equates for serial code
```

```
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 ****************************************
C100:  C100    34 serslot   equ   $C100
C100:  C200    35 comslot   equ   $C200
C100:              36           MSB   ON
C100:  00BF    37 cmdcur    equ   '?'          ;Cursor while on command mode
C100:  00DF    38 termcur   equ   ' '          ;Cursor while in terminal mode
C100:              39           MSB   OFF
C100:  0091    40 xon       equ   $91          ;XON character
C100:  03B8    41 sermode   equ   $3B8         ;D7=1 if in cmd; D6=1 if term 479 & 47A
C100:  0438    42 astat     equ   $438         ;Acia status from int 4F9 & 4FA
C100:  04B8    43 pwdth     equ   $4B8         ;Printer width 579 & 57A
C100:  0538    44 extint    equ   $538         ;extint & typhed enable 5F9 & 5FA
C100:  05F9    45 extint2   equ   $5F9
C100:  05FA    46 typhed    equ   $5FA
C100:  0679    47 oldcur    equ   $679         ;Saves cursor while in command
C100:  067A    48 oldcur2   equ   $67A         ;Saves cursor while in terminal mode
C100:  0638    49 eschar    equ   $638         ;Current escape character 6F9 & 6FA
C100:  06B8    50 flags     equ   $6B8         ;D7 = Video echo D6 = CRLF 779 & 77A
C100:  0738    51 col       equ   $738         ;Current printer column 7F9 & 7FA
C100:  047F    52 number    equ   $47F         ;Number accumulated in command
C100:  04FF    53 aciabuf   equ   $4FF         ;Owner of serial buffer
C100:  057F    54 twser     equ   $57F         ;Storage pointer for serial buffer
C100:  05FF    55 twkey     equ   $5FF         ;Storage pointer for type ahead buffer
C100:  067F    56 trser     equ   $67F         ;Retrieve pointer for serial buffer
C100:  06FF    57 trkey     equ   $6FF         ;Retrieve buffer for type ahead buffer
C100:  0800    58 thbuf     equ   $800         ;Buffer in alt ram space
C100:  06F8    59 temp      equ   $6F8         ;Temp storage
C100:  BFF8    60 sdata     equ   $BFF8        ;+$N0+$90 is output port
```

```
C100:          BFF9   61 sstat     equ   $BFF9          ;ACIA status register
C100:          BFFA   62 scomd     equ   $BFFA          ;ACIA command register
C100:          BFFB   63 scntl     equ   $BFFB          ;ACIA control register
C100:          FF58   64 iorts     equ   $FF58          ;RTS opcode
C100:                 18           INCLUDE SER          ;Printer port @ $C100
```

```
C100:                    3 *org serslot
C100:2C 58 FF            4           bit   iorts          ;Set V to indicate initial entry
C103:70 0C   C111        5           bvs   entrl          ;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   entrl          ;Always taken

C10B:01                 12           dfb   $01            ;pascal signiture byte
C10C:31                 13           dfb   $31            ;device signiture
C10D:E4                 14           dfb   >plinit
C10E:EE                 15           dfb   >plread
C10F:F6                 16           dfb   >plwrite
C110:FB                 17           dfb   >plstatus

C111:DA                 19 entrl     phx                   ;Save the reg
C112:A2 C1              20           ldx   #<serslot      ;X = Cn
C114:4C 33 C2           21           jmp   setup          ;Set mslot, etc
C117:90 05   C11E       22 serport   bcc   serisout       ;Only output allowed
C119:20 4D CE           23           jsr   zzquit         ;Reset the hooks
C11C:80 6A   C188       24           bra   done
C11E:0A                 25 serisout  asl   A              ;A = flags
C11F:7A                 26           ply                   ;Get char
C120:5A                 27           phy
C121:BD B8 04           28           lda   pwdth,x        ;Formatting enabled?
C124:F0 42   C168       29           beq   prnow
C126:A5 24              30           lda   ch             ;Get current horiz position
C128:B0 1C   C146       31           bcs   servid         ;Branch if video echo
C12A:DD B8 04           32           cmp   pwdth,x        ;If CH >= PWIDTH, then CH = COL
C12D:90 03   C132       33           bcc   chok
C12F:BD 38 07           34           lda   col,x
C132:DD 38 07           35 chok      cmp   col,x          ;Must be > col for valid tab
C135:B0 0B   C142       36           bcs   fixch          ;Branch if ok
C137:C9 11              37           cmp   #$11           ;8 or 16?
C139:B0 11   C14C       38           bcs   prnt           ;If > forget it
C13B:09 F0              39           ora   #$F0           ;Find next comma cheaply
C13D:3D 38 07           40           and   col,x          ;Don't blame me it's Dick's trick
C140:65 24              41           adc   ch
C142:85 24              42 fixch     sta   ch             ;Save the new position
C144:80 06   C14C       43           bra   prnt
C146:C5 21              44 servid    cmp   wndwdth        ;If ch>= wndwdth go back to start of line
C148:90 02   C14C       45           blt   prnt
C14A:64 24              46           stz   ch             ;Go back to left edge


C14C:                   48 * We have a char to print
C14C:7A                 49 prnt      ply
C14D:5A                 50           phy
C14E:BD 38 07           51           lda   col,x          ;Have we exceeded width?
C151:DD B8 04           52           cmp   pwdth,x
C154:B0 08   C15E       53           bge   toofar
C156:C5 24              54           cmp   ch             ;Are we tabbing?
C158:B0 0E   C168       55           bge   prnow
C15A:A9 40              56           lda   #$40           ;Space * 2
C15C:80 02   C160       57           bra   tab
C15E:A9 1A              58 toofar    lda   #$1A           ;CR * 2
```

```
C160:C0 80          59 tab       cpy   #$80          ;C = High bit
C162:6A             60           ror   A             ;Shift it into char
C163:20 9D C1       61           jsr   serout3       ;Out it goes
C166:80 E4   C14C   62           bra   prnt
C168:98             63 prnow     tya
C169:20 8C C1       64           jsr   serout        ;Print the actual char
C16C:BD B8 04       65           lda   pwdth,x       ;Formatting enabled
C16F:F0 17   C188   66           beq   done
C171:3C B8 06       67           bit   flags,x       ;In video echo?
C174:30 12   C188   68           bmi   done
C176:BD 38 07       69           lda   col,x         ;Check if within 8 chars of right edge
C179:FD B8 04       70           sbc   pwdth,x       ;So BASIC can format output
C17C:C9 F8          71           cmp   #$F8
C17E:90 04   C184   72           bcc   setch         ;If not within 8, we're done
C180:18             73           clc
C181:65 21          74           adc   wndwdth
C183:AC             75           dfb   $AC           ;Dummy LDY to skip next two bytes
C184:A9 00          76 setch     lda   #0            ;Keep cursor at 0 if video off
C186:85 24          77           sta   ch
C188:68             78 done      pla                  ;Restore regs
C189:7A             79           ply
C18A:FA             80           plx
C18B:60             81 socmd     rts


C18C:       C18C    83 serout    equ   *             ;Serial output
C18C:20 EB C9       84           jsr   command       ;Check if command
C18F:90 FA   C18B   85           bcc   socmd         ;All done if it is
C191:       C191    86 serout2   equ   *
C191:3C B8 06       87           bit   flags,x       ;N=1 iff video on
C194:10 07   C19D   88           bpl   serout3
C196:C9 91          89           cmp   #xon          ;Don't echo ^Q
C198:F0 03   C19D   90           beq   serout3
C19A:20 F0 FD       91           jsr   coutl         ;Echo it
C19D:       C19D    92 serout3   equ   *
C19D:BC 85 C8       93           ldy   devno,x       ;Y points to ACIA
C1A0:48             94           pha                  ;Save the char
C1A1:2C 58 FF       95           bit   iorts         ;Control char?
C1A4:F0 03   C1A9   96           beq   sordy         ;Don't inc column if so
C1A6:FE 38 07       97           inc   col,x
C1A9:08             98 sordy     php                 ;can't have real interrupts for a while
C1AA:78             99           sei
C1AB:B9 F9 BF      100           lda   sstat,y       ;Check XMIT empty & DCD
C1AE:10 11   C1C1  101           bpl   sordy2        ;branch if not clearing an interrupt
C1B0:48            102           pha                 ;save original status
C1B1:5A            103           phy
C1B2:2C 14 C0      104           bit   rdramwrt      ;Save state of aux ram
C1B5:08            105           php
C1B6:20 1C C9      106           jsr   aitst2
C1B9:28            107           plp
C1BA:10 03   C1BF  108           bpl   somain        ;Branch if was main
C1BC:8D 05 C0      109           sta   wrcardram     ;Was alt ram
C1BF:7A            110 somain    ply
C1C0:68            111           pla
C1C1:28            112 sordy2    plp
C1C2:29 30         113           and   #$30
C1C4:C9 10         114           cmp   #$10
```

```
C1C6:D0 E1    C1A9  115           bne   sordy
C1C8:68             116           pla
C1C9:48             117           pha                  ;Get char to XMIT
C1CA:99 F8 BF       118           sta   sdata,y        ;Out it goes
C1CD:3C B8 06       119           bit   flags,x        ;V=1 if LF after CR
C1D0:49 0D          120           eor   #$0D           ;check for CR.
C1D2:0A             121           asl   A              ;preserve bit 7
C1D3:D0 0D    C1E2  122           bne   sodone         ;branch if not CR.
C1D5:50 06    C1DD  123           bvc   clrcol         ;branch if no LF after CR
C1D7:A9 14          124           lda   #$14           ;Get LF*2
C1D9:6A             125           ror   A              ;no shift in high bit
C1DA:20 9D C1       126           jsr   serout3        ;Output the LF but don't echo it
C1DD:64 24          127 clrcol    stz   ch             ;0 position & column
C1DF:9E 38 07       128           stz   col,x
C1E2:68             129 sodone    pla                  ;Get the char back
C1E3:60             130           rts
```

```
C1E4:                132 * Pascal support stuff

C1E4:48              134 plinit    pha
C1E5:20 C8 C2        135           jsr   default       ;set defaults, enable acia
C1E8:9E B8 06        136           stz   flags,x
C1EB:68              137           pla
C1EC:80 05   C1F3    138           bra   plread2       ;all done...

C1EE:20 C5 C8        140 plread    jsr   XRDSER        ;read data from serial port (or buffer)
C1F1:90 FB   C1EE    141           bcc   plread        ;Branch if data not ready
C1F3:A2 00           142 plread2   ldx   #0
C1F5:60              143           rts

C1F6:20 8C C1        145 plwrite   jsr   serout        ;Go output character
C1F9:80 F8   C1F3    146           bra   plread2
C1FB:80 1A   C217    147 plstatus  bra   p2status

C1FD:        0003    149           ds    comslot-*,$00
C200:                 19           INCLUDE COMM        ;Communications port @ $C200
```

```
C200:2C 58 FF        3           bit   iorts         ;Set V to indicate initial entry
C203:70 2B   C230    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 25   C230    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 D1   C1E4   20 p2init     bra   plinit
C213:80 D9   C1EE   21 p2read     bra   plread
C215:80 DF   C1F6   22 p2write    bra   plwrite

C217:        C217   24 p2status   equ   *
C217:A2 40          25           ldx   #$40          ;anticipate bad status request
C219:4A             26           lsr   a             ;shift request to carry
C21A:D0 12   C22E   27           bne   notrdy
C21C:AA             28           tax                 ;clear x for no error return code
C21D:A9 08          29           lda   #8            ;anticpate input ready request
C21F:B0 01   C222   30           bcs   pstat2        ;branch if good guess.
C221:0A             31           asl   a
C222:09 20          32 pstat2    ora   #$20          ;include DCD in test
C224:39 89 C0       33           and   sstat+$90,y
C227:F0 05   C22E   34           beq   notrdy        ;branch if not ready for I/O
C229:49 20          35           eor   #$20
C22B:38             36           sec                 ;assume port is ready
C22C:D0 01   C22F   37           bne   isrdy         ;branch if good assumption
C22E:18             38 notrdy    clc                 ;indicate acia not ready for I/O
C22F:60             39 isrdy     rts

C230:DA             41 entr      phx
C231:A2 C2          42           ldx   #<comslot     ;X = <CN00
C233:        C233   43 setup     equ   *
C233:5A             44           phy
C234:48             45           pha
C235:8E F8 07       46           stx   mslot
C238:50 22   C25C   47           bvc   sudone        ;First call?
C23A:A5 36          48           lda   cswl          ;If both hooks CN00 setup defaults
C23C:45 38          49           eor   kswl
C23E:F0 06   C246   50           beq   sudodef
C240:A5 37          51           lda   cswh          ;If both hooks CN then don't do def
C242:C5 39          52           cmp   kswh          ;since it has already been done
C244:F0 03   C249   53           beq   sunodef
C246:20 C8 C2       54 sudodef   jsr   default       ;Set up defaults
C249:8A             55 sunodef   txa
C24A:45 39          56           eor   kswh          ;Input call?
C24C:05 38          57           ora   kswl
C24E:D0 07   C257   58           bne   suout         ;Must be Cn00
C250:A9 05          59           lda   #>sin         ;Fix the input hook
C252:85 38          60           sta   kswl
```

```
C254:38                61            sec                  ;C = 1 for input call
C255:80 05   C25C      62            bra   sudone
C257:A9 07             63 suout      lda   #>sout         ;Fix output hook
C259:85 36             64            sta   cswl           ;Note C might not be 0
C25B:18                65            clc                  ;C=0 for output
C25C:BD B8 06          66 sudone     lda   flags,x        ;Check if serial or comm port
C25F:89 01             67            bit   #1             ;Leave flags in a for serport
C261:D0 03   C266      68            bne   commport
C263:4C 17 C1          69 comout     jmp   serport
C266:90 FB   C263      70 commport   bcc   comout         ;Output?
C268:68                71            pla                  ;Get the char
C269:80 28   C293      72            bra   terml          ;Input
C26B:3C B8 03          73 noesc      bit   sermode,x      ;In terminal mode?
C26E:50 1C   C28C      74            bvc   exitl          ;If not, return key
C270:20 91 C1          75            jsr   serout2        ;Out it goes
C273:80 1E   C293      76            bra   terml
C275:        C275      77 testkbd    equ   *
C275:68                78            pla                  ;Get current char
C276:20 70 CC          79            jsr   update         ;Update cursor & check keyboard
C279:10 1B   C296      80            bpl   serin          ;N=0 if no new key
C27B:20 EB C9          81            jsr   command        ;Test for command
C27E:B0 EB   C26B      82            bcs   noesc          ;Branch if not
C280:29 5F             83            and   #$5f           ;upshift for following tests
C282:C9 51             84            cmp   #'Q'           ;Quit?
C284:F0 04   C28A      85            beq   exitX
C286:C9 52             86            cmp   #'R'           ;Reset?
C288:D0 09   C293      87            bne   terml          ;Go check serial
C28A:A9 98             88 exitX      lda   #$98           ;return a CTRL-X
C28C:7A                89 exitl      ply
C28D:FA                90            plx
C28E:60                91            rts
C28F:18                92 goremote   clc                  ;Into remote mode
C290:20 CD CA          93 goterm     jsr   setterm        ;Into terminal mode
C293:        C293      94 terml      equ   *
C293:20 4C CC          95            jsr   showcur        ;Get current char on screen
C296:48                96 serin      pha
C297:20 C5 C8          97            jsr   XRDSER         ;Is it ready?
C29A:90 D9   C275      98            bcc   testkbd        ;If not, try the keyboard
C29C:A8                99            tay                  ;Save new input in y for now
C29D:68               100            pla
C29E:5A               101            phy                  ;Save new char on stack
C29F:20 B8 C3         102            jsr   storch         ;Fix the screen
C2A2:68               103            pla                  ;Get the new data
C2A3:BC 38 06         104            ldy   eschar,x       ;If 0, don't modify char
C2A6:F0 16   C2BE     105            beq   sinomod
C2A8:09 80            106            ora   #$80           ;Apple loves the high bit
C2AA:C9 8A            107            cmp   #$8A           ;Ignore line feed
C2AC:F0 E5   C293     108            beq   terml
C2AE:C9 91            109            cmp   #xon
C2B0:F0 E1   C293     110            beq   terml          ;Ignore ^Q
C2B2:C9 FF            111            cmp   #$FF           ;Ignore FFs
C2B4:F0 DD   C293     112            beq   terml
C2B6:C9 92            113            cmp   #$92           ;^R for remote?
C2B8:F0 D5   C28F     114            beq   goremote
C2BA:C9 94            115            cmp   #$94           ;^T for terminal mode?
C2BC:F0 D2   C290     116            beq   goterm
C2BE:3C B8 03         117 sinomod   bit   sermode,x      ;In terminal mode?
C2C1:50 C9   C28C     118            bvc   exitl          ;Return to user if not A = char
```

```
C2C3:20 ED FD      119           jsr   cout          ;Onto the screen with it
C2C6:80 CB   C293  120           bra   terml
C2C8:        C2C8  121 default   equ   *             ;Set up the defaults
C2C8:20 A2 C8      122           jsr   moveirq       ;make sure irq vectors ok
C2CB:BC 3B C2      123           ldy   defidx-$C1,x  ;Index into alt screen. Table in command
C2CE:20 7C C3      124 defloop   jsr   getalt        ;Get default from alt screen
C2D1:48            125           pha
C2D2:88            126           dey
C2D3:30 04   C2D9  127           bmi   defff         ;Done if minus
C2D5:C0 03         128           cpy   #3
C2D7:D0 F5   C2CE  129           bne   defloop       ;Or if 2
C2D9:20 A2 C8      130 defff     jsr   moveirq       ;Jam irq vector into LC
C2DC:68            131           pla                  ;Command, control & flags on stack
C2DD:BC 85 C8      132           ldy   devno,x
C2E0:99 FB BF      133           sta   scntl,y       ;Set command reg
C2E3:68            134           pla
C2E4:99 FA BF      135           sta   scomd,y
C2E7:68            136           pla
C2E8:9D B8 06      137           sta   flags,x       ;And the flags
C2EB:29 01         138           and   #1            ;A = $01 (^A) if comm mode
C2ED:D0 02   C2F1  139           bne   defcom
C2EF:A9 09         140           lda   #9            ;^I for serial port
C2F1:9D 38 06      141 defcom    sta   eschar,x
C2F4:68            142           pla                  ;Get printer width
C2F5:9D B8 04      143           sta   pwdth,x
C2F8:9E B8 03      144           stz   sermode,x
C2FB:60            145           rts
C2FC:03 07         146 defidx    dfb   3,7
C2FE:        0002  147           ds    $C300-*,$00
C300:                20           INCLUDE C3SPACE     ;80 column card @ $C300
```

```
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 86 CF           30           JMP   MOVEAUX       ;MEMORY MOVE ACROSS BANKS
C314:4C CD CF           31           JMP   XFER          ;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   COUTZ         ;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?
```

```
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:                21           INCLUDE MOUSE         ;Equates for the mouse
```

```
C400:                2           MSB   ON
C400:                3 ***************************************
C400:                4 *
C400:                5 * Mouse firmware for the Chels
C400:                6 *
C400:                7 *  by Rich Williams
C400:                8 *  July, 1983
C400:                9 *
C400:               10 ***************************************


C400:               12 ***************************************
C400:               13 *
C400:               14 * Equates
C400:               15 *
C400:               16 ***************************************


C400:               18 * Input bounds are in scratch area
C400:  0478         19 moutemp   equ   $478          ;Temporary storage
C400:  0478         20 minl      equ   $478
C400:  04F8         21 maxl      equ   $4F8
C400:  0578         22 minh      equ   $578
C400:  05F8         23 maxh      equ   $5F8
C400:               24 * Mouse bounds in slot 5 screen area
C400:  047D         25 minxl     equ   $47D
C400:  04FD         26 minyl     equ   $4FD
C400:  057D         27 minxh     equ   $57D
C400:  05FD         28 minyh     equ   $5FD
C400:  067D         29 maxxl     equ   $67D
C400:  06FD         30 maxyl     equ   $6FD
C400:  077D         31 maxxh     equ   $77D
C400:  07FD         32 maxyh     equ   $7FD
C400:               33 * Mouse holes in slot 4 screen area
C400:  047C         34 mouxl     equ   $47C          ;X position low byte
C400:  04FC         35 mouyl     equ   $4FC          ;Y position low byte
C400:  057C         36 mouxh     equ   $57C          ;X position high byte
C400:  05FC         37 mouyh     equ   $5FC          ;Y position high byte
C400:  067C         38 mouarm    equ   $67C          ;Arm interrupts from movement or button
C400:  077C         39 moustat   equ   $77C          ;Mouse status
C400:               40 * Moustat provides the following
C400:               41 *  D7= Button pressed
C400:               42 *  D6= Status of button on last read
C400:               43 *  D5= Moved since last read
C400:               44 *  D4= Reserved
C400:               45 *  D3= Interrupt from VBL
C400:               46 *  D2= Interrupt from button
C400:               47 *  D1= Interrupt from movement
C400:               48 *  D0= Reserved
C400:  07FC         49 moumode   equ   $7FC          ;Mouse mode
C400:               50 *  D7-D4= Unused
C400:               51 *  D3= VBL active
C400:               52 *  D2= VBL interrupt on button
C400:               53 *  D1= VBL interrupt on movement
C400:               54 *  D0= Mouse active
C400:  0020         55 movarm    equ   $20
```

```
C400:          000C   56 vblmode   equ   $0C
C400:          0004   57 butmode   equ   $04          ;D2 mask
C400:          0002   58 movmode   equ   $02          ;D1 mask


C400:                 60 * Hardware addresses
C400:          C015   61 mouxint   equ   $C015        ;D7 = x interrupt
C400:          C017   62 mouyint   equ   $C017        ;D7 = y interrupt
C400:          C019   63 vblint    equ   $C019        ;D7 = vbl interrupt
C400:          C078   64 ioudsbl   equ   $C078        ;Disable iou access
C400:          C079   65 iouenbl   equ   $C079        ;Enable iou access
C400:          C048   66 mouclr    equ   $C048        ;Clear mouse interrupt
C400:          C058   67 iou       equ   $C058        ;IOU interrupt switches
C400:          C058   68 moudsbl   equ   $C058        ;Disable mouse interrupts
C400:          C059   69 mouenbl   equ   $C059        ;Enable mouse interrupts
C400:          C063   70 moubut    equ   $C063        ;D7 = Mouse button
C400:          C066   71 mouxl     equ   $C066        ;D7 = X1
C400:          C067   72 mouyl     equ   $C067        ;D7 = Y1
C400:          C070   73 vblclr    equ   $C070        ;Clear VBL interrupt
C400:                 74 *
C400:                 75 * Other addresses
C400:                 76 *
C400:          0200   77 inbuf     equ   $200         ;Input buffer
C400:          0214   78 binl      equ   inbuf+20     ;Temp for binary conversion
C400:          0215   79 binh      equ   inbuf+21
C400:                 22          INCLUDE MCODE      ;Mouse @ $C400
```

```
C400:                2 ****************************************
C400:                3 *
C400:                4 * Entry points for mouse firmware
C400:                5 *
C400:                6 ****************************************
C400:80 05   C407    7 mbasic    bra   outent
C402:A2 03           8 pnull     ldx   #3
C404:60              9           rts                  ;Null for pascal entry
C405:38             10 inent     sec                  ;Signature bytes
C406:90             11           dfb   $90
C407:18             12 outent    clc
C408:4C 80 C7       13           jmp   xmbasic        ;Go do basic entry
C40B:01             14           dfb   $01            ;More signature stuff
C40C:20             15           dfb   $20
C40D:02             16           dfb   >pnull
C40E:02             17           dfb   >pnull
C40F:02             18           dfb   >pnull
C410:02             19           dfb   >pnull
C411:00             20           dfb   $0
C412:3D             21           dfb   >xsetmou       ;SETMOUSE
C413:FC             22           dfb   >xmtstint      ;SERVEMOUSE
C414:95             23           dfb   >xmread        ;READMOUSE
C415:84             24           dfb   >xmclear       ;CLEARMOUSE
C416:6B             25           dfb   >noerror       ;POSMOUSE
C417:B0             26           dfb   >xmclamp       ;CLAMPMOUSE
C418:6D             27           dfb   >xmhome        ;HOMEMOUSE
C419:1C             28           dfb   >initmouse     ;INITMOUSE
C41A:02             29           dfb   >pnull
C41B:CF             30           dfb   >xmint
```

```
C41C:                32 ****************************************
C41C:                33 *
C41C:                34 * Initmouse - resets the mouse
C41C:                35 *  Also clears all of the mouse holes
C41C:                36 * note that iou access fires pdlstrb & makes mouse happy
C41C:                37 *
C41C:                38 ****************************************
C41C:          C41C  39 initmouse equ   *
C41C:9C 7C 07        40           stz   moustat        ;Clear status
C41F:A2 80           41           ldx   #$80
C421:A0 01           42           ldy   #1
C423:9E 7D 04        43 xrloop    stz   minxl,x        ;Minimum = $0000
C426:9E 7D 05        44           stz   minxh,x
C429:A9 FF           45           lda   #$FF           ;Maximum = $03FF
C42B:9D 7D 06        46           sta   maxxl,x
C42E:A9 03           47           lda   #03
C430:9D 7D 07        48           sta   maxxh,x
C433:A2 00           49           ldx   #0
C435:88              50           dey
C436:10 EB   C423    51           bpl   xrloop
C438:20 6D C4        52           jsr   xmhome         ;Clear the mouse holes
C43B:A9 00           53           lda   #0             ;Fall into SETMOU


C43D:                55 ****************************************
C43D:                56 *
C43D:                57 * XSETMOU - Sets the mouse mode to A
C43D:                58 *
C43D:                59 ****************************************
C43D:          C43D  60 xsetmou   equ   *
C43D:AA              61           tax
C43E:20 A2 C8        62           jsr   moveirq        ;Make sure interrupt vector is right
C441:8A              63           txa                   ;Only x preserved by moveirq
C442:8D 78 04        64           sta   moutemp
C445:4A              65           lsr   A              ;D0 = 1 if mouse active
C446:0D 78 04        66           ora   moutemp        ;D2 = 1 if vbl active
C449:C9 10           67           cmp   #$10           ;If >=$10 then invalid mode
C44B:B0 1F   C46C    68           bcs   sminvalid
C44D:29 05           69           and   #5             ;Extract VBL & Mouse
C44F:F0 01   C452    70           beq   xsoff          ;Turning it off?
C451:58              71           cli                   ;If not, ints active
C452:69 55           72 xsoff     adc   #$55           ;Make iou byte C=0


C454:                74 ****************************************
C454:                75 *
C454:                76 * SETIOU - Sets the IOU interrupt modes to A
C454:                77 *  Inputs: A = Bits to change
C454:                78 *  D7 = Y int on falling edge
C454:                79 *  D6 = Y int on rising edge
C454:                80 *  D5 = X int on falling edge
C454:                81 *  D4 = X int on rising edge
C454:                82 *  D3 = Enable VBL int
C454:                83 *  D2 = Disable VBL int
C454:                84 *  D1 = Enable mouse int
C454:                85 *  D0 = Disable mouse int
```

```
C454:                 86 *
C454:                 87 *
C454:                 88 ****************************************
C454:          C454   89 setiou     equ   *
C454:08               90            php
C455:78               91            sei                    ;Don't allow ints while iou enabled
C456:8E FC 07         92            stx   moumode
C459:8D 79 C0         93            sta   iouenbl          ;Enable iou access
C45C:A2 08            94            ldx   #8
C45E:CA               95 siloop     dex
C45F:0A               96            asl   A                ;Get a bit to check
C460:90 03    C465    97            bcc   sinoch           ;No change if C=0
C462:9D 58 C0         98            sta   iou,x            ;Set it
C465:D0 F7    C45E    99 sinoch     bne   siloop           ;Any bits left in A?
C467:8D 78 C0        100            sta   ioudsbl          ;Turn off iou access
C46A:28              101            plp
C46B:18              102 noerror    clc
C46C:60              103 sminvalid  rts


C46D:                105 ****************************************
C46D:                106 *
C46D:                107 * XMHOME- Clears mouse position & status
C46D:                108 *
C46D:                109 ****************************************
C46D:          C46D  110 xmhome     equ   *
C46D:A2 80           111            ldx   #$80             ;Point mouse to upper left
C46F:80 02    C473   112            bra   xmh2
C471:A2 00           113 xmhloop    ldx   #0
C473:BD 7D 04        114 xmh2       lda   minxl,x
C476:9D 7C 04        115            sta   mouxl,x
C479:BD 7D 05        116            lda   minxh,x
C47C:9D 7C 05        117            sta   mouxh,x
C47F:CA              118            dex
C480:10 EF    C471   119            bpl   xmhloop
C482:80 0C    C490   120            bra   xmcdone


C484:                122 ****************************************
C484:                123 *
C484:                124 * XMCLEAR - Sets the mouse to 0,0
C484:                125 *
C484:                126 ****************************************
C484:          C484  127 xmclear    equ   *
C484:9C 7C 04        128            stz   mouxl
C487:9C 7C 05        129            stz   mouxh
C48A:9C FC 04        130            stz   mouyl
C48D:9C FC 05        131            stz   mouyh
C490:9C 7C 06        132 xmcdone    stz   mouarm
C493:18              133            clc
C494:60              134            rts
```

```
C495:                 136 ****************************************
C495:                 137 *
C495:                 138 * XMREAD - Updates the screen holes
C495:                 139 *
C495:                 140 ****************************************
C495:          C495   141 xmread     equ   *
C495:A9 20            142           lda   #movarm       ;Has mouse moved?
C497:2D 7C 06         143           and   mouarm
C49A:1C 7C 06         144           trb   mouarm        ;Clear arm bit
C49D:2C 63 C0         145           bit   moubut        ;Button pressed?
C4A0:30 02    C4A4    146           bmi   xrbut
C4A2:09 80            147           ora   #$80
C4A4:2C 7C 07         148 xrbut     bit   moustat       ;Pressed last time?
C4A7:10 02    C4AB    149           bpl   xrbut2
C4A9:09 40            150           ora   #$40
C4AB:8D 7C 07         151 xrbut2    sta   moustat
C4AE:18               152           clc
C4AF:60               153           rts


C4B0:                 155 ****************************************
C4B0:                 156 *
C4B0:                 157 * XMCLAMP - Store new bounds
C4B0:                 158 *  Inputs A = 1 for Y, 0 for X axis
C4B0:                 159 *    minl, minh, maxl, maxh = new bounds
C4B0:                 160 *
C4B0:                 161 ****************************************
C4B0:          C4B0   162 xmclamp    equ   *
C4B0:6A               163           ror   A             ;1 -> 80
C4B1:6A               164           ror   A
C4B2:29 80            165           and   #$80
C4B4:AA               166           tax
C4B5:AD 78 04         167           lda   minl
C4B8:9D 7D 04         168           sta   minxl,x
C4BB:AD 78 05         169           lda   minh
C4BE:9D 7D 05         170           sta   minxh,x
C4C1:AD F8 04         171           lda   maxl
C4C4:9D 7D 06         172           sta   maxxl,x
C4C7:AD F8 05         173           lda   maxh
C4CA:9D 7D 07         174           sta   maxxh,x
C4CD:18               175           clc                 ;No error
C4CE:60               176           rts
```

```
C4CF:                 178 ****************************************
C4CF:                 179 *
C4CF:                 180 * Mouse interrupt handler
C4CF:                 181 *
C4CF:                 182 * MOUSEINT - Monitor's interrupt handler
C4CF:                 183 * XMINT - Interrupt handler the user can use
C4CF:                 184 * XMTSTINT - Checks mouse status bits
C4CF:                 185 ****************************************
C4CF:         C4CF    186 xmint     equ   *
C4CF:AE 66 C0         187           ldx   mouxl          ;Get X1 & Y1 asap
C4D2:AC 67 C0         188           ldy   mouyl
C4D5:         C4D5    189 mouseint  equ   *              ;Entry point if X & Y set up
C4D5:A9 0E            190           lda   #$0E           ;Clear status bits
C4D7:1C 7C 07         191           trb   moustat



C4DA:38               193           sec                   ;Assume interrupt not handled
C4DB:                 194 * Check for vertical blanking interrupt
C4DB:AD 19 C0         195           lda   vblint         ;VBL interrupt?
C4DE:10 48    C528    196           bpl   chkmou
C4E0:8D 79 C0         197           sta   iouenbl        ;Enable iou access & clear VBL interrupt
C4E3:A9 0C            198           lda   #vblmode       ;Should we leave vbl active?
C4E5:2C FC 07         199           bit   moumode
C4E8:D0 03    C4ED    200           bne   cvnovbl
C4EA:8D 5A C0         201           sta   iou+2          ;Disable VBL
C4ED:09 02            202 cvnovbl   ora   #movmode
C4EF:80 1B    C50C    203           bra   xmskip



C4F1:A9 0E            205 mistat    lda   #$0E
C4F3:2D 7C 07         206           and   moustat
C4F6:D0 01    C4F9    207           bne   nostat2
C4F8:38               208           sec
C4F9:68               209 nostat2   pla
C4FA:60               210           rts
C4FB:         0000    211           ds    $C4FB-*
C4FB:D6               212           dfb   $D6             ;Signature byte
C4FC:48               213 xmtstint  pha
C4FD:18               214           clc
C4FE:80 F1    C4F1    215           bra   mistat        ;Go check status
C500:FF               216           dfb   $FF
C501:20 4D CE         217           jsr   zzquit        ;Get out of the hooks
C504:A2 FF            218           ldx   #$FF
C506:20 24 CB         219 qloop     jsr   zznm2
C509:10 FB    C506    220           bpl   qloop
C50B:60               221           rts
C50C:         C50C    222 xmskip    equ   *
C50C:8D 78 C0         223           sta   ioudsbl
C50F:2C 7C 06         224           bit   mouarm         ;VBL bit in arm isn't used
C512:D0 02    C516    225           bne   cvmoved
C514:A9 0C            226           lda   #vblmode       ;Didn't move
C516:2C 63 C0         227 cvmoved   bit   moubut         ;Button pressed?
C519:10 02    C51D    228           bpl   cvbut
C51B:49 04            229           eor   #butmode       ;Clear the button bit
C51D:2D FC 07         230 cvbut     and   moumode        ;Which bits were set in the mode
C520:0C 7C 07         231           tsb   moustat
```

```
C523:1C 7C 06      232           trb   mouarm
C526:69 FE         233           adc   #$FE          ;C=1 if int passes to user
C528:              234 * Check & update mouse movement
C528:        C528  235 chkmou    equ   *
C528:AD 15 C0      236           lda   mouxint       ;Mouse interrupt?
C52B:0D 17 C0      237           ora   mouyint
C52E:10 6A   C59A  238           bpl   xmdone        ;If not return with C from vbl
C530:8A            239           txa                  ;Get X1 in A
C531:A2 00         240           ldx   #0
C533:2C 15 C0      241           bit   mouxint       ;X movement?
C536:30 0A   C542  242           bmi   cmxmov
C538:98            243 cmloop    tya                  ;Get Y1 into A
C539:49 80         244           eor   #$80          ;Complement direction
C53B:A2 80         245           ldx   #$80
C53D:2C 17 C0      246           bit   mouyint
C540:10 39   C57B  247           bpl   cmnoy
C542:0A            248 cmxmov    asl   A
C543:BD 7C 04      249           lda   mouxl,x       ;A = current low byte
C546:B0 1A   C562  250           bcs   cmrght        ;Which way?
C548:DD 7D 04      251           cmp   minxl,x       ;Move left
C54B:D0 08   C555  252           bne   cmlok
C54D:BD 7C 05      253           lda   mouxh,x
C550:DD 7D 05      254           cmp   minxh,x
C553:F0 22   C577  255           beq   cmnoint
C555:BD 7C 04      256 cmlok     lda   mouxl,x
C558:D0 03   C55D  257           bne   cmnt0         ;Borrow from high byte?
C55A:DE 7C 05      258           dec   mouxh,x
C55D:DE 7C 04      259 cmnt0     dec   mouxl,x
C560:80 15   C577  260           bra   cmnoint
C562:DD 7D 06      261 cmrght    cmp   maxxl,x       ;At high bound?
C565:D0 08   C56F  262           bne   cmrok
C567:BD 7C 05      263           lda   mouxh,x
C56A:DD 7D 07      264           cmp   maxxh,x
C56D:F0 08   C577  265           beq   cmnoint
C56F:FE 7C 04      266 cmrok     inc   mouxl,x       ;Move right
C572:D0 03   C577  267           bne   cmnoint
C574:FE 7C 05      268           inc   mouxh,x
C577:E0 00         269 cmnoint   cpx   #0
C579:F0 BD   C538  270           beq   cmloop
C57B:8D 48 C0      271 cmnoy     sta   mouclr
C57E:A9 02         272           lda   #movmode      ;Should we enable VBL?
C580:2D FC 07      273           and   moumode
C583:F0 09   C58E  274           beq   cmnovbl       ;Branch if not
C585:8D 79 C0      275           sta   iouenbl
C588:8D 5B C0      276           sta   iou+3         ;Enable VBL int
C58B:8D 78 C0      277           sta   ioudsbl
C58E:09 20         278 cmnovbl   ora   #movarm       ;Mark that we moved
C590:0C 7C 06      279           tsb   mouarm
C593:A9 0E         280           lda   #$0E
C595:2D 7C 07      281           and   moustat
C598:69 FE         282           adc   #$FE          ;C=1 iff any bits were 1
C59A:60            283 xmdone    rts
```

```
C59B:                  285 ****************************************
C59B:                  286 *
C59B:                  287 * HEXTODEC - Puts +0000, into the input buffer
C59B:                  288 *  inputs: A = Low byte of number
C59B:                  289 *          X = High byte of number
C59B:                  290 *          Y = Position of ones digit
C59B:                  291 *
C59B:                  292 ****************************************
C59B:          C59B    293 hextodec  equ   *
C59B:E0 80             294           cpx   #$80          ;Is it a negative number?
C59D:90 0D    C5AC     295           bcc   hexdec2
C59F:49 FF             296           eor   #$FF          ;Form two's complement
C5A1:69 00             297           adc   #0            ;C = 1 from compare
C5A3:48                298           pha                  ;Save it
C5A4:8A                299           txa
C5A5:49 FF             300           eor   #$FF
C5A7:69 00             301           adc   #0
C5A9:AA                302           tax
C5AA:68                303           pla
C5AB:38                304           sec
C5AC:8D 14 02          305 hexdec2   sta   binl          ;Store the number to convert
C5AF:8E 15 02          306           stx   binh
C5B2:A9 AB             307           lda   #'+'          ;Store the sigh in the buffer
C5B4:90 02    C5B8     308           bcc   hdpos2
C5B6:A9 AD             309           lda   #'-'
C5B8:48                310 hdpos2    pha                  ;Save the sign
C5B9:A9 AC             311           lda   #','          ;Store a comma after the number
C5BB:99 01 02          312           sta   inbuf+1,y
C5BE:          C5BE    313 hdloop    equ   *             ;Divide by 10
C5BE:                  314 *
C5BE:                  315 * Divide BINH,L by 10 and leave remainder in A
C5BE:                  316 *
C5BE:A2 11             317           ldx   #16+1         ;16 bits and first time do nothing
C5C0:A9 00             318           lda   #0
C5C2:18                319           clc                  ;C=0 so first ROL leaves A=0
C5C3:2A                320 dv10loop  rol   A
C5C4:C9 0A             321           cmp   #10           ;A >= 10?
C5C6:90 02    C5CA     322           bcc   dv10lt        ;Branch if <
C5C8:E9 0A             323           sbc   #10           ;C = 1 from compare and is left set
C5CA:2E 14 02          324 dv10lt    rol   binl
C5CD:2E 15 02          325           rol   binh
C5D0:CA                326           dex
C5D1:D0 F0    C5C3     327           bne   dv10loop
C5D3:09 B0             328           ora   #'0'          ;Make a ascii char
C5D5:99 00 02          329           sta   inbuf,y
C5D8:88                330           dey
C5D9:F0 08    C5E3     331           beq   hddone        ;Stop on 0,6,12
C5DB:C0 07             332           cpy   #7
C5DD:F0 04    C5E3     333           beq   hddone
C5DF:C0 0E             334           cpy   #14
C5E1:D0 DB    C5BE     335           bne   hdloop
C5E3:68                336 hddone    pla                  ;Get the sign
C5E4:99 00 02          337           sta   inbuf,y
C5E7:60                338           rts
C5E8:DF 67 37 1C       339 qtbl      dfb   $DF,$67,$37,$1C,$07,$0C,$45,$62
C5F0:6E 7E 3B 0A       340           dfb   $6E,$7E,$3B,$0A,$0B,$48,$77,$7B
C5F8:66 2B 0C 08       341           dfb   $66,$2B,$0C,$08,$16,$53,$68,$C5
C600:          0000    342           ds    $C600-*
```

```
C600:          0356     3 DNIBL      EQU   $356
C600:          0300     4 NBUF1      EQU   $300
C600:          07DB     5 BOOTSCRN   EQU   $7DB
C600:          002B     6 SLOTZ      EQU   $2B
C600:          003C     7 BOOTTMP    EQU   $3C
C600:          004F     8 BOOTDEV    EQU   $4F
C600:A2 20               9            LDX   #$20
C602:A0 00              10            LDY   #$00
C604:64 03              11            STZ   $03
C606:64 3C              12            STZ   $3C
C608:A9 60              13            LDA   #$60
C60A:AA                 14            TAX
C60B:86 2B              15 DRV2ENT    STX   SLOTZ
C60D:85 4F              16            STA   BOOTDEV
C60F:5A                 17            PHY                  ;Y=1 IF DRIVE 2 BOOT, ELSE Y=0
C610:BD 8E C0           18            LDA   $C08E,X
C613:BD 8C C0           19            LDA   $C08C,X
C616:7A                 20            PLY
C617:B9 EA C0           21            LDA   $C0EA,Y        ;SELECT DRIVE 1 OR 2
C61A:BD 89 C0           22            LDA   $C089,X
C61D:A0 50              23            LDY   #$50
C61F:BD 80 C0           24 SEEKZERO   LDA   $C080,X
C622:98                 25            TYA
C623:29 03              26            AND   #$03
C625:0A                 27            ASL   A
C626:05 2B              28            ORA   SLOTZ
C628:AA                 29            TAX
C629:BD 81 C0           30            LDA   $C081,X
C62C:A9 56              31            LDA   #$56
C62E:20 A8 FC           32            JSR   WAIT
C631:88                 33            DEY
C632:10 EB    C61F      34            BPL   SEEKZERO
C634:85 26              35            STA   $26
C636:85 3D              36            STA   $3D
C638:85 41              37            STA   $41
C63A:20 09 C7           38            JSR   MAKTBL
C63D:64 03              39 EXTENT1    STZ   $03
C63F:18                 40 RDADR      CLC
C640:08                 41            PHP
C641:28                 42 RETRY1     PLP
C642:A6 2B              43 RDDHDR     LDX   SLOTZ          ;RESTORE X TO $60
C644:C6 03              44            DEC   $03            ;UPDATE RETRY COUNT
C646:D0 0E    C656      45            BNE   RDHD0          ;BRANCH IF NOT OUT OF RETRIES
C648:BD 88 C0           46 FUGIT      LDA   $C088,X        ;SHUT OFF DISK AND QUIT!
C64B:BD CF C6           47 FUG1       LDA   MSG-$60,X      ;(X STARTS AT $60)
C64E:10 FE    C64E      48 HANGING    BPL   HANGING        ;HANG, HANG, HANG!
C650:9D 7B 07           49            STA   BOOTSCRN-$60,X
C653:E8                 50            INX
C654:80 F5    C64B      51            BRA   FUG1
C656:08                 52 RDHD0      PHP
C657:88                 53 RETRY      DEY
C658:D0 04    C65E      54            BNE   RDHD1
C65A:F0 E5    C641      55            BEQ   RETRY1
C65C:80 DF    C63D      56 EXTENT     BRA   EXTENT1
C65E:                   57 * * * * * * * * * * * * * * * * * * *
C65E:                   58 * The following code is sacred in it's  *
C65E:                   59 * present form.  To change it would     *
C65E:                   60 * cause volcanos to errupt, the ground  *
```

```
C65E:                61 * to shake, and ProDOS not to boot!     *
C65E:                62 * * * * * * * * * * * * * * * * * * * * *
C65E:BD 8C C0        63 RDHD1      LDA   $C08C,X
C661:10 FB   C65E    64            BPL   RDHD1
C663:49 D5           65 ISMRK1     EOR   #$D5
C665:D0 F0   C657    66            BNE   RETRY
C667:BD 8C C0        67 RDHD2      LDA   $C08C,X
C66A:10 FB   C667    68            BPL   RDHD2
C66C:C9 AA           69            CMP   #$AA
C66E:D0 F3   C663    70            BNE   ISMRK1
C670:EA              71            NOP
C671:BD 8C C0        72 RDHD3      LDA   $C08C,X
C674:10 FB   C671    73            BPL   RDHD3
C676:C9 96           74            CMP   #$96
C678:F0 09   C683    75            BEQ   RDSECT
C67A:28              76            PLP
C67B:90 C2   C63F    77            BCC   RDADR
C67D:49 AD           78            EOR   #$AD
C67F:F0 25   C6A6    79            BEQ   RDATA
C681:D0 BC   C63F    80            BNE   RDADR
C683:A0 03           81 RDSECT     LDY   #$03
C685:85 40           82 RDSEC1     STA   $40
C687:BD 8C C0        83 RDSEC2     LDA   $C08C,X
C68A:10 FB   C687    84            BPL   RDSEC2
C68C:2A              85            ROL   A
C68D:85 3C           86            STA   BOOTTMP
C68F:BD 8C C0        87 RDSEC3     LDA   $C08C,X
C692:10 FB   C68F    88            BPL   RDSEC3
C694:25 3C           89            AND   BOOTTMP
C696:88              90            DEY
C697:D0 EC   C685    91            BNE   RDSEC1
C699:28              92            PLP
C69A:C5 3D           93            CMP   $3D
C69C:D0 A1   C63F    94            BNE   RDADR
C69E:A5 40           95            LDA   $40
C6A0:C5 41           96            CMP   $41
C6A2:D0 9B   C63F    97 BADRD1     BNE   RDADR
C6A4:B0 9C   C642    98            BCS   RDDHDR
C6A6:A0 56           99 RDATA      LDY   #$56
C6A8:84 3C          100 RDAT0      STY   BOOTTMP
C6AA:BC 8C C0       101 RDAT1      LDY   $C08C,X
C6AD:10 FB   C6AA   102            BPL   RDAT1
C6AF:59 D6 02       103            EOR   DNIBL-$80,Y
C6B2:A4 3C          104            LDY   BOOTTMP
C6B4:88             105            DEY
C6B5:99 00 03       106            STA   NBUF1,Y
C6B8:D0 EE   C6A8   107            BNE   RDAT0
C6BA:84 3C          108 RDAT2      STY   BOOTTMP
C6BC:BC 8C C0       109 RDAT3      LDY   $C08C,X
C6BF:10 FB   C6BC   110            BPL   RDAT3
C6C1:59 D6 02       111            EOR   DNIBL-$80,Y
C6C4:A4 3C          112            LDY   BOOTTMP
C6C6:91 26          113            STA   ($26),Y
C6C8:C8             114            INY
C6C9:D0 EF   C6BA   115            BNE   RDAT2
C6CB:BC 8C C0       116 RDAT4      LDY   $C08C,X
C6CE:10 FB   C6CB   117            BPL   RDAT4
C6D0:59 D6 02       118            EOR   DNIBL-$80,Y
```

```
C6D3:D0 CD    C6A2  119 BADREAD   BNE   BADRD1
C6D5:A0 00          120           LDY   #$00
C6D7:A2 56          121 DENIBL    LDX   #$56
C6D9:CA             122 DENIB1    DEX
C6DA:30 FB    C6D7  123           BMI   DENIBL
C6DC:B1 26          124           LDA   ($26),Y
C6DE:5E 00 03       125           LSR   NBUF1,X
C6E1:2A             126           ROL   A
C6E2:5E 00 03       127           LSR   NBUF1,X
C6E5:2A             128           ROL   A
C6E6:91 26          129           STA   ($26),Y
C6E8:C8             130           INY
C6E9:D0 EE    C6D9  131           BNE   DENIB1
C6EB:               132 * * * * * * * * * * * * * * * * * *
C6EB:               133 * Code beyond this point is not *
C6EB:               134 * sacred... It may be perverted *
C6EB:               135 * in any manner by any pervert. *
C6EB:               136 * * * * * * * * * * * * * * * * * *
C6EB:E6 27          137           INC   $27
C6ED:E6 3D          138           INC   $3D
C6EF:A5 3D          139           LDA   $3D
C6F1:CD 00 08       140           CMP   $0800
C6F4:A6 4F          141           LDX   BOOTDEV
C6F6:90 DB    C6D3  142           BCC   BADREAD
C6F8:4C 01 08       143           JMP   $0801
C6FB:4C 0B C6       144 DODRV2    JMP   DRV2ENT
C6FE:         0002  145           DS    $C700-*,0
C700:FF             146           DFB   $FF           ;MAKE IT LOOK LIKE NOTHING IN SLOT
C701:A9 E0          147 DRV2BOOT  LDA   #$E0          ;FOR DEVICE #2
C703:A0 01          148           LDY   #1            ;TO SELECT DRIVE 2
C705:A2 60          149           LDX   #$60
C707:80 F2    C6FB  150           BRA   DODRV2
C709:A2 03          151 MAKTBL    LDX   #$03
C70B:A0 00          152           LDY   #0
C70D:86 3C          153 TBLLOOP   STX   BOOTTMP
C70F:8A             154           TXA
C710:0A             155           ASL   A
C711:24 3C          156           BIT   BOOTTMP
C713:F0 10    C725  157           BEQ   NOPATRN
C715:05 3C          158           ORA   BOOTTMP
C717:49 FF          159           EOR   #$FF
C719:29 7E          160           AND   #$7E
C71B:B0 08    C725  161 TBLLOOP2  BCS   NOPATRN
C71D:4A             162           LSR   A
C71E:D0 FB    C71B  163           BNE   TBLLOOP2
C720:98             164           TYA
C721:9D 56 03       165           STA   DNIBL,X
C724:C8             166           INY
C725:E8             167 NOPATRN   INX
C726:10 E5    C70D  168           BPL   TBLLOOP
C728:A9 08          169           LDA   #$08
C72A:85 27          170           STA   $27
C72C:A0 7F          171           LDY   #$7F
C72E:60             172           RTS
C72F:         C72F  173 MSG       EQU   *
C72F:               174           MSB   ON
C72F:C3 E8 E5 E3    175           ASC   'Check        Disk Drive.'
C740:               176 *
```

```
C740:                177 * The following code is Teri's memory and
C740:                178 * soft switch exercise program.  The only
C740:                179 * purpose is exercise, not diagnostic
C740:                180 * functions.  This code is activated on
C740:                181 * a system without a keyboard, or when
C740:                182 * both open and closed apple keys are
C740:                183 * pressed during the reset sequence.
C740:                184 *
C740:08 50 52        185 TBL1      DFB   $08,$50,$52   ;These are low order
C743:00 02 04        186           DFB   $00,$02,$04   ; addresses of $C0XX
C746:8B 8B E8        187           DFB   $8B,$8B,$E8   ; that must be re-selected
C749:09 50 52        188 TBL2      DFB   $09,$50,$52   ; after each page write
C74C:00 03 05        189           DFB   $00,$03,$05   ; (especially $C000!)
C74F:83 83 E8        190           DFB   $83,$83,$E8
C752:                191 *
C752:64 00           192 XLOOP1    STZ   $00           ;Reset low address to 2
C754:E6 00           193           INC   $00           ;Hi addr assumed to = 0
C756:E6 00           194           INC   $00
C758:92 00           195 XPAGE     STA   ($00)         ;Write entire page with
C75A:9D 00 C0        196           STA   $C000,X       ; shifted data... BUT
C75D:6A              197           ROR   A             ; restore Z-page after
C75E:E6 00           198           INC   $00           ; write in case $C008-9
C760:D0 F6   C758    199           BNE   XPAGE         ; is current pointer
C762:18              200           CLC                  ;Indicates regular pass
C763:98              201 XMODE     TYA                  ;Get settings, each bit
C764:A0 08           202           LDY   #$08          ;Specifies main/alt set
C766:BE 40 C7        203 XRSET     LDX   TBL1,Y        ;Assume Main $C000 setting
C769:90 03   C76E    204           BCC   XRST1         ;Branch if Main setting
C76B:BE 49 C7        205           LDX   TBL2,Y        ;Else get Alternate index
C76E:9D 00 C0        206 XRST1     STA   $C000,X
C771:2A              207           ROL   A             ;Accumulator makes full
C772:88              208           DEY                 ; circle
C773:10 F1   C766    209           BPL   XRSET
C775:A8              210           TAY                 ;Preserve settings in Y
C776:B0 DA   C752    211           BCS   XLOOP1        ;Branch if new setting
C778:E6 01           212           INC   $01
C77A:D0 DC   C758    213           BNE   XPAGE         ;Loop til all pages writen
C77C:38              214 BANGER    SEC                 ;Indicate new settings,
C77D:C8              215           INY                 ; reset mem pointer after
C77E:80 E3   C763    216           BRA   XMODE         ; after new settings
C780:        0000    217           DS    $C780-*
C780:                 24           INCLUDE MBASIC      ;Mouse BASIC routines @$C780
```

```
C780:                 3 ****************************************
C780:                 4 *
C780:                 5 * XMBASIC - Basic call to the mouse
C780:                 6 *
C780:                 7 ****************************************
C780:          C780   8 xmbasic   equ   *
C780:5A               9           phy
C781:B0 1C    C79F   10           bcs   basicin       ;Input?
C783:A0 C4           11           ldy   #<mbasic      ;Input from $C400?
C785:C4 39           12           cpy   kswh
C787:D0 04    C78D   13           bne   xmbout
C789:A4 38           14           ldy   kswl
C78B:F0 12    C79F   15           beq   basicin
C78D:DA              16 xmbout    phx                  ;Save X too
C78E:48              17           pha
C78F:29 7F           18           and   #$7F          ;We don't care about high bit
C791:C9 02           19           cmp   #2
C793:B0 06    C79B   20           bge   mbbad         ;Only 0,1 valid
C795:20 3D C4        21           jsr   xsetmou
C798:20 6D C4        22           jsr   xmhome
C79B:68              23 mbbad     pla
C79C:FA              24           plx
C79D:7A              25           ply
C79E:60              26           rts


C79F:                28 ****************************************
C79F:                29 *
C79F:                30 * BASICIN - Input from basic
C79F:                31 *
C79F:                32 * Creates  +XXXXX,+YYYYY,+SS
C79F:                33 *  XXXXX = X position
C79F:                34 *  YYYYY = Y position
C79F:                35 *  SS = Status
C79F:                36 *       - = Key pressed
C79F:                37 *       1 = Button pressed
C79F:                38 *       2 = Button just pressed
C79F:                39 *       3 = Button just released
C79F:                40 *       4 = Button not pressed
C79F:                41 *
C79F:                42 ****************************************
C79F:          C79F  43 basicin   equ   *
C79F:91 28           44           sta   (basl),y      ;Fix flashing char
C7A1:A9 05           45           lda   #>inent       ;Fix input entry
C7A3:85 38           46           sta   kswl
C7A5:AD 00 C0        47           lda   kbd           ;test the keyboard
C7A8:0A              48           asl   A
C7A9:08              49           php                  ;Save kbd and int stat for later
C7AA:78              50           sei                  ;No interrupts while getting position
C7AB:20 95 C4        51           jsr   xmread
C7AE:A0 05           52           ldy   #5            ;Move X position into the buffer
C7B0:AE 7C 05        53           ldx   mouxh
C7B3:AD 7C 04        54           lda   mouxl
C7B6:20 9B C5        55           jsr   hextodec      ;Convert it
C7B9:A0 0C           56           ldy   #12
C7BB:AE FC 05        57           ldx   mouyh
C7BE:AD FC 04        58           lda   mouyl
```

```
C7C1:20 9B C5       59            jsr   hextodec
C7C4:AD 7C 07       60            lda   moustat
C7C7:2A             61            rol   A
C7C8:2A             62            rol   A
C7C9:2A             63            rol   A
C7CA:29 03          64            and   #3
C7CC:49 03          65            eor   #3
C7CE:1A             66            inc   A
C7CF:28             67            plp                    ;Restore int & kbd status
C7D0:A0 10          68            ldy   #16
C7D2:20 AC C5       69            jsr   hexdec2          ;X=0 from last div10
C7D5:7A             70            ply
C7D6:A2 11          71            ldx   #17              ;X = EOL
C7D8:A9 8D          72            lda   #$8D             ;Carriage return
C7DA:9D 00 02       73 putinbuf   sta   inbuf,x
C7DD:60             74            rts


C7DE:               76 ************************************
C7DE:               77 *
C7DE:               78 * PADDLE patch
C7DE:               79 *
C7DE:               80 ************************************
C7DE:        C7DE   81 mpaddle    equ   *
C7DE:AD FC 07       82            lda   moumode          ;Is the mouse active?
C7E1:C9 01          83            cmp   #01              ;Only transparent mode
C7E3:F0 06   C7EB   84            beq   pdon
C7E5:AD 70 C0       85            lda   vblclr           ;Fire the strobe
C7E8:4C 21 FB       86            jmp   $FB21
C7EB:        C7EB   87 pdon       equ   *
C7EB:E0 01          88            cpx   #1               ;C=1 if X=1
C7ED:6A             89            ror   A                ;A=80 or 0
C7EE:A8             90            tay
C7EF:B9 7C 05       91            lda   mouxh,y          ;Get high byte
C7F2:F0 02   C7F6   92            beq   pdok
C7F4:A9 FF          93            lda   #$FF
C7F6:19 7C 04       94 pdok       ora   mouxl,y
C7F9:A8             95            tay
C7FA:60             96            rts
C7FB:5D E8 C5       97 zznml      eor   qtbl,x
C7FE:80 DA   C7DA   98            bra   putinbuf
C800:        0000   99            ds    $C800-*,0
C800:               25            INCLUDE IRQBUF         ;Interrupt stuff @$C800
```

```
C800:                3 *
C800:                4 * this is the main (only) IRQ handling routines
C800:                5 *
C800:4C E4 C1        6            jmp   plinit         ;Pascal 1.0 Initialization
C803:48              7 NEWIRQ     PHA                  ;SAVE ACC ON STACK, NOT $45
C804:68              8            PLA                  ;LEGAL BECAUSE IF IRQ, IRQ DISABLED.
C805:68              9            PLA                  ;GET STATUS REGISTER
C806:48             10 IRQ1       PHA
C807:D8             11            CLD                  ;CLEAR DEC MODE, ELSE THINGS GET SCREWED.
C808:29 10          12            AND   #$10           ;SET CARRY TO INDICATE BRK
C80A:69 F0          13            ADC   #$F0
C80C:8A             14            TXA                  ;SAVE X IN A WHILE
C80D:BA             15            TSX                  ; FUTZING WITH THE STACK
C80E:CA             16            DEX                  ; RECOVER A-REG AT TOP...
C80F:9A             17            TXS
C810:48             18            PHA                  ;SAVE X ON STACK (ON TOP OF A)
C811:5A             19            PHY                  ; AND Y ALSO
C812:AE 66 C0       20            LDX   MOUX1          ;Get mouse info
C815:AC 67 C0       21            LDY   MOUY1
C818:AD 18 C0       22            LDA   RD80COL        ;TEST FOR 80-STORE WITH
C81B:2D 1C C0       23            AND   RDPAGE2        ; PAGE 2 TEXT.
C81E:29 80          24            AND   #$80           ; MAKE IT ZERO OR $80
C820:F0 05   C827   25            BEQ   IRQ2
C822:8D 54 C0       26            STA   TXTPAGE1
C825:A9 40          27            LDA   #$40           ;SET PAGE 2 RESET BIT.
C827:2C 13 C0       28 IRQ2       BIT   RDRAMRD
C82A:10 05   C831   29            BPL   IRQ3           ;BRANCH IF MAIN RAM READ
C82C:8D 02 C0       30            STA   RDMAINRAM      ;ELSE, SWITCH IT IN
C82F:09 20          31            ORA   #$20           ; AND RECORD THE EVENT!
C831:2C 14 C0       32 IRQ3       BIT   RDRAMWRT       ;DO THE SAME FOR RAM WRITE.
C834:10 05   C83B   33            BPL   IRQ4
C836:8D 04 C0       34            STA   WRMAINRAM
C839:09 10          35            ORA   #$10
C83B:B0 13   C850   36 IRQ4       BCS   IRQ5           ;BRANCH IF BREAK, NOT INTERRUPT
C83D:48             37            PHA                  ;SAVE MACHINE STATES SO FAR...
C83E:20 D5 C4       38            JSR   MOUSEINT       ;GO TEST THE MOUSE
C841:90 3F   C882   39            BCC   IRQDONE        ;BRANCH IF IT WAS THE MOUSE
C843:20 00 C9       40            JSR   ACIAINT        ;GO TEST ACIA AND KEYBOARD INTERRUPTS
C846:90 3A   C882   41            BCC   IRQDONE        ;BRANCH IF INTERRUPT SERVICED
C848:68             42            PLA                  ;RESTORE STATES RECORDED SO FAR
C849:18             43            CLC                  ;RESET BREAK/INTERRUPT INDICATOR
C84A:80 04   C850   44            bra   passkipl       ;Skip around pascal 1.0 stuff
C84C:        0001   45            ds    $C84D-*,$00
C84D:4C EE C1       46            jmp   plread
C850:        C850   47 passkipl   equ   *
C850:2C 12 C0       48 IRQ5       BIT   RDLCRAM        ;DETERMINE IF LANGUAGE CARD ACTIVE
C853:10 0C   C861   49            BPL   IRQ7
C855:09 0C          50            ORA   #$C            ;SET TWO BITS SO RESTORED
C857:2C 11 C0       51            BIT   RDLCBNK2       ; LANGUAGE CARD IS WRITE ENABLED
C85A:10 02   C85E   52            BPL   IRQ6           ;BRANCH IF NOT PAGE 2 OF $D000
C85C:49 06          53            EOR   #$6            ;ENABLE READ FOR PAGE 2 ON EXIT
C85E:8D 81 C0       54 IRQ6       STA   ROMIN
C861:2C 16 C0       55 IRQ7       BIT   RDALTZP        ;LAST...AND VERY IMPORTANT!
C864:10 0D   C873   56            BPL   IRQ8           ; UNLESS IT IS NOT ENABLED
C866:BA             57            TSX                  ;SAVE CURRENT STACK POINTER
C867:8E 01 01       58            STX   $101           ;AT BOTTOM OF STACK
C86A:AE 00 01       59            LDX   $100           ;GET MAIN STACK POINTER
C86D:9A             60            TXS
```

```
C86E:8D 08 C0       61            STA   SETSTDZP
C871:09 80          62            ORA   #$80
C873:B0 2A   C89F   63 IRQ8       BCS   GOBREAK
C875:48             64            PHA
C876:A9 C8          65            LDA   #<IRQDONE
C878:48             66            PHA
C879:A9 82          67            LDA   #>IRQDONE     ;SAVE RETURN IRQ ADDR
C87B:48             68            PHA
C87C:A9 04          69            LDA   #4            ; SO WHEN INTERRUPT DOES RTI
C87E:48             70            PHA                 ; IT RETURNS TO IRQDONE.
C87F:6C FE 03       71            JMP   ($3FE)        ;PROCESS EXTERNAL INTERRUPT


C882:68             73 IRQDONE    PLA                 ;RECOVER MACHINE STATE
C883:10 07   C88C   74            BPL   IRQDNE1       ;BRANCH IF MAIN ZP WAS ACTIVE
C885:8D 09 C0       75            STA   SETALTZP
C888:AE 01 01       76            LDX   $101          ;RESTORE ALTERNATE STACK POINTER
C88B:9A             77            TXS
C88C:0A             78 IRQDNE1    ASL   A
C88D:A0 05          79            LDY   #$05
C88F:BE 89 C9       80 IRQDNE2    LDX   IRQTBLE,Y
C892:88             81            DEY
C893:0A             82            ASL   A
C894:90 03   C899   83            BCC   IRQDNE3       ;BRANCH IF SWITCH IS OK.
C896:9D 00 C0       84            STA   $C000,X
C899:D0 F4   C88F   85 IRQDNE3    BNE   IRQDNE2       ;BRANCH IF MORE SWITCHES
C89B:7A             86            PLY
C89C:FA             87            PLX                 ;RESTORE ALL REGISTERS
C89D:68             88            PLA
C89E:40             89            RTI                 ;DO THE REAL RTI!
C89F:4C 47 FA       90 GOBREAK    JMP   NEWBRK        ;PASS THE BREAKER THROUGH


C8A2:               92 ****************************************
C8A2:               93 *
C8A2:               94 * MOVEIRQ - This routine transfers the roms interrupt vector into
C8A2:               95 *  both language cards
C8A2:               96 *
C8A2:               97 ****************************************
C8A2:       C8A2    98 moveirq    equ   *
C8A2:20 60 C3       99            JSR   SETROM        ;Read ROM and Write to RAM
C8A5:AD 16 C0      100            LDA   RDALTZP       ;Which language card?
C8A8:0A            101            ASL   A             ;C=1 if alternate card
C8A9:A0 01         102            LDY   #1            ;Move two bytes
C8AB:B9 FE FF      103 MIRQLP     LDA   IRQVECT,Y     ;Get byte from ROM
C8AE:8D 09 C0      104            STA   SETALTZP      ;Set alternate card
C8B1:99 FE FF      105            STA   IRQVECT,Y     ;Store it in the RAM card
C8B4:8D 08 C0      106            STA   SETSTDZP      ;Set main card
C8B7:99 FE FF      107            STA   IRQVECT,Y
C8BA:88            108            DEY
C8BB:10 EE   C8AB  109            BPL   MIRQLP        ;Go do the second byte
C8BD:90 03   C8C2  110            BCC   MIRQSTD       ;Is the card set right?
C8BF:8D 09 C0      111            STA   SETALTZP      ;No, it wasn't
C8C2:4C 54 C3      112 MIRQSTD    JMP   RESETLC       ;Clean up & go home
```

```
C8C5:                 114 *   This is the serial input routine.  Carry
C8C5:                 115 * flag set indicates that returned data is
C8C5:                 116 * valid.
C8C5:                 117 *
C8C5:                 118 * GETBUF- Gets a byte from the buffer & updates pointers
C8C5:                 119 *  On entry Y=0 for Serial buffer Y=$80 for Keyboard buffer
C8C5:EC FF 04         120 XRDSER    CPX   ACIABUF       ;is serial input buffered?
C8C8:D0 26    C8F0    121           BNE   XNOSBUF       ;(in english "NO SERIAL BUFFER")
C8CA:A0 00            122           LDY   #0            ;Y=0 for serial buffer
C8CC:         C8CC    123 GETBUF    EQU   *
C8CC:B9 7F 06         124           LDA   TRSER,Y       ;Test for data in buffer
C8CF:D9 7F 05         125           CMP   TWSER,Y       ;If = then no data
C8D2:F0 24    C8F8    126           BEQ   GBEMPTY
C8D4:48               127           PHA                 ;Save current value
C8D5:1A               128           INC   A             ;Update the pointer
C8D6:89 7F            129           BIT   #$7F          ;Overflow
C8D8:D0 01    C8DB    130           BNE   GBNOOVR
C8DA:98               131           TYA
C8DB:99 7F 06         132 GBNOOVR   STA   TRSER,Y       ;Store the updated pointer
C8DE:7A               133           PLY                 ;Get the old value of the pointer
C8DF:AD 13 C0         134           LDA   RDRAMRD       ;Are we in main ram
C8E2:0A               135           ASL   A             ;C=1 for Aux ram
C8E3:8D 03 C0         136           STA   RDCARDRAM     ;Force Aux ram
C8E6:B9 00 08         137           LDA   THBUF,Y       ;Get byte from buffer
C8E9:B0 14    C8FF    138           BCS   XRDSNO        ;Branch if we were in aux bank
C8EB:8D 02 C0         139           STA   RDMAINRAM     ;Set back to main
C8EE:38               140           SEC
C8EF:60               141           RTS                 ;Note C=1
C8F0:                 142 *
C8F0:BC 85 C8         143 XNOSBUF   LDY   DEVNO,X       ;Get index to ACIA
C8F3:B9 F9 BF         144           LDA   SSTAT,Y       ;Test ACIA directly for data
C8F6:29 08            145           AND   #$8
C8F8:18               146 GBEMPTY   CLC                 ;indicate no data
C8F9:F0 04    C8FF    147           BEQ   XRDSNO        ;Branch if no data!
C8FB:B9 F8 BF         148           LDA   SDATA,Y       ;get serial input
C8FE:38               149 notacia   SEC                 ;indicate valid data returned.
C8FF:         C8FF    150 acdone    equ   *
C8FF:60               151 XRDSNO    RTS
```

```
C900:                 153 *   This routine will determine if the source of
C900:                 154 * is either of the built in ACIAs.  If neither port
C900:                 155 * generated the interrupt, or the interrupt was due
C900:                 156 * to a transmit buffer empty, protocol converter, or
C900:                 157 * 'unbuffered' receiver full, the carry is set indi-
C900:                 158 * cating an externally serviced interrupt.
C900:                 159 *   If the interrupt source was keyboard, 'buffered'
C900:                 160 * serial input, or the DCD, the interrupt is serviced
C900:                 161 * and the carry is cleared indicating interrupt was
C900:                 162 * serviced. (DCD handshake replaces CTS.)
C900:                 163 *   Location "ACIABUF" specifies which (if either) re-
C900:                 164 * ceiver data is buffered.  For port 1 it must contain
C900:                 165 * $C1, for port 2 a $C2.  Any other values are cause
C900:                 166 * interrupts to pass to external (RAM based) routines.
C900:                 167 *   Location "TYPHED" specifies whether Keyboard in-
C900:                 168 * put should be buffered, ignored, or processed by
C900:                 169 * RAM based routines.  If bit 7=1 and bit 6=0, key-
C900:                 170 * board data is placed in the type-ahead buffer. If
C900:                 171 * bit 6 is set the interrupt is cleared, but must
C900:                 172 * be recognized and serviced by a RAM routine.  If
C900:                 173 * both bits = 0, the interrupt is serviced, but the
C900:                 174 * keyboard data is ignored.
C900:                 175 *   While using type-ahead, Open-Apple CTRL-X will
C900:                 176 * flush the buffer.  No other code is recognized.
C900:                 177 *   If the source was an ACIA that has the transmit
C900:                 178 * interrupt enabled, the original value of the ACIAs
C900:                 179 * status registers is preserved. Automatic serial input
C900:                 180 * buffering is not serviced from a port so configured.
C900:                 181 * Interrupts originating from the protocol converter or
C900:                 182 * keyboard (RAM serviced) do not inhibit serial buffering
C900:                 183 * and are passed thru.  The RAM service routine can rec-
C900:                 184 * ognize the interrupt source by a 1 state in bit 6 of
C900:                 185 * the ACIAs status register.  The RAM service routine must
C900:                 186 * cause the clearing of DSR (bit 6) AND make a second ac-
C900:                 187 * cess to the status register before returning.
C900:                 188 *
C900:                 189 *
C900:       C900      190 aciaint   equ   *
C900:A2 C2            191           ldx   #<comslot     ;Test port 2 first
C902:20 08 C9         192           jsr   aciatst       ;Check for interrupt
C905:90 F8   C8FF     193           bcc   acdone        ;Return if interrupt done
C907:CA               194           dex                  ;Try port 1
C908:BC 85 C8         195 aciatst   ldy   devno,x       ;Get index for acia
C90B:A9 04            196           lda   #$4           ;If xmit ints enabled pass to user
C90D:59 FA BF         197           eor   scomd,y       ;Check if D<3>, D<2> = 01
C910:29 0C            198           and   #$0C          ;
C912:F0 EA   C8FE     199           beq   notacia       ;User better take it!
C914:B9 F9 BF         200           lda   sstat,y       ;Get status
C917:9D 38 04         201           sta   astat,x       ;Save it away
C91A:10 E2   C8FE     202           bpl   notacia       ;No interrupt
C91C:E0 C2            203 aitst2    cpx   #<comslot     ;C=1 if com port
C91E:B0 02   C922     204           bcs   aiport2       ;Invert DSR if port1
C920:49 40            205           eor   #$40
C922:3C 38 05         206 aiport2   bit   extint,x      ;Is DSR enabled?
C925:70 26   C94D     207           bvs   aipass        ;Yes, user wants it
C927:10 22   C94B     208           bpl   aieatit       ;No, eat it
C929:90 20   C94B     209           bcc   aieatit       ;Yes but I don't want it for port 1
C92B:89 40            210           bit   #$40          ;Is DSR 1?
```

```
C92D:F0 1E   C94D  211           beq   aipass        ;If not, skip it
C92F:              212 * It's a keyboard interrupt
C92F:AD 00 C0      213           lda   kbd           ;Get the key
C932:A0 80         214           ldy   #$80
C934:20 67 C9      215           jsr   putbuf        ;Put it in the buffer
C937:C9 98         216           cmp   #$98          ;Is it a ^x?
C939:D0 08   C943  217           bne   ainoflsh
C93B:AD 62 C0      218           lda   butnl         ;And the closed apple?
C93E:10 03   C943  219           bpl   ainoflsh
C940:20 1B CB      220           jsr   flush         ;Flush the buffer
C943:AD 10 C0      221 ainoflsh  lda   kbdstrb       ;Clear the keyboard
C946:              222 * $A0 $B0 table needed by serial firmware
C946:       00C1   223 sltdmy    equ   <serslot
C946:       C885   224 devno     equ   *-sltdmy
C946:A0 B0         225           ldy   #$B0          ;Restore y
C948:B9 F9 BF      226           lda   sstat,y       ;Read status to clear int
C94B:29 BF         227 aieatit   and   #$BF          ;Clear the DSR bit
C94D:0A            228 aipass    asl   A             ;Shift DSR into C
C94E:0A            229           asl   A
C94F:29 20         230           and   #$20          ;Is the receiver full?
C951:F0 35   C988  231           beq   aciadone      ;If not, we're done
C953:B9 FA BF      232           lda   scomd,y       ;Are receive interrupts enabled?
C956:49 01         233           eor   #1            ;Check for D<1>,D<0> = 01
C958:29 03         234           and   #3
C95A:D0 2C   C988  235           bne   aciadone      ;If not, were done
C95C:8A            236           txa                  ;Is this acia buffered?
C95D:4D FF 04      237           eor   aciabuf
C960:D0 9C   C8FE  238           bne   notacia       ;The user better handle it!
C962:B9 F8 BF      239           lda   sdata,y       ;It's mine
C965:A0 00         240           ldy   #0
C967:       C967   241 putbuf    equ   *
C967:DA            242           phx
C968:48            243           pha
C969:B9 7F 05      244           lda   twser,y       ;Get buffer pointer
C96C:AA            245           tax                  ;Save it for later
C96D:1A            246           inc   A             ;Bump it to next free byte
C96E:89 7F         247           bit   #$7F          ;Overflow?
C970:D0 01   C973  248           bne   pbok
C972:98            249           tya                  ;Wrap pointer
C973:08            250 pbok      php                  ;Save DSR status
C974:D9 7F 06      251           cmp   trser,y       ;Buffer full?
C977:F0 03   C97C  252           beq   pbfull
C979:99 7F 05      253           sta   twser,y       ;Save the new pointer
C97C:28            254 pbfull    plp
C97D:68            255           pla                  ;Get the data
C97E:8D 05 C0      256           sta   wrcardram     ;It goes to aux ram
C981:9D 00 08      257           sta   thbuf,x
C984:8D 04 C0      258           sta   wrmainram
C987:FA            259           plx
C988:60            260 aciadone  rts


C989:83 8B 8B      262 IRQTBLE   DFB   >LCBANK2,>LCBANK1,>LCBANK1
C98C:05 03 55      263           DFB   >WRCARDRAM,>RDCARDRAM,TXTPAGE2
```

```
C98F:                 266 *   The following two routines are for reading key-
C98F:                 267 * board and serial input from buffers or directly.
C98F:                 268 *   Type-ahead buffering only occurs for non auto-
C98F:                 269 * repeat keypresses.  When a key is pressed for
C98F:                 270 * auto-repeat the buffer is first emptied, then the
C98F:                 271 * repeated characters are returned.
C98F:                 272 *   The minus flag is used to indicate if a keystroke
C98F:                 273 * is being returned.
C98F:                 274 *

C98F:20 AD C9         276 XRDKBD    JSR   XBITKBD       ;is keyboard input ready?
C992:10 14    C9A8    277           BPL   XNOKEY        ;Branch if not.
C994:90 0A    C9A0    278           BCC   XRKBD1        ;Branch if direct KBD input.
C996:5A               279           PHY                 ;Save Y
C997:A0 80            280           LDY   #$80          ;Y=$80 for keyboard buffer
C999:20 CC C8         281           JSR   GETBUF        ;Get data from buffer
C99C:7A               282           PLY
C99D:09 00            283           ORA   #0            ;Set minus flag
C99F:60               284           RTS

C9A0:AD 00 C0         286 XRKBD1    LDA   KBD           ;test keyboard directly
C9A3:10 EA    C98F    287           BPL   XRDKBD        ;loop if buffered since test.
C9A5:8D 10 C0         288           STA   KBDSTRB       ;Clear keyboard strobe.
C9A8:60               289 XNOKEY    RTS                 ;Minus flag indicates valid character

C9A9:         0001    291           ds    $C9AA-*,$00
C9AA:4C F6 C1         292           jmp   plwrite       ;Pascal 1.0 entry point

C9AD:2C FA 05         294 XBITKBD   BIT   TYPHED         ;This routine replaces "BIT KBD" instrucs
C9B0:10 10    C9C2    295           BPL   XBKB2         ; so as to function with type-ahead.
C9B2:38               296           SEC                 ;anticipate data in buffer is ready
C9B3:08               297           PHP                 ;save carry and minus flags
C9B4:48               298           PHA                 ;preserve accumulator
C9B5:AD FF 06         299           LDA   TRKEY
C9B8:CD FF 05         300           CMP   TWKEY         ;is there data to be read?
C9BB:F0 03    C9C0    301           BEQ   XBKB1         ;branch if type-ahead buffer empty
C9BD:68               302           PLA
C9BE:28               303           PLP
C9BF:60               304           RTS                 ;Carry and minus flag already set.
C9C0:                 305 *
C9C0:68               306 XBKB1     PLA
C9C1:28               307           PLP                 ;restore ACC and Status
C9C2:2C 00 C0         308 XBKB2     BIT   KBD           ;test KBD Directly
C9C5:18               309           CLC                 ;indicate direct test
C9C6:60               310           RTS
C9C7:                  26           INCLUDE COMMAND     ;Serial firmware command processor
```

```
C9C7:                    3           MSB   OFF
C9C7:          C9C7      4 cmdtable  equ   *
C9C7:66                  5           dfb   >cmdi-1
C9C8:66                  6           dfb   >cmdk-1
C9C9:66                  7           dfb   >cmdl-1
C9CA:5C                  8           dfb   >cmdn-1
C9CB:5C                  9           dfb   >cmdcr-1
C9CC:7C                 10           dfb   >cmdb-1
C9CD:78                 11           dfb   >cmdd-1
C9CE:77                 12           dfb   >cmdp-1
C9CF:C3                 13           dfb   >cmdq-1
C9D0:B4                 14           dfb   >cmdr-1
C9D1:98                 15           dfb   >cmds-1
C9D2:C5                 16           dfb   >cmdt-1
C9D3:54                 17           dfb   >cmdz-1
C9D4:7F BF BF 7F        18 mask1     dfb   $7F,$BF,$BF,$7F,$FF
C9D9:80 00 40 00        19 mask2     dfb   $80,$00,$40,$00,$00
C9DE:          C9DE     20 cmdlist   equ   *
C9DE:49 4B 4C 4E        21           asc   "IKLN"
C9E2:0D                 22           dfb   $0D           ;Carriage return
C9E3:42 44 50 51        23           asc   "BDPQRSTZ"
C9EB:48                 24 command   pha                  ;Check for command to firmware
C9EC:3C B8 03           25           bit   sermode,x     ;Already in command?
C9EF:30 1B    CA0C      26           bmi   incmd         ;If so,go do it
C9F1:BC 38 06           27           ldy   eschar,x      ;If eschar = 0 ignore commands
C9F4:F0 13    CA09      28           beq   nocmd
C9F6:5D 38 06           29           eor   eschar,x      ;Is it the command char?
C9F9:0A                 30           asl   A             ;Ignore high bit
C9FA:D0 0D    CA09      31           bne   nocmd
C9FC:AC FB 07           32           ldy   cursor        ;Save the cursor
C9FF:8C 79 06           33           sty   oldcur
CA02:A0 BF              34           ldy   #cmdcur       ;Set command cursor
CA04:8C FB 07           35           sty   cursor
CA07:80 2D    CA36      36           bra   cominit
CA09:38                 37 nocmd     sec                  ;Mark char not handled
CA0A:68                 38           pla                  ;Restore char
CA0B:60                 39           rts
CA0C:          CA0C     40 incmd     equ   *             ;Command mode
CA0C:BC 85 C8           41           ldy   devno,x       ;Get index for ACIA
CA0F:29 5F              42           and   #$5F          ;Ignore hi bit: just upshift lowercase
CA11:DA                 43           phx                  ;Save slot
CA12:A2 0C              44           ldx   #12           ;Check 13 commands
CA14:DD DE C9           45 cmdloop   cmp   cmdlist,x
CA17:F0 34    CA4D      46           beq   cmfound       ;Right char?
CA19:CA                 47           dex
CA1A:10 F8    CA14      48           bpl   cmdloop
CA1C:FA                 49           plx                  ;We didn't find it
CA1D:68                 50           pla
CA1E:48                 51           pha
CA1F:29 7F              52           and   #$7F          ;if char is cntl char
CA21:C9 20              53           cmp   #$20          ;it can be the new comd char
CA23:B0 03    CA28      54           bcs   ckdig         ;branch if not cntl character
CA25:9D 38 06           55 cmdz2     sta   eschar,x      ;Save comd char, drop thru ckdig to cdone
CA28:49 30              56 ckdig     eor   #$30          ;Is it a number?
CA2A:C9 0A              57           cmp   #$0A
CA2C:B0 0E    CA3C      58           bcs   cdone         ;If so, branch
CA2E:A0 0A              59           ldy   #10           ;A = A + 10 * current number
CA30:6D 7F 04           60 digloop   adc   number        ;C=0 on first entry
```

```
CA33:88              61           dey
CA34:D0 FA   CA30   62           bne   digloop
CA36:8D 7F 04       63 cominit   sta   number
CA39:38             64           sec                  ;Mark in command mode
CA3A:80 07   CA43   65           bra   cmset
CA3C:18             66 cdone     clc                  ;Out of command mode
CA3D:AD 79 06       67           lda   oldcur         ;Restore the cursor
CA40:8D FB 07       68           sta   cursor
CA43:08             69 cmset     php
CA44:1E B8 03       70           asl   sermode,x      ;set command mode according to carry
CA47:28             71           plp
CA48:7E B8 03       72           ror   sermode,x      ;leaves carry clear
CA4B:68             73           pla                   ;character handled
CA4C:60             74           rts                  ;because carry clear...

CA4D:A9 CA          76 cmfound   lda   #<cmdcr
CA4F:48             77           pha                  ;do JMP via RTS
CA50:BD C7 C9       78           lda   cmdtable,x
CA53:48             79           pha
CA54:60             80           rts                  ;Go to it



CA55:FA             82 cmdz      plx                  ;Zero escape character
CA56:9E B8 04       83           stz   pwdth,x        ;And the width
CA59:A9 00          84           lda   #0
CA5B:80 C8   CA25   85           bra   cmdz2



CA5D:        CA5D   87 cmdcr     equ   *
CA5D:        CA5D   88 cmdn      equ   *
CA5D:7A             89           ply
CA5E:AD 7F 04       90           lda   number         ;Get number inputted
CA61:F0 05   CA68   91           beq   cmdi2          ;Don't change printer width if 0
CA63:99 B8 04       92           sta   pwdth,y        ;Update printer width
CA66:F0             93           dfb   $F0            ;BEQ opcode to skip next byte
CA67:        CA67   94 cmdi      equ   *
CA67:        CA67   95 cmdk      equ   *
CA67:        CA67   96 cmdl      equ   *
CA67:7A             97           ply
CA68:B9 B8 06       98 cmdi2     lda   flags,y
CA6B:3D D4 C9       99           and   mask1,x        ;Mask off bit we'll change
CA6E:1D D9 C9      100           ora   mask2,x        ;Change it
CA71:99 B8 06      101           sta   flags,y        ;Back it goes
CA74:98            102           tya                   ;Put slot back in x
CA75:AA            103           tax
CA76:80 C4   CA3C  104 cdone2    bra   cdone          ;Good bye



CA78:88            106 cmdp      dey                   ;Make y point to command reg
CA79:A9 1F         107 cmdd      lda   #$1F           ;Mask off high three bits
CA7B:38            108           sec                   ;C=1 means high 3 bits
CA7C:90            109           dfb   $90            ;BCC opcode to skip next byte
CA7D:A9 F0         110 cmdb      lda   #$F0           ;Mask off lower 4 bits F0 = BNE
CA7F:18            111           clc                   ;F0 will skip this if cmdp or cmdd
CA80:39 FB BF      112           and   scntl,y        ;Mask off bits being changed
```

```
CA83:8D F8 06      113           sta   temp          ;Save it
CA86:FA            114           plx
CA87:AD 7F 04      115           lda   number        ;Get inputed number
CA8A:29 0F         116           and   #$0F          ;Only lower nibble valid
CA8C:90 05   CA93  117           bcc   noshift       ;If C=1 shift to upper 3 bits
CA8E:0A            118           asl   A
CA8F:0A            119           asl   A
CA90:0A            120           asl   A
CA91:0A            121           asl   A
CA92:0A            122           asl   A
CA93:0D F8 06      123 noshift   ora   temp          ;Get the rest of the bits
CA96:C8            124           iny                 ;Put them in the ACIA
CA97:80 17   CAB0  125           bra   cmdp2         ;increment puts em away where they go.


CA99:B9 FA BF      127 cmds      lda   scomd,y       ;Transmit a break
CA9C:48            128           pha                  ;Save current ACIA state
CA9D:09 0C         129           ora   #$0C          ;Do the braek
CA9F:99 FA BF      130           sta   scomd,y
CAA2:A9 E9         131           lda   #233          ;For 233 ms
CAA4:A2 53         132 mswait    ldx   #83           ;Wait 1 ms
CAA6:48            133 msloop    pha                 ;((12*82)+11)+2+3=1000us
CAA7:68            134           pla
CAA8:CA            135           dex
CAA9:D0 FB   CAA6  136           bne   msloop
CAAB:3A            137           dec   a
CAAC:D0 F6   CAA4  138           bne   mswait
CAAE:68            139           pla
CAAF:FA            140           plx
CAB0:99 FA BF      141 cmdp2     sta   scomd,y
CAB3:80 C1   CA76  142           bra   cdone2


CAB5:99 F9 BF      144 cmdr      sta   sstat,y       ;Reset the ACIA
CAB8:AD 7B 06      145           lda   vfactv        ;Check if video firmware active
CABB:0A            146           asl   A             ;Save it in C
CABC:20 23 CE      147           jsr   sethooks      ;assume video firmware active
CABF:90 03   CAC4  148           bcc   cmdq          ;branch if good guesser...
CAC1:20 4D CE      149           jsr   zzquit        ;Reset the hooks
CAC4:18            150 cmdq      clc                  ;Quit terminal mode
CAC5:B0            151           dfb   $B0           ;BCS to skip next byte
CAC6:38            152 cmdt      sec                  ;Into terminal mode
CAC7:FA            153           plx                  ;Recover X
CAC8:20 CD CA      154           jsr   setterm
CACB:80 A9   CA76  155           bra   cdone2
CACD:BD B8 03      156 setterm   lda   sermode,x     ;Get terminal mode status
CAD0:89 40         157           bit   #$40          ;Z=1 if not in terminal mode
CAD2:90 12   CAE6  158           bcc   stclr         ;Branch if clearing terminal mode
CAD4:D0 20   CAF6  159           bne   stwasok       ;Was already set
CAD6:E4 39         160           cpx   kswh          ;Are we in the input hooks
CAD8:D0 47   CB21  161           bne   strts         ;Leaves C=1 if =
CADA:09 40         162           ora   #$40          ;Set term mode bit
CADC:AC 79 06      163           ldy   oldcur        ;Save what was in oldcur
CADF:8C 7A 06      164           sty   oldcur2
CAE2:A0 DF         165           ldy   #termcur      ;Get new cursor value
CAE4:80 07   CAED  166           bra   stset
```

```
CAE6:F0 0E    CAF6  167 stclr     beq   stwasok       ;Branch if already clear
CAE8:29 BF          168           and   #$BF          ;Clear the bit
CAEA:AC 7A 06       169           ldy   oldcur2       ;Restore the cursor
CAED:9D B8 03       170 stset     sta   sermode,x
CAF0:8C 79 06       171           sty   oldcur        ;Save cursor to be restored after command
CAF3:8C FB 07       172           sty   cursor
CAF6:BC 85 C8       173 stwasok   ldy   devno,x
CAF9:58             174           cli                 ;want to leave with interrupts active
CAFA:08             175           php
CAFB:78             176           sei                 ;but off while we twittle bits
CAFC:B9 FA BF       177           lda   scomd,y
CAFF:09 02          178           ora   #$2           ;disable receiver interrupts if
CB01:90 02    CB05  179           bcc   cmdt2         ; not in terminal mode
CB03:29 FD          180           and   #$FD          ;enable when in terminal mode
CB05:99 FA BF       181 cmdt2     sta   scomd,y
CB08:A9 00          182           lda   #0
CB0A:6A             183           ror   a             ;set kbd interrupts according to t-mode
CB0B:8D FA 05       184           sta   typhed
CB0E:10 07    CB17  185           bpl   cmdt3         ;branch if leaving terminal mode
CB10:9C 7F 05       186           stz   twser         ; and ser buf...
CB13:9C 7F 06       187           stz   trser
CB16:8A             188           txa                 ;use x to enable serial buffering
CB17:8D FF 04       189 cmdt3     sta   aciabuf
CB1A:28             190           plp                 ;restore carry, enable interrupts.
CB1B:8E FF 05       191 flush     stx   twkey         ;Flush the type ahead buffer
CB1E:8E FF 06       192           stx   trkey
CB21:60             193 strts     rts
CB22:         0002  194           ds    $CB24-*,$00
CB24:E8             195 zznm2     inx
CB25:4C FB C7       196           jmp   zznml
CB28:9E 0B 40 50    197 comtbl    dfb   $9E,$0B,$40,$50,$16,$0B,$01,$00
CB30:                27           INCLUDE SCROLLING   ;More Video stuff @$CB30
```

```
CB30:                 3 *
CB30:                 4 * SCROLLIT scrolls the screen either up or down, depending
CB30:                 5 * on the value of X.  It scrolls within windows with even
CB30:                 6 * or odd edges for both 40 and 80 columns.  It can scroll
CB30:                 7 * windows down to 1 characters wide.
CB30:                 8 *
CB30:DA               9 SCROLLDN  PHX                 ;save X
CB31:A2 00           10           LDX   #0            ;direction = down
CB33:80 03   CB38    11           BRA   SCROLLIT      ;do scroll
CB35:                12 *
CB35:DA              13 SCROLLUP  PHX                 ;save X
CB36:A2 01           14           LDX   #1            ;direction = up
CB38:A4 21           15 SCROLLIT  LDY   WNDWDTH       ;get width of screen window
CB3A:2C 1F C0        16           BIT   RD80VID       ;in 40 or 80 columns?
CB3D:10 18   CB57    17           BPL   GETST         ;=>40, determine starting line
CB3F:8D 01 C0        18           STA   SET80COL      ;make sure this is enabled
CB42:98              19           TYA                 ;get WNDWDTH for test
CB43:4A              20           LSR   A             ;divide by 2 for 80 column index
CB44:A8              21           TAY                 ;and save
CB45:A5 20           22           LDA   WNDLFT        ;test oddity of right edge
CB47:4A              23           LSR   A             ;by rotating low bit into carry
CB48:B8              24           CLV                 ;V=0 if left edge even
CB49:90 03   CB4E    25           BCC   CHKRT         ;=>check right edge
CB4B:2C C1 CB        26           BIT   SEV1          ;V=1 if left edge odd
CB4E:2A              27 CHKRT     ROL   A             ;restore WNDLFT
CB4F:45 21           28           EOR   WNDWDTH       ;get oddity of right edge
CB51:4A              29           LSR   A             ;C=1 if right edge even
CB52:70 03   CB57    30           BVS   GETST         ;if odd left, don't DEY
CB54:B0 01   CB57    31           BCS   GETST         ;if even right, don't DEY
CB56:88              32           DEY                 ;if right edge odd, need one less
CB57:8C F8 05        33 GETST     STY   TEMPY         ;save window width
CB5A:AD 1F C0        34           LDA   RD80VID       ;N=1 if 80 columns
CB5D:08              35           PHP                 ;save N,Z,V
CB5E:A5 22           36           LDA   WNDTOP        ;assume scroll from top
CB60:E0 00           37           CPX   #0            ;up or down?
CB62:D0 03   CB67    38           BNE   SETDBAS       ;=>up
CB64:A5 23           39           LDA   WNDBTM        ;down, start scrolling at bottom
CB66:3A              40           DEC   A             ;really need one less
CB67:                41 *
CB67:8D 78 05        42 SETDBAS   STA   TEMPA         ;save current line
CB6A:20 24 FC        43           JSR   VTABZ         ;calculate base with window width
CB6D:                44 *
CB6D:A5 28           45 SCRLIN    LDA   BASL          ;current line is destination
CB6F:85 2A           46           STA   BAS2L
CB71:A5 29           47           LDA   BASH
CB73:85 2B           48           STA   BAS2H
CB75:                49 *
CB75:AD 78 05        50           LDA   TEMPA         ;get current line
CB78:E0 00           51           CPX   #0            ;going up?
CB7A:D0 07   CB83    52           BNE   SETUP2        =>up, inc current line
CB7C:C5 22           53           CMP   WNDTOP        ;down. Reached top yet?
CB7E:F0 39   CBB9    54           BEQ   SCRL3         ;yes! clear top line, exit
CB80:3A              55           DEC   A             ;no, go up a line
CB81:80 05   CB88    56           BRA   SETSRC        ;set source for scroll
CB83:1A              57 SETUP2    INC   A             ;up, inc current line
CB84:C5 23           58           CMP   WNDBTM        ;at bottom yet?
CB86:B0 31   CBB9    59           BCS   SCRL3         ;yes! clear bottom line, exit
CB88:                60 *
```

```
CB88:8D 78 05        61 SETSRC    STA   TEMPA          ;save new current line
CB8B:20 24 FC        62           JSR   VTABZ          ;get base for new current line
CB8E:AC F8 05        63           LDY   TEMPY          ;get width for scroll
CB91:28              64           PLP                  ;get status for scroll
CB92:08              65           PHP                  ;N=1 if 80 columns
CB93:10 1F   CBB4    66           BPL   SKPRT          ;=>only do 40 columns
CB95:AD 55 C0        67           LDA   TXTPAGE2       ;scroll aux page first (even bytes)
CB98:98              68           TYA                  ;test Y
CB99:F0 07   CBA2    69           BEQ   SCRLFT         ;if Y=0, only scroll one byte
CB9B:B1 28           70 SCRLEVEN  LDA   (BASL),Y
CB9D:91 2A           71           STA   (BAS2L),Y
CB9F:88              72           DEY
CBA0:D0 F9   CB9B    73           BNE   SCRLEVEN       ;do all but last even byte
CBA2:70 04   CBA8    74 SCRLFT    BVS   SKPLFT         ;odd left edge, skip this byte
CBA4:B1 28           75           LDA   (BASL),Y
CBA6:91 2A           76           STA   (BAS2L),Y
CBA8:AD 54 C0        77 SKPLFT    LDA   TXTPAGE1       ;now do main page (odd bytes)
CBAB:AC F8 05        78           LDY   TEMPY          ;restore width
CBAE:B0 04   CBB4    79           BCS   SKPRT          ;even right edge, skip this byte
CBB0:B1 28           80 SCRLODD   LDA   (BASL),Y
CBB2:91 2A           81           STA   (BAS2L),Y
CBB4:88              82 SKPRT     DEY
CBB5:10 F9   CBB0    83           BPL   SCRLODD
CBB7:80 B4   CB6D    84           BRA   SCRLIN         ;scroll next line
CBB9:                85 *
CBB9:20 A0 FC        86 SCRL3     JSR   CLRLIN         ;clear current line
CBBC:20 22 FC        87           JSR   VTAB           ;restore original cursor line
CBBF:28              88           PLP                  ;pull status off stack
CBC0:FA              89           PLX                  ;restore X
CBC1:60              90 SEV1      RTS                  ;done!!!
```

```
CBC2:                    92 *
CBC2:                    93 * DOCLR is called by CLREOL.  It decides whether
CBC2:                    94 * to do a (quick) 40 or 80 column clear to end of line.
CBC2:                    95 *
CBC2:2C 1F C0            96 DOCLR     BIT   RD80VID        ;40 or 80 column clear?
CBC5:30 13    CBDA       97           BMI   CLR80          ;=>clear 80 columns
CBC7:91 28               98 CLR40     STA   (BASL),Y
CBC9:C8                  99           INY
CBCA:C4 21              100           CPY   WNDWDTH
CBCC:90 F9    CBC7      101           BCC   CLR40
CBCE:60                 102           RTS
CBCF:                   103 *
CBCF:DA                 104 CLRHALF   PHX                  ;clear right half of screen
CBD0:A2 D8              105           LDX   #$D8           ;for SCRN48
CBD2:A0 14              106           LDY   #20
CBD4:A5 32              107           LDA   INVFLG
CBD6:29 A0              108           AND   #$A0
CBD8:80 17    CBF1      109           BRA   CLR2           ;=>jump into middle
CBDA:                   110 *
CBDA:DA                 111 CLR80     PHX                  ;preserve X
CBDB:48                 112           PHA                  ;and blank
CBDC:98                 113           TYA                  ;get count for CH
CBDD:48                 114           PHA                  ;save for left edge check
CBDE:38                 115           SEC                  ;count=WNDWDTH-Y-1
CBDF:E5 21              116           SBC   WNDWDTH
CBE1:AA                 117           TAX                  ;save CH counter
CBE2:98                 118           TYA                  ;div CH by 2 for half pages
CBE3:4A                 119           LSR   A
CBE4:A8                 120           TAY
CBE5:68                 121           PLA                   ;restore original CH
CBE6:45 20              122           EOR   WNDLFT         ;get starting page
CBE8:6A                 123           ROR   A
CBE9:B0 03    CBEE      124           BCS   CLR0
CBEB:10 01    CBEE      125           BPL   CLR0
CBED:C8                 126           INY                  ;iff WNDLFT odd, starting byte odd
CBEE:68                 127 CLR0      PLA                  ;get blankity blank
CBEF:B0 0B    CBFC      128           BCS   CLR1           ;starting page is 1 (default)
CBF1:2C 55 C0           129 CLR2      BIT   TXTPAGE2       ;else do page 2
CBF4:91 28              130           STA   (BASL),Y
CBF6:2C 54 C0           131           BIT   TXTPAGE1       ;now do page 1
CBF9:E8                 132           INX
CBFA:F0 06    CC02      133           BEQ   CLR3           ;all done
CBFC:91 28              134 CLR1      STA   (BASL),Y
CBFE:C8                 135           INY                  ;forward 2 columns
CBFF:E8                 136           INX                  ;next CH
CC00:D0 EF    CBF1      137           BNE   CLR2           ;not done yet
CC02:FA                 138 CLR3      PLX                  ;restore X
CC03:60                 139           RTS                  ;and exit
CC04:                   140 *
CC04:9C FA 05           141 CLRPORT   STZ   TYPHED         ;disable typeahead
CC07:9C F9 05           142           STZ   EXTINT2        ;and external interrupts
CC0A:60                 143           RTS
```

```
CC0B:                 145 *
CC0B:                 146 * PASINVERT is used by Pascal to display the cursor. Pascal
CC0B:                 147 * normally leaves the cursor on the screen at all times. It
CC0B:                 148 * is fleetingly removed while a character is displayed, then
CC0B:                 149 * promptly redisplayed.  CTL-F and CTL-E, respectively,
CC0B:                 150 * disable and enable display of the cursor when printed using
CC0B:                 151 * the Pascal 1.1 entry point (PWRITE).  Screen I/O is
CC0B:                 152 * significantly faster when the cursor is disabled.  This
CC0B:                 153 * feature is supported by Pascal 1.2 and later.
CC0B:                 154 *
CC0B:AD FB 04         155 PASINVERT LDA   VMODE          ;Called by pascal to
CC0E:29 10            156           AND   #M.CURSOR      ;display cursor
CC10:D0 0A    CC1C    157           BNE   INVX           ;=>cursor off, don't invert
CC12:         CC12    158 INVERT    EQU   *
CC12:20 1D CC         159           JSR   PICKY          ;load Y and get char
CC15:48               160           PHA
CC16:49 80            161           EOR   #$80           ;FLIP INVERSE/NORMAL
CC18:20 B3 C3         162           JSR   STORY          ;stuff onto screen
CC1B:68               163           PLA                  ;for RDCHAR
CC1C:60               164 INVX      RTS
CC1D:                 165 *
CC1D:                 166 * PICK lifts a character from the screen in either
CC1D:                 167 * 40 or 80 columns from the current cursor position.
CC1D:                 168 * If the alternate character set is switched in,
CC1D:                 169 * character codes $0-$1F are returned as $40-$5F (which
CC1D:                 170 * is what must have been originally printed to the location).
CC1D:                 171 *
CC1D:5A               172 PICKY     PHY                  ;save Y
CC1E:20 9D CC         173           JSR   GETCUR         ;get newest cursor into Y
CC21:AD 1F C0         174           LDA   RD80VID        ;80 columns?
CC24:10 17    CC3D    175           BPL   PICK1          ;=>no
CC26:8D 01 C0         176           STA   SET80COL       ;force 80STORE if 80 columns
CC29:98               177           TYA
CC2A:45 20            178           EOR   WNDLFT         ;C=1 if char in main RAM
CC2C:6A               179           ROR   A              ;get low bit into carry
CC2D:B0 04    CC33    180           BCS   PICK2          ;=>store in main memory
CC2F:AD 55 C0         181           LDA   TXTPAGE2       ;else switch in page 2
CC32:C8               182           INY                  ;for odd left, aux bytes
CC33:98               183 PICK2     TYA                  ;divide pos'n by 2
CC34:4A               184           LSR   A
CC35:A8               185           TAY                  ;and use as offset into line
CC36:B1 28            186           LDA   (BASL),Y       ;pick character
CC38:8D 54 C0         187           STA   TXTPAGE1       ;80 columns, switch in
CC3B:80 02    CC3F    188           BRA   PICK3          ;skip 40 column pick
CC3D:B1 28            189 PICK1     LDA   (BASL),Y       ;pick 40 column char
CC3F:2C 1E C0         190 PICK3     BIT   ALTCHARSET     ;only allow if alt set
CC42:10 06    CC4A    191           BPL   PICK4
CC44:C9 20            192           CMP   #$20
CC46:B0 02    CC4A    193           BCS   PICK4
CC48:09 40            194           ORA   #$40
CC4A:7A               195 PICK4     PLY                  ;restore real Y
CC4B:60               196           RTS
CC4C:                 197 *
CC4C:                 198 * SHOWCUR displays either a checkerboard cursor, a solid
CC4C:                 199 * rectangle, or the current cursor character, depending
CC4C:                 200 * on the value of the CURSOR location.  0=inverse cursor,
CC4C:                 201 * $FF=checkerboard cursor, anything else is displayed
CC4C:                 202 * after being anded with inverse mask.
```

```
CC4C:                 203 *
CC4C:AC FB 07         204 SHOWCUR   LDY   CURSOR        ;what's my type?
CC4F:D0 02   CC53     205           BNE   NOTINV        ;=>not inverse
CC51:80 BF   CC12     206           BRA   INVERT        ;else invert the char (exit)
CC53:                 207 *
CC53:                 208 * Exit with char in accumulator
CC53:                 209 *
CC53:20 1D CC         210 NOTINV    JSR   PICKY         ;get char on screen
CC56:48               211           PHA                ;preserve it
CC57:8D 7B 07         212           STA   NXTCUR        ;save for update
CC5A:98               213           TYA                ;test for checkerboard
CC5B:C8               214           INY
CC5C:F0 0D   CC6B     215           BEQ   NOTINV2       ;=>checkerboard, display it
CC5E:7A               216           PLY                ;test char
CC5F:5A               217           PHY
CC60:30 09   CC6B     218           BMI   NOTINV2       ;don't need inverse
CC62:AD 1E C0         219           LDA   ALTCHARSET    ;mask = $7F if alternate
CC65:09 7F            220           ORA   #$7F          ; character set,
CC67:4A               221           LSR   A             ;$3F if normal char set
CC68:2D FB 07         222 NOTINV1   AND   CURSOR        ;form char to display
CC6B:20 B3 C3         223 NOTINV2   JSR   STORY         ;and display it
CC6E:68               224           PLA                ;restore real char
CC6F:60               225           RTS
CC70:                 226 *
CC70:                 227 * The UPDATE routine increments the random seed.
CC70:                 228 * If a certain value is reached and we are in Apple II
CC70:                 229 * mode, the blinking check cursor is updated.  If a
CC70:                 230 * key has been pressed, the old char is replaced on the
CC70:                 231 * screen, and we return with BMI.
CC70:                 232 *
CC70:                 233 * NOTE: this routine used by COMM firmware!!
CC70:                 234 *
CC70:48               235 UPDATE    PHA                ;save char
CC71:E6 4E            236           INC   RNDL         ;update seed
CC73:D0 1C   CC91     237           BNE   UD2          ;check for key
CC75:A5 4F            238           LDA   RNDH
CC77:E6 4F            239           INC   RNDH
CC79:45 4F            240           EOR   RNDH
CC7B:29 10            241           AND   #$10         ;need to update cursor?
CC7D:F0 12   CC91     242           BEQ   UD2          ;=>no, check for key
CC7F:AD FB 07         243           LDA   CURSOR       ;what cursor are we using?
CC82:F0 0D   CC91     244           BEQ   UD2          ;=>//e cursor, leave alone
CC84:20 1D CC         245           JSR   PICKY        ;get the character into A
CC87:AC 7B 07         246           LDY   NXTCUR       ;get next character
CC8A:8D 7B 07         247           STA   NXTCUR       ;save next next character
CC8D:98               248           TYA
CC8E:20 B3 C3         249           JSR   STORY        ;and print it
CC91:68               250 UD2       PLA                ;get real char
CC92:20 AD C9         251           JSR   XBITKBD      ;was a key pressed?
CC95:10 28   CCBF     252           BPL   GETCURX      ;=>no key pressed
CC97:20 B3 C3         253 CLRKBD    JSR   STORY        ;restore old key
CC9A:4C 8F C9         254           JMP   XRDKBD       ;look for keystroke and exit
CC9D:                 255 *
CC9D:                 256 * ON CURSORS.  Whenever the horizontal cursor position is
CC9D:                 257 * needed, a call to GETCUR is done. This is the equivalent
CC9D:                 258 * of a LDY CH.  This returns the current cursor for II and
CC9D:                 259 * //e mode, which may have been poked as either CH or OURCH.
CC9D:                 260 *
```

```
CC9D:                 261 * It also forces CH and OLDCH to 0 if 80 column mode active.
CC9D:                 262 * This prevents LDY CH, STA (BASL),Y from trashing non screen
CC9D:                 263 * memory. It works just like the //e.
CC9D:                 264 *
CC9D:                 265 * All routines that update the cursor's horizontal position
CC9D:                 266 * are here.  This ensures that the newest value of the cursor
CC9D:                 267 * is always used, and that 80 column CH is always 0.
CC9D:                 268 *
CC9D:                 269 * GETCUR only affects the Y register
CC9D:                 270 *
CC9D:A4 24            271 GETCUR    LDY   CH            ;if CH=OLDCH, then
CC9F:CC 7B 04         272           CPY   OLDCH         ;OURCH is valid
CCA2:D0 03   CCA7     273           BNE   GETCUR1       ;=>else CH must have been changed
CCA4:AC 7B 05         274           LDY   OURCH         ;use OURCH
CCA7:C4 21            275 GETCUR1   CPY   WNDWDTH       ;is the value too big
CCA9:90 02   CCAD     276           BCC   GETCUR2       ;=>no, fits just fine
CCAB:A0 00            277           LDY   #0            ;else force CH to 0
CCAD:                 278 *
CCAD:                 279 * GETCUR2 is commonly used to set the current cursor
CCAD:                 280 * position when Y can be used.
CCAD:                 281 *
CCAD:8C 7B 05         282 GETCUR2   STY   OURCH         ;update real cursor
CCB0:2C 1F C0         283           BIT   RD80VID       ;80 columns?
CCB3:10 02   CCB7     284           BPL   GETCUR3       ;=>no, set all cursors
CCB5:A0 00            285           LDY   #0            ;yes, peg CH to 0
CCB7:84 24            286 GETCUR3   STY   CH
CCB9:8C 7B 04         287           STY   OLDCH
CCBC:AC 7B 05         288           LDY   OURCH         ;get cursor
CCBF:60               289 GETCURX   RTS                 ;and fly...
CCC0:                  28           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 AD C9        40 ESC0      JSR   XBITKBD       ;check for keystroke
CCDA:10 FB   CCD7    41           BPL   ESC0
CCDC:68              42           PLA                 ;get old char
CCDD:20 97 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:                 234 *
CD9F:                 235 * Disable mouse text by setting M.MOUSE
CD9F:                 236 *
CD9F:A9 01            237 MOUSOFF   LDA   #M.MOUSE
CDA1:0C FB 04         238 SETIT     TSB   VMODE
CDA4:60               239           RTS
CDA5:                 240 *
CDA5:                 241 * EXECUTE HOME:
CDA5:                 242 *
CDA5:20 E9 FE         243 HOMECUR   JSR   CLRCH          ;move cursors to far left
CDA8:A8               244           TAY                 ;(probably not needed)
CDA9:A5 22            245           LDA   WNDTOP        ;and to top of window
CDAB:85 25            246           STA   CV
CDAD:4C 88 FC         247           JMP   NEWVTABZ      ;then set base address, OURCV
CDB0:                 248 *
CDB0:                 249 * EXECUTE "NORMAL VIDEO"
CDB0:                 250 *
CDB0:20 84 FE         251 X.SO      JSR   SETNORM       ;set INVFLG to $FF
CDB3:A9 04            252           LDA   #M.VMODE      ;then clear inverse mode bit
CDB5:80 E4   CD9B     253           BRA   CLRIT
CDB7:                 254 *
CDB7:                 255 * EXECUTE "INVERSE VIDEO"
CDB7:                 256 *
CDB7:20 80 FE         257 X.SI      JSR   SETINV        ;set INVFLG to $3F
CDBA:A9 04            258           LDA   #M.VMODE      ;then set inverse mode bit
CDBC:80 E3   CDA1     259           BRA   SETIT
CDBE:                 260 *
CDBE:                 261 * EXECUTE '40COL MODE' or '80COL MODE':
CDBE:                 262 *
CDBE:38               263 SET80     SEC                 ;flag an 80 column window
CDBF:90               264           DFB   $90           ;BCC opcode (never taken)
CDC0:18               265 SET40     CLC                 ;flag a 40 column window
CDC1:2C FB 04         266           BIT   VMODE         ;but...is it pascal?
CDC4:10 54   CE1A     267           BPL   SETX          ;=>yes, don't execute
CDC6:08               268           PHP                 ;save window size
CDC7:20 1B CE         269           JSR   HOOKITUP      ;COPYROM if needed, set I/O hooks
CDCA:28               270           PLP                 ;and get 40/80
CDCB:80 08   CDD5     271           BRA   WIN0          ;=>set window
CDCD:                 272 *
CDCD:                 273 * CHK80 is called by PR#0 to convert to 40 if it was
CDCD:                 274 * 80. Otherwise the window is left ajar.
CDCD:                 275 *
CDCD:2C 1F C0         276 CHK80     BIT   RD80VID       ;don't set 40 if
CDD0:10 48   CE1A     277           BPL   SETX          ;already 40
CDD2:                 278 *
CDD2:18               279 WIN40     CLC                 ;flag 40 column window
CDD3:B0               280           DFB   $B0           ;BCS opcode (never taken)
CDD4:38               281 WIN80     SEC                 ;flag 80 column window
CDD5:64 22            282 WIN0      STZ   WNDTOP        ;set window top now
CDD7:2C 1A C0         283           BIT   RDTEXT        ;for text or mixed
CDDA:30 04   CDE0     284           BMI   WIN1          ;=>text
CDDC:A9 14            285           LDA   #20
CDDE:85 22            286           STA   WNDTOP        ;used by 80<->40 conversion
CDE0:2C 1F C0         287 WIN1      BIT   RD80VID       ;80 columns now?
CDE3:08               288           PHP                 ;save 80 or 40
CDE4:B0 07   CDED     289           BCS   WIN2          ;=>80: convert if 40
CDE6:10 0A   CDF2     290           BPL   WIN3          ;=>40: no convert
CDE8:20 53 CE         291           JSR   SCRN84        ;80: convert to 40
```

```
CDEB:80 05    CDF2  292           BRA   WIN3          ;done converting
CDED:30 03    CDF2  293 WIN2      BMI   WIN3          ;=>80: no convert
CDEF:20 80 CE       294           JSR   SCRN48        ;40: convert to 80
CDF2:20 9D CC       295 WIN3      JSR   GETCUR        ;determine absolute CH
CDF5:98             296           TYA                 ;in case the window setting
CDF6:18             297           CLC                 ;was different
CDF7:65 20          298           ADC   WNDLFT
CDF9:28             299           PLP                 ;pin to right edge if
CDFA:B0 06    CE02  300           BCS   WIN4          ;80 to 40 leaves cursor
CDFC:C9 28          301           CMP   #40           ;off the screen
CDFE:90 02    CE02  302           BCC   WIN4
CE00:A9 27          303           LDA   #39
CE02:20 EC FE       304 WIN4      JSR   SETCUR        ;set new cursor
CE05:A5 25          305           LDA   CV            ;set new base address
CE07:20 C1 FB       306           JSR   BASCALC       ;for left = 0 (always)
CE0A:               307 *
CE0A:64 20          308 WNDREST   STZ   WNDLFT        ;Called by INIT and Pascal
CE0C:A9 18          309           LDA   #$18          ;and bottom
CE0E:85 23          310           STA   WNDBTM
CE10:A9 28          311           LDA   #$28          ;set left,width,bottom
CE12:2C 1F C0       312           BIT   RD80VID       ;set width to 80 if 80 columns
CE15:10 01    CE18  313           BPL   WIN5
CE17:0A             314           ASL   A
CE18:85 21          315 WIN5      STA   WNDWDTH       ;set width
CE1A:60             316 SETX      RTS                 ;exit used by SET40/80
CE1B:               317 *
CE1B:               318 * Turn on video firmware:
CE1B:               319 *
CE1B:               320 * This routine is used by BASIC init, ESC-4, ESC-8
CE1B:               321 * It copies the Monitor ROM to the language card
CE1B:               322 * if necessary; it sets the input and output hooks to
CE1B:               323 * $C30x; it sets all switches for video firmware operation
CE1B:               324 *
CE1B:2C 7B 06       325 HOOKITUP  BIT   VFACTV        ;don't touch hooks
CE1E:10 11    CE31  326           BPL   VIDMODE       ;if video firmware already active
CE20:20 38 C3       327 HOOKUP    JSR   COPYROM       ;Copy ROM to LC?
CE23:A9 05          328 SETHOOKS  LDA   #>C3KEYIN     ;set up $C300 hooks
CE25:85 38          329           STA   KSWL
CE27:A9 07          330           LDA   #>C3COUT1
CE29:85 36          331           STA   CSWL
CE2B:A9 C3          332           LDA   #<C3COUT1
CE2D:85 39          333           STA   KSWH
CE2F:85 37          334           STA   CSWH
CE31:               335 *
CE31:               336 * Now set the video firmware active
CE31:               337 *
CE31:9C FB 07       338 VIDMODE   STZ   CURSOR        ;set a solid inverse cursor
CE34:A9 08          339           LDA   #M.CTL        ;preserve M.CTL bit
CE36:2D FB 04       340           AND   VMODE
CE39:09 81          341           ORA   #M.PASCAL+M.MOUSE ;no pascal,mouse
CE3B:               342 *
CE3B:               343 * Pascal calls here to set its mode
CE3B:               344 *
CE3B:8D FB 04       345 PVMODE    STA   VMODE         ;set mode bits
CE3E:9C 7B 06       346           STZ   VFACTV        ;say video firmware active
CE41:8D 0F C0       347           STA   SETALTCHAR    ;and set alternate char set
CE44:60             348 QX        RTS
CE45:               349 *
```

```
CE45:                 350 * QUIT converts the screen from 80 to 40 if necessary,
CE45:                 351 * sets a 40 column window, and restores the normal I/O
CE45:                 352 * hooks (COUT1 and KEYIN).
CE45:                 353 *
CE45:2C FB 04         354 QUIT      BIT   VMODE         ;no quitting from pascal
CE48:10 FA   CE44     355           BPL   QX
CE4A:20 D2 CD         356           JSR   WIN40         ;first, do an escape 4
CE4D:20 89 FE         357 ZZQUIT    JSR   SETKBD        ;do a IN#0  (used by COMM)
CE50:4C 93 FE         358           JMP   SETVID        ;and a PR#0
```

```
CE53:                  360 *
CE53:                  361 * SCRN84 and SCRN48 convert screens between 40 & 80 cols.
CE53:                  362 * WNDTOP must be set up to indicate the last line to
CE53:                  363 * be done.  All registers are trashed.
CE53:                  364 *
CE53:A2 17             365 SCRN84     LDX   #23           ;start at bottom of screen
CE55:8D 01 C0          366            STA   SET80COL      ;allow page 2 access
CE58:8A                367 SCR1       TXA                 ;calc base for line
CE59:20 C1 FB          368            JSR   BASCALC
CE5C:A0 27             369            LDY   #39           ;start at right of screen
CE5E:5A                370 SCR2       PHY                 ;save 40 index
CE5F:98                371            TYA                 ;div by 2 for 80 column index
CE60:4A                372            LSR   A
CE61:B0 03   CE66      373            BCS   SCR3
CE63:2C 55 C0          374            BIT   TXTPAGE2      ;even column, do page 2
CE66:A8                375 SCR3       TAY                 ;get 80 index
CE67:B1 28             376            LDA   (BASL),Y      ;get 80 char
CE69:2C 54 C0          377            BIT   TXTPAGE1      ;restore page1
CE6C:7A                378            PLY                 ;get 40 index
CE6D:91 28             379            STA   (BASL),Y
CE6F:88                380            DEY
CE70:10 EC   CE5E      381            BPL   SCR2          ;do next 40 byte
CE72:CA                382            DEX                 ;do next line
CE73:30 04   CE79      383            BMI   SCR4          ;=>done with setup
CE75:E4 22             384            CPX   WNDTOP        ;at top yet?
CE77:B0 DF   CE58      385            BCS   SCR1
CE79:8D 00 C0          386 SCR4       STA   CLR80COL      ;clear 80STORE for 40 columns
CE7C:8D 0C C0          387            STA   CLR80VID      ;clear 80VID for 40 columns
CE7F:60                388            RTS
CE80:                  389 *
CE80:A2 17             390 SCRN48     LDX   #23           ;start at bottom of screen
CE82:8A                391 SCR5       TXA                 ;set base for current line
CE83:20 C1 FB          392            JSR   BASCALC
CE86:A0 00             393            LDY   #0            ;start at left of screen
CE88:8D 01 C0          394            STA   SET80COL      ;enable page2 store
CE8B:B1 28             395 SCR6       LDA   (BASL),Y      ;get 40 column char
CE8D:5A                396 SCR8       PHY                 ;save 40 column index
CE8E:48                397            PHA                 ;save char
CE8F:98                398            TYA                 ;div 2 for 80 column index
CE90:4A                399            LSR   A
CE91:B0 03   CE96      400            BCS   SCR7          ;save on page1
CE93:8D 55 C0          401            STA   TXTPAGE2
CE96:A8                402 SCR7       TAY                 ;get 80 column index
CE97:68                403            PLA                 ;now save character
CE98:91 28             404            STA   (BASL),Y
CE9A:8D 54 C0          405            STA   TXTPAGE1      ;flip page1
CE9D:7A                406            PLY                 ;restore 40 column index
CE9E:C8                407            INY                 ;move to the right
CE9F:C0 28             408            CPY   #40           ;at right yet?
CEA1:90 E8   CE8B      409            BCC   SCR6          ;=>no, do next column
CEA3:20 CF CB          410            JSR   CLRHALF       ;clear half of screen
CEA6:CA                411            DEX                 ;else do next line of screen
CEA7:30 04   CEAD      412            BMI   SCR9          ;=>done with top line
CEA9:E4 22             413            CPX   WNDTOP        ;at top yet?
CEAB:B0 D5   CE82      414            BCS   SCR5
CEAD:8D 0D C0          415 SCR9       STA   SET80VID      ;convert to 80 columns
CEB0:60                416            RTS
CEB1:                   29            INCLUDE PASCAL      ;Pascal support stuff
```

```
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 AD C9        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
```

```
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 8F C9       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:             30            INCLUDE AUXSTUFF   ;Aux RAM routines
```

```
CF86:                    4 ****************************************
CF86:                    5 * NAME     : MOVEAUX
CF86:                    6 * FUNCTION: PERFORM CROSSBANK MEMORY MOVE
CF86:                    7 * INPUT    : A1=SOURCE ADDRESS
CF86:                    8 *          : A2=SOURCE END
CF86:                    9 *          : A4=DESTINATION START
CF86:                   10 *          : CARRY SET=MAIN-->CARD
CF86:                   11 *                CLR=CARD-->MAIN
CF86:                   12 * OUTPUT   : NONE
CF86:                   13 * VOLATILE: NOTHING
CF86:                   14 * CALLS    : NOTHING
CF86:                   15 ****************************************
CF86:           CF86    16 MOVEAUX   EQU   *
CF86:48                 17           PHA                  ;SAVE AC
CF87:AD 13 C0           18           LDA   RDRAMRD        ;SAVE STATE OF
CF8A:48                 19           PHA                  ; MEMORY FLAGS
CF8B:AD 14 C0           20           LDA   RDRAMWRT
CF8E:48                 21           PHA
CF8F:                   22 *
CF8F:                   23 * SET FLAGS FOR CROSSBANK MOVE:
CF8F:                   24 *
CF8F:90 08    CF99      25           BCC   MOVEC2M        ;=>CARD-->MAIN
CF91:8D 02 C0           26           STA   RDMAINRAM      ;SET FOR MAIN
CF94:8D 05 C0           27           STA   WRCARDRAM      ; TO CARD
CF97:B0 06    CF9F      28           BCS   MOVESTRT       ;=>(ALWAYS TAKEN)
CF99:                   29 *
CF99:           CF99    30 MOVEC2M   EQU   *
CF99:8D 04 C0           31           STA   WRMAINRAM      ;SET FOR CARD
CF9C:8D 03 C0           32           STA   RDCARDRAM      ; TO MAIN
CF9F:                   33 *
CF9F:           CF9F    34 MOVESTRT  EQU   *
CF9F:B2 3C              35 MOVELOOP  LDA   (A1L)          ;get a byte
CFA1:92 42              36           STA   (A4L)          ;move it
CFA3:E6 42              37           INC   A4L
CFA5:D0 02    CFA9      38           BNE   NEXTA1
CFA7:E6 43              39           INC   A4H
CFA9:A5 3C              40 NEXTA1    LDA   A1L
CFAB:C5 3E              41           CMP   A2L
CFAD:A5 3D              42           LDA   A1H
CFAF:E5 3F              43           SBC   A2H
CFB1:E6 3C              44           INC   A1L
CFB3:D0 02    CFB7      45           BNE   C01
CFB5:E6 3D              46           INC   A1H
CFB7:90 E6    CF9F      47 C01       BCC   MOVELOOP       ;=>more to move
CFB9:                   48 *
CFB9:8D 04 C0           49           STA   WRMAINRAM      ;CLEAR FLAG2
CFBC:68                 50           PLA                  ;GET ORIGINAL STATE
CFBD:10 03    CFC2      51           BPL   C03            ;=>IT WAS OFF
CFBF:8D 05 C0           52           STA   WRCARDRAM
CFC2:           CFC2    53 C03       EQU   *
CFC2:8D 02 C0           54           STA   RDMAINRAM      ;CLEAR FLAG1
CFC5:68                 55           PLA                  ;GET ORIGINAL STATE
CFC6:10 03    CFCB      56           BPL   MOVERET        ;=>IT WAS OFF
CFC8:8D 03 C0           57           STA   RDCARDRAM
CFCB:           CFCB    58 MOVERET   EQU   *
CFCB:68                 59           PLA                  ;Restore AC
CFCC:60                 60           RTS
```

```
CFCD:                    62 ****************************************
CFCD:                    63 * NAME     : XFER
CFCD:                    64 * FUNCTION: TRANSFER CONTROL CROSSBANK
CFCD:                    65 * INPUT    : $03ED=TRANSFER ADDR
CFCD:                    66 *          : CARRY SET=XFER TO CARD
CFCD:                    67 *                CLR=XFER TO MAIN
CFCD:                    68 *          : VFLAG CLR=USE STD ZP/STK
CFCD:                    69 *          :       SET=USE ALT ZP/STK
CFCD:                    70 * OUTPUT   : NONE
CFCD:                    71 * VOLATILE: $03ED/03EE IN DEST BANK
CFCD:                    72 * CALLS    : NOTHING
CFCD:                    73 * NOTE     : ENTERED VIA JMP, NOT JSR
CFCD:                    74 ****************************************
CFCD:                    75 *
CFCD:          CFCD      76 XFER       EQU   *
CFCD:48                  77            PHA                  ;SAVE AC ON CURRENT STACK
CFCE:                    78 *
CFCE:                    79 * COPY DESTINATION ADDRESS TO THE
CFCE:                    80 *  OTHER BANK SO THAT WE HAVE IT
CFCE:                    81 *  IN CASE WE DO A SWAP:
CFCE:                    82 *
CFCE:AD ED 03            83            LDA   $03ED          ;GET XFERADDR LO
CFD1:48                  84            PHA                  ;SAVE ON CURRENT STACK
CFD2:AD EE 03            85            LDA   $03EE          ;GET XFERADDR HI
CFD5:48                  86            PHA                  ;SAVE IT TOO
CFD6:                    87 *
CFD6:                    88 * SWITCH TO APPROPRIATE BANK:
CFD6:                    89 *
CFD6:90 08    CFE0       90            BCC   XFERC2M        ;=>CARD-->MAIN
CFD8:8D 03 C0            91            STA   RDCARDRAM      ;SET FOR RUNNING
CFDB:8D 05 C0            92            STA   WRCARDRAM      ; IN CARD RAM
CFDE:B0 06    CFE6       93            BCS   XFERZP         ;=> always taken
CFE0:          CFE0      94 XFERC2M    EQU   *
CFE0:8D 02 C0            95            STA   RDMAINRAM      ;SET FOR RUNNING
CFE3:8D 04 C0            96            STA   WRMAINRAM      ; IN MAIN RAM
CFE6:                    97 *
CFE6:          CFE6      98 XFERZP     EQU   *              ;SWITCH TO ALT ZP/STK
CFE6:68                  99            PLA                  ;STUFF XFERADDR
CFE7:8D EE 03           100            STA   $03EE          ; HI AND
CFEA:68                 101            PLA
CFEB:8D ED 03           102            STA   $03ED          ;  LO
CFEE:68                 103            PLA                  ;RESTORE AC
CFEF:70 05    CFF6      104            BVS   XFERAZP        ;=>switch in alternate zp
CFF1:8D 08 C0           105            STA   SETSTDZP       ;else force standard zp
CFF4:50 03    CFF9      106            BVC   JMPDEST        ;=>always perform transfer
CFF6:8D 09 C0           107 XFERAZP   STA   SETALTZP       ;switch in alternate zp
CFF9:6C ED 03           108 JMPDEST   JMP   ($03ED)        ;=>off we go
CFFC:                   109 ****************************************
CFFC:          0004     110            DS    $D000-*,$00
----- NEXT OBJECT FILE NAME IS FIRM.1
F800:          F800      31            ORG   F8ORG
F800:                    32            INCLUDE AUTOST1      ;F8 monitor rom
```

```
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 INSDS2   LDA   (PCL,X)        ;GET OPCODE
F88E:A8             92          TAY
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 *
FA40:48          445 IRQ       PHA                ;save accumulator
FA41:68          446           PLA                ;rescued by stack trick later
FA42:68          447           PLA
FA43:4C 06 C8    448           JMP   IRQ1         ;do rest of IRQ handler
FA46:            449 *
FA46:EA          450           NOP
FA47:            451 *
FA47:            452 * NEWBRK is called by the interrupt handler which has
FA47:            453 * set the hardware to its default state and encoded
FA47:            454 * the state in the accumulator.  Software that wants
FA47:            455 * to do break processing using full system resources
FA47:            456 * can restore the machine state from this value.
FA47:            457 *
FA47:85 44       458 NEWBRK    STA   MACSTAT      ;save state of machine
FA49:7A          459           PLY                ;restore registers for save
FA4A:FA          460           PLX
FA4B:68          461           PLA
FA4C:            462 *
FA4C:28          463 BREAK     PLP                ;Note: same as old BREAK routine!!
FA4D:20 4A FF    464           JSR   SAVE         ;save reg's on BRK
FA50:68          465           PLA                ;including PC
FA51:85 3A       466           STA   PCL
```

```
FA53:68             467           PLA
FA54:85 3B          468           STA   PCH
FA56:6C F0 03       469           JMP   (BRKV)          ;call BRK HANDLER
FA59:               470 *
FA59:20 82 F8       471 OLDBRK    JSR   INSDS1           ;PRINT USER PC
FA5C:20 DA FA       472           JSR   RGDSP1           ;  AND REGS
FA5F:4C 65 FF       473           JMP   MON              ;GO TO MONITOR (NO PASS GO, NO $200!)
FA62:               474 *
FA62:D8             475 RESET     CLD                    ;DO THIS FIRST THIS TIME
FA63:20 84 FE       476           JSR   SETNORM
FA66:20 2F FB       477           JSR   INIT
FA69:20 93 FE       478           JSR   SETVID
FA6C:20 89 FE       479           JSR   SETKBD
FA6F:20 1C C4       480           JSR   INITMOUSE        ;initialize the mouse
FA72:20 04 CC       481           JSR   CLRPORT          ;clear port setup bytes
FA75:9C FF 04       482           STZ   ACIABUF          ;and the commahead buffer
FA78:AD 5F C0       483           LDA   SETAN3           ; AN3 = TTL HI
FA7B:20 BD FA       484           JSR   RESET.X          ; initialize other devices
FA7E:2C 10 C0       485           BIT   KBDSTRB          ; CLEAR KEYBOARD
FA81:D8             486 NEWMON    CLD
FA82:20 3A FF       487           JSR   BELL             ; CAUSES DELAY IF KEY BOUNCES
FA85:AD F3 03       488           LDA   SOFTEV+1         ;IS RESET HI
FA88:49 A5          489           EOR   #$A5             ;A FUNNY COMPLEMENT OF THE
FA8A:CD F4 03       490           CMP   PWREDUP          ; PWR UP BYTE ???
FA8D:D0 17   FAA6   491           BNE   PWRUP            ; NO SO PWRUP
FA8F:AD F2 03       492           LDA   SOFTEV           ; YES SEE IF COLD START
FA92:D0 3B   FACF   493           BNE   NOFIX            ; HAS BEEN DONE YET?
FA94:A9 E0          494           LDA   #$E0             ; DOES SEV POINT AT BASIC?
FA96:CD F3 03       495           CMP   SOFTEV+1
FA99:D0 34   FACF   496           BNE   NOFIX            ; YES SO REENTER SYSTEM
FA9B:A0 03          497 FIXSEV    LDY   #3               ; NO SO POINT AT WARM START
FA9D:8C F2 03       498           STY   SOFTEV           ; FOR NEXT RESET
FAA0:4C 00 E0       499           JMP   BASIC            ; AND DO THE COLD START
FAA3:               500 *
FAA3:20 3A FF       501 BEEPFIX   JSR   BELL             ;Beep on powerup
FAA6:               502 *
FAA6:20 CA FC       503 PWRUP     JSR   COLDSTART        ;Trash memory, init ports
FAA9:        FAA9   504 SETPG3    EQU   *                ; SET PAGE 3 VECTORS
FAA9:A2 05          505           LDX   #5
FAAB:BD FC FA       506 SETPLP    LDA   PWRCON-1,X       ; WITH CNTRL B ADRS
FAAE:9D EF 03       507           STA   BRKV-1,X         ; OF CURRENT BASIC
FAB1:CA             508           DEX
FAB2:D0 F7   FAAB   509           BNE   SETPLP
FAB4:A9 C6          510           LDA   #$C6             ; LOAD HI SLOT +1
FAB6:80 5A   FB12   511           BRA   PWRUP2           ;branch around mnemonics
FAB8:               512 *
FAB8:               513 * Extension to MNEML (left mnemonics)
FAB8:               514 *
FAB8:8A             515           DFB   $8A              ;PHY
FAB9:8B             516           DFB   $8B              ;PLY
FABA:A5             517           DFB   $A5              ;STZ
FABB:AC             518           DFB   $AC              ;TRB
FABC:00             519           DFB   $00              ;???
FABD:               520 *
FABD:               521 * This extension to the monitor reset routine ($FA62)
FABD:               522 * checks for apple keys.  If both are pressed, it goes
FABD:               523 * into an exerciser mode.  If the open apple key only is
FABD:               524 * pressed, memory is selectively trashed and a cold start
```

```
FABD:                525 * is done.
FABD:                526 *
FABD:A9 FF           527 RESET.X   LDA   #$FF
FABF:8D FB 04        528           STA   VMODE          ;initialize mode
FAC2:0E 62 C0        529           ASL   BUTN1
FAC5:2C 61 C0        530           BIT   BUTN0
FAC8:10 64   FB2E    531           BPL   RTS2D
FACA:90 D7   FAA3    532           BCC   BEEPFIX        ;open apple only, reboot
FACC:4C 7C C7        533           JMP   BANGER         ;both apples, exercise 'er
FACF:                534 *
FACF:6C F2 03        535 NOFIX     JMP   (SOFTEV)
FAD2:                536 *
FAD2:C1 D8 D9 D0     537 RTBL      ASC   'AXYPS'
FAD7:                538 *
FAD7:20 8E FD        539 REGDSP    JSR   CROUT          ;DISPLAY USER REG CONTENTS
FADA:A9 45           540 RGDSP1    LDA   #$45           ;WITH LABELS
FADC:85 40           541           STA   A3L
FADE:A9 00           542           LDA   #$00
FAE0:85 41           543           STA   A3H
FAE2:A2 FB           544           LDX   #$FB
FAE4:A9 A0           545 RDSP1     LDA   #$A0
FAE6:20 ED FD        546           JSR   COUT
FAE9:BD D7 F9        547           LDA   RTBL-251,X
FAEC:20 ED FD        548           JSR   COUT
FAEF:A9 BD           549           LDA   #$BD
FAF1:20 ED FD        550           JSR   COUT
FAF4:B5 4A           551           LDA   ACC+5,X
FAF6:80 0A   FB02    552           BRA   RGDSP2         ;make room for mnemonics
FAF8:                553 *
FAF8:                554 * Right half of new mnemonics, indexed from MNEMR
FAF8:                555 *
FAF8:74              556           DFB   $74            ;PHY
FAF9:74              557           DFB   $74            ;PLY
FAFA:76              558           DFB   $76            ;STZ
FAFB:C6              559           DFB   $C6            ;TRB
FAFC:00              560           DFB   $00            ;???
FAFD:                561 *
FAFD:59 FA           562 PWRCON    DW    OLDBRK
FAFF:00 E0 45        563           DFB   $00,$E0,$45
FB02:                564 *
FB02:20 DA FD        565 RGDSP2    JSR   PRBYTE
FB05:E8              566           INX
FB06:30 DC   FAE4    567           BMI   RDSP1
FB08:60              568           RTS
FB09:                569 *
FB09:C1 F0 F0 EC     570 TITLE     ASC   'Apple         ]['
FB11:C4              571           DFB   $C4            ;optional filler
FB12:                572 *
FB12:86 00           573 PWRUP2    STX   LOC0           ; SETPG3 MUST RETURN X=0
FB14:85 01           574           STA   LOC1           ; SET PTR H
FB16:20 60 FB        575           JSR   APPLEII        ;Display our banner...
FB19:6C 00 00        576           JMP   (LOC0)         ;JUMP $C600
FB1C:00              577           BRK
FB1D:00              578           BRK
FB1E:                579 *
FB1E:4C DE C7        580 PREAD     JMP   MPADDLE        ;read mouse paddle
FB21:A0 00           581           LDY   #$00           ;INIT COUNT
FB23:EA              582           NOP                  ;COMPENSATE FOR 1ST COUNT
```

```
FB24:EA              583           NOP
FB25:BD 64 C0        584 PREAD2    LDA   PADDL0,X        ;COUNT Y-REG EVERY 12 USEC.
FB28:10 04   FB2E    585           BPL   RTS2D
FB2A:C8              586           INY
FB2B:D0 F8   FB25    587           BNE   PREAD2          ;EXIT AT 255 MAX
FB2D:88              588           DEY
FB2E:60              589 RTS2D     RTS
FB2F:                 33           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   CTLCHARO     ;execute control char
FB59:                   22 *
FB59:A9 17              23 VTAB23    LDA   #$17          ;VTAB TO ROW 23
FB5B:85 25              24 TABV      STA   CV            ;VTABS TO ROW IN A-REG
FB5D:4C 22 FC           25           JMP   VTAB         ;don't set OURCV!!
FB60:                   26 *
FB60:20 58 FC           27 APPLEII   JSR   HOME          ;CLEAR THE SCRN
FB63:A0 09              28           LDY   #9
FB65:B9 02 FD           29 STITLE    LDA   APPLE2C-1,Y   ;GET A CHAR
FB68:99 0D 04           30           STA   LINE1+13,Y    ;PUT IT AT TOP CENTER OF SCREEN
FB6B:88                 31           DEY
FB6C:D0 F7   FB65       32           BNE   STITLE
FB6E:60                 33           RTS
FB6F:                   34 *
FB6F:AD F3 03           35 SETPWRC   LDA   SOFTEV+1      ;ROUTINE TO CALCULATE THE 'FUNNY
FB72:49 A5              36           EOR   #$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:00              75             BRK
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.
```

```
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
FCE4:38             258             SEC
FCE5:A2 88          259             LDX   #$88
FCE7:BD 27 CB       260 COM1        LDA   COMTBL-1,X      ;XFER from rom
FCEA:90 0A    FCF6  261             BCC   COM2            ;branch if defaults ok
FCEC:DD 77 04       262             CMP   $477,X          ;test for prior setup
FCEF:18             263             CLC                   ;branch if not valid
FCF0:D0 04    FCF6  264             BNE   COM2            ;If $4F8 & $4FF = TBL values
FCF2:E0 82          265             CPX   #$82
FCF4:90 06    FCFC  266             BCC   COM3
FCF6:9D 77 04       267 COM2        STA   $477,X
FCF9:CA             268             DEX                   ;move all 8...
FCFA:D0 EB    FCE7  269             BNE   COM1
FCFC:AD 54 C0       270 COM3        LDA   TXTPAGE1        ;restore switches
FCFF:8D 00 C0       271             STA   CLR80COL        ;to default states
FD02:60             272             RTS
FD03:               273 *
FD03:               274             MSB   ON
FD03:C1 F0 F0 EC    275 APPLE2C     ASC   "Apple          //c"
FD0C:               276 *
FD0C:A4 24          277 RDKEY       LDY   CH              ;get char at current position
FD0E:B1 28          278             LDA   (BASL),Y        ;for those who restore it
FD10:EA             279             NOP                   ;if a program controls input
FD11:EA             280             NOP                   ;hooks, no cursor may be displayed
FD12:EA             281             NOP
FD13:EA             282             NOP
FD14:EA             283             NOP
FD15:EA             284             NOP
FD16:EA             285             NOP
FD17:EA             286             NOP
FD18:               287 *
FD18:6C 38 00       288 KEYINO      JMP   (KSWL)          ;GO TO USER KEY-IN
FD1B:               289 *
FD1B:91 28          290 KEYIN       STA   (BASL),Y        ;erase false images
FD1D:20 4C CC       291             JSR   SHOWCUR         ;display true cursor
```

```
FD20:20 70 CC      292 DONXTCUR  JSR   UPDATE        ;look for key, blink II cursor
FD23:10 FB   FD20  293           BPL   DONXTCUR      ;loop until keypress
FD25:48            294 GOTKEY    PHA                 ;save character
FD26:A9 08         295           LDA   #M.CTL        ;were escapes enabled?
FD28:2C FB 04      296           BIT   VMODE
FD2B:D0 1D   FD4A  297           BNE   NOESC2        ;=>no, there is no escape
FD2D:68            298           PLA                 ;yes, there may be a way out!!
FD2E:C9 9B         299           CMP   #ESC          ;escape?
FD30:D0 06   FD38  300           BNE   LOOKPICK      ;=>no escape
FD32:4C CC CC      301           JMP   NEWESC        ;=>go do escape sequence
FD35:              302 *
FD35:4C ED CC      303 RDCHAR    JMP   ESCRDKEY      ;do RDKEY with escapes
FD38:              304 *
FD38:2C 7B 06      305 LOOKPICK  BIT   VFACTV        ;only process f.arrow
FD3B:30 07   FD44  306           BMI   NOESCAPE      ;if video firmware is active
FD3D:C9 95         307           CMP   #PICK         ;was it PICK? (->,CTL-U)
FD3F:D0 03   FD44  308           BNE   NOESCAPE      ;no, just return
FD41:20 1D CC      309           JSR   PICKY         ;yes, pick the character
FD44:              310 *
FD44:              311 * NOESCAPE is used by GETCOUT too.
FD44:              312 *
FD44:48            313 NOESCAPE  PHA                 ;save it
FD45:A9 08         314 NOESC1    LDA   #M.CTL        ;disable escape sequences
FD47:0C FB 04      315           TSB   VMODE         ;and enable controls
FD4A:68            316 NOESC2    PLA                 ;by setting M.CTL
FD4B:60            317           RTS
FD4C:              318 *
FD4C:EA            319           NOP
FD4D:              320 *
FD4D:20 A6 C3      321 NOTCR     JSR   GETCOUT       ;disable controls and print
FD50:C9 88         322           CMP   #$88           ;CHECK FOR EDIT KEYS
FD52:F0 1D   FD71  323           BEQ   BCKSPC         ;  - BACKSPACE
FD54:C9 98         324           CMP   #$98
FD56:F0 0A   FD62  325           BEQ   CANCEL         ;  - CONTROL-X
FD58:E0 F8         326           CPX   #$F8
FD5A:90 03   FD5F  327           BCC   NOTCR1         ;MARGIN?
FD5C:20 3A FF      328           JSR   BELL           ; YES, SOUND BELL
FD5F:E8            329 NOTCR1    INX                  ;ADVANCE INPUT INDEX
FD60:D0 13   FD75  330           BNE   NXTCHAR
FD62:A9 DC         331 CANCEL    LDA   #$DC           ;BACKSLASH AFTER CANCELLED LINE
FD64:20 A6 C3      332           JSR   GETCOUT
FD67:20 8E FD      333 GETLNZ    JSR   CROUT          ;OUTPUT 'CR'
FD6A:A5 33         334 GETLN     LDA   PROMPT         ;OUTPUT PROMPT CHAR
FD6C:20 ED FD      335           JSR   COUT
FD6F:A2 01         336 GETLN1    LDX   #$01           ;INIT INPUT INDEX
FD71:8A            337 BCKSPC    TXA
FD72:F0 F3   FD67  338           BEQ   GETLNZ         ;WILL BACKSPACE TO 0
FD74:CA            339           DEX
FD75:20 ED CC      340 NXTCHAR   JSR   ESCRDKEY      ;do new RDCHAR (allow escapes)
FD78:C9 95         341           CMP   #PICK          ;USE SCREEN CHAR
FD7A:D0 08   FD84  342           BNE   ADDINP         ; FOR CONTROL-U
FD7C:20 1D CC      343           JSR   PICKY          ;lift char from screen
FD7F:EA            344           NOP
FD80:EA            345           NOP
FD81:EA            346           NOP                  ;no upshifting needed
FD82:EA            347           NOP
FD83:EA            348           NOP
FD84:9D 00 02      349 ADDINP    STA   IN,X           ;ADD TO INPUT BUFFER
```

```
FD87:C9 8D          350           CMP   #$8D
FD89:D0 C2   FD4D  351           BNE   NOTCR
FD8B:20 9C FC      352 CROUT1    JSR   CLREOL        ;CLR TO EOL IF CR
FD8E:A9 8D         353 CROUT     LDA   #$8D
FD90:D0 5B   FDED  354           BNE   COUT          ;(ALWAYS)
FD92:              355 *
FD92:A4 3D         356 PRA1      LDY   A1H           ;PRINT CR,A1 IN HEX
FD94:A6 3C         357           LDX   A1L
FD96:20 8E FD      358 PRYX2     JSR   CROUT
FD99:20 40 F9      359           JSR   PRNTYX
FD9C:A0 00         360           LDY   #$00
FD9E:A9 AD         361           LDA   #$AD          ;PRINT '-'
FDA0:4C ED FD      362           JMP   COUT
FDA3:              363 *
FDA3:A5 3C         364 XAM8      LDA   A1L
FDA5:09 07         365           ORA   #$07          ;SET TO FINISH AT
FDA7:85 3E         366           STA   A2L           ; MOD 8=7
FDA9:A5 3D         367           LDA   A1H
FDAB:85 3F         368           STA   A2H
FDAD:A5 3C         369 MOD8CHK   LDA   A1L
FDAF:29 07         370           AND   #$07
FDB1:D0 03   FDB6  371           BNE   DATAOUT
FDB3:20 92 FD      372 XAM       JSR   PRA1
FDB6:A9 A0         373 DATAOUT   LDA   #$A0
FDB8:20 ED FD      374           JSR   COUT          ;OUTPUT BLANK
FDBB:B1 3C         375           LDA   (A1L),Y
FDBD:20 DA FD      376           JSR   PRBYTE        ;OUTPUT BYTE IN HEX
FDC0:20 BA FC      377           JSR   NXTA1
FDC3:90 E8   FDAD  378           BCC   MOD8CHK       ;NOT DONE YET. GO CHECK MOD 8
FDC5:60            379 RTS4C     RTS                 ;DONE.
FDC6:              380 *
FDC6:4A            381 XAMPM     LSR   A             ;DETERMINE IF MONITOR MODE IS
FDC7:90 EA   FDB3  382           BCC   XAM           ; EXAMINE, ADD OR SUBTRACT
FDC9:4A            383           LSR   A
FDCA:4A            384           LSR   A
FDCB:A5 3E         385           LDA   A2L
FDCD:90 02   FDD1  386           BCC   ADD
FDCF:49 FF         387           EOR   #$FF          ;FORM 2'S COMPLEMENT FOR SUBTRACT.
FDD1:65 3C         388 ADD       ADC   A1L
FDD3:48            389           PHA
FDD4:A9 BD         390           LDA   #$BD          ;PRINT '=', THEN RESULT
FDD6:20 ED FD      391           JSR   COUT
FDD9:68            392           PLA
FDDA:              393 *
FDDA:48            394 PRBYTE    PHA                 ;PRINT BYTE AS 2 HEX DIGITS
FDDB:4A            395           LSR   A             ; (DESTROYS A-REG)
FDDC:4A            396           LSR   A
FDDD:4A            397           LSR   A
FDDE:4A            398           LSR   A
FDDF:20 E5 FD      399           JSR   PRHEXZ
FDE2:68            400           PLA
FDE3:              401 *
FDE3:29 0F         402 PRHEX     AND   #$0F          ;PRINT HEX DIGIT IN A-REG
FDE5:09 B0         403 PRHEXZ    ORA   #$B0          ;LSBITS ONLY.
FDE7:C9 BA         404           CMP   #$BA
FDE9:90 02   FDED  405           BCC   COUT
FDEB:69 06         406           ADC   #$06
FDED:              407 *
```

```
FDED:6C 36 00      408 COUT      JMP   (CSWL)        ;VECTOR TO USER OUTPUT ROUTINE
FDF0:              409 *
FDF0:2C 7B 06      410 COUT1     BIT   VFACTV        ;video firmware active?
FDF3:4C B4 FB      411           JMP   DOCOUT1       ;mask II mode characters
FDF6:84 35         412 COUTZ     STY   YSAV1         ;SAVE Y-REG
FDF8:48            413           PHA                 ;SAVE A -REG
FDF9:20 78 FB      414           JSR   VIDWAIT       ;OUTPUT CHR AND CHECK FOR CTRL-S
FDFC:68            415           PLA                 ;RESTORE A-REG
FDFD:A4 35         416           LDY   YSAV1         ;AND Y-REG
FDFF:60            417           RTS                 ;RETURN TO SENDER...
FE00:              418 *
FE00:C6 34         419 BL1       DEC   YSAV
FE02:F0 9F   FDA3  420           BEQ   XAM8
FE04:              421 *
FE04:CA            422 BLANK     DEX                  ;BLANK TO MON
FE05:D0 16   FE1D  423           BNE   SETMDZ         ;AFTER BLANK
FE07:C9 BA         424           CMP   #$BA           ;DATA STORE MODE?
FE09:D0 BB   FDC6  425           BNE   XAMPM          ; NO; XAM, ADD, OR SUBTRACT.
FE0B:              426 *
FE0B:85 31         427 STOR      STA   MODE           ;KEEP IN STORE MODE
FE0D:A5 3E         428           LDA   A2L
FE0F:91 40         429           STA   (A3L),Y        ;STORE AS LOW BYTE AT (A3)
FE11:E6 40         430           INC   A3L
FE13:D0 02   FE17  431           BNE   RTS5           ;INCR A3, RETURN.
FE15:E6 41         432           INC   A3H
FE17:60            433 RTS5      RTS
FE18:              434 *
FE18:A4 34         435 SETMODE   LDY   YSAV           ;SAVE CONVERTED ':', '+',
FE1A:B9 FF 01      436           LDA   IN-1,Y         ; '-', '.' AS MODE
FE1D:85 31         437 SETMDZ    STA   MODE
FE1F:60            438           RTS
FE20:              439 *
FE20:A2 01         440 LT        LDX   #$01
FE22:B5 3E         441 LT2       LDA   A2L,X          ;COPY A2 (2 BYTES) TO
FE24:95 42         442           STA   A4L,X          ; A4 AND A5
FE26:95 44         443           STA   A5L,X
FE28:CA            444           DEX
FE29:10 F7   FE22  445           BPL   LT2
FE2B:60            446           RTS
FE2C:              447 *
FE2C:B1 3C         448 MOVE      LDA   (A1L),Y        ;MOVE (A1) THRU (A2) TO (A4)
FE2E:91 42         449           STA   (A4L),Y
FE30:20 B4 FC      450           JSR   NXTA4
FE33:90 F7   FE2C  451           BCC   MOVE
FE35:60            452           RTS
FE36:              453 *
FE36:B1 3C         454 VERIFY    LDA   (A1L),Y        ;VERIFY (A1) THRU (A2)
FE38:D1 42         455           CMP   (A4L),Y        ; WITH (A4)
FE3A:F0 1C   FE58  456           BEQ   VFYOK
FE3C:20 92 FD      457           JSR   PRA1
FE3F:B1 3C         458           LDA   (A1L),Y
FE41:20 DA FD      459           JSR   PRBYTE
FE44:A9 A0         460           LDA   #$A0
FE46:20 ED FD      461           JSR   COUT
FE49:A9 A8         462           LDA   #$A8
FE4B:20 ED FD      463           JSR   COUT
FE4E:B1 42         464           LDA   (A4L),Y
FE50:20 DA FD      465           JSR   PRBYTE
```

```
FE53:A9 A9          466           LDA   #$A9
FE55:20 ED FD       467           JSR   COUT
FE58:20 B4 FC       468 VFYOK     JSR   NXTA4
FE5B:90 D9   FE36   469           BCC   VERIFY
FE5D:60             470           RTS
FE5E:               471 *
FE5E:20 75 FE       472 LIST      JSR   A1PC          ;MOVE A1 (2 BYTES) TO
FE61:A9 14          473           LDA   #$14          ; PC IF SPEC'D AND
FE63:48             474 LIST2     PHA                 ; DISASSEMBLE 20 INSTRUCTIONS.
FE64:20 D0 F8       475           JSR   INSTDSP
FE67:20 53 F9       476           JSR   PCADJ         ;ADJUST PC AFTER EACH INSTRUCTION.
FE6A:85 3A          477           STA   PCL
FE6C:84 3B          478           STY   PCH
FE6E:68             479           PLA
FE6F:38             480           SEC
FE70:E9 01          481           SBC   #$01          ;NEXT OF 20 INSTRUCTIONS
FE72:D0 EF   FE63   482           BNE   LIST2
FE74:60             483           RTS
FE75:               484 *
FE75:8A             485 A1PC      TXA                 ;IF USER SPECIFIED AN ADDRESS,
FE76:F0 07   FE7F   486           BEQ   A1PCRTS       ; COPY IT FROM A1 TO PC.
FE78:B5 3C          487 A1PCLP    LDA   A1L,X         ;YEP, SO COPY IT.
FE7A:95 3A          488           STA   PCL,X
FE7C:CA             489           DEX
FE7D:10 F9   FE78   490           BPL   A1PCLP
FE7F:60             491 A1PCRTS   RTS
FE80:               492 *
FE80:A0 3F          493 SETINV    LDY   #$3F          ;SET FOR INVERSE VID
FE82:D0 02   FE86   494           BNE   SETIFLG       ; VIA COUT1
FE84:A0 FF          495 SETNORM   LDY   #$FF          ;SET FOR NORMAL VID
FE86:84 32          496 SETIFLG   STY   INVFLG
FE88:60             497           RTS
FE89:               498 *
FE89:A9 00          499 SETKBD    LDA   #$00          ;DO 'IN#0'
FE8B:85 3E          500 INPORT    STA   A2L           ;DO 'IN#AREG'
FE8D:A2 38          501 INPRT     LDX   #KSWL
FE8F:A0 1B          502           LDY   #KEYIN
FE91:D0 08   FE9B   503           BNE   IOPRT
FE93:               504 *
FE93:A9 00          505 SETVID    LDA   #$0           ;DO 'PR#0'
FE95:85 3E          506 OUTPORT   STA   A2L           ;DO 'PR#AREG'
FE97:A2 36          507 OUTPRT    LDX   #CSWL
FE99:A0 F0          508           LDY   #COUT1
FE9B:A5 3E          509 IOPRT     LDA   A2L
FE9D:29 0F          510           AND   #$0F
FE9F:D0 06   FEA7   511           BNE   NOTPRT0       ;not slot 0
FEA1:C0 1B          512           CPY   #KEYIN        ;Continue if KEYIN
FEA3:F0 39   FEDE   513           BEQ   IOPRT1
FEA5:80 1B   FEC2   514           BRA   OPRT0         ;=>do PR#0
FEA7:09 C0          515 NOTPRT0   ORA   #<IOADR
FEA9:A0 00          516           LDY   #$00
FEAB:94 00          517 IOPRT2    STY   LOC0,X
FEAD:95 01          518           STA   LOC1,X
FEAF:60             519           RTS
FEB0:               520 *
FEB0:4C 00 E0       521 XBASIC    JMP   BASIC         ;TO BASIC, COLD START
FEB3:               522 *
FEB3:4C 03 E0       523 BASCONT   JMP   BASIC2        ;TO BASIC, WARM START
```

```
FEB6:                 524 *
FEB6:20 75 FE         525 GO        JSR   A1PC           ;ADDR TO PC IF SPECIFIED
FEB9:20 3F FF         526           JSR   RESTORE        ;RESTORE FAKE REGISTERS
FEBC:6C 3A 00         527           JMP   (PCL)        ; AND GO!
FEBF:                 528 *
FEBF:4C D7 FA         529 REGZ      JMP   REGDSP         ;GO DISPLAY REGISTERS
FEC2:                 530 *
FEC2:3A               531 OPRT0     DEC   A              ;Need $FF
FEC3:8D FB 07         532           STA   CURSOR         ;set checkerboard cursor
FEC6:A9 F7            533           LDA   #$FF-M.CTL     ;reset mode
FEC8:80 04    FECE    534           BRA   DOPR0
FECA:                 535 *
FECA:4C F8 03         536 USR       JMP   USRADR         ;JUMP TO CONTROL-Y VECTOR IN RAM
FECD:                 537 *
FECD:60               538 WRITE     RTS                  ;Tape write not needed
FECE:                 539 *
FECE:8D 7B 06         540 DOPR0     STA   VFACTV         ;say video firmware inactive
FED1:8D 0E C0         541           STA   CLRALTCHAR     ;switch in normal char set
FED4:0C FB 04         542           TSB   VMODE          ;don't change M.CTL
FED7:DA               543           PHX                  ;save X and Y
FED8:5A               544           PHY                  ;for rest of PR#0
FED9:20 CD CD         545           JSR   CHK80          ;convert to 40 if needed
FEDC:7A               546           PLY
FEDD:FA               547           PLX
FEDE:A9 FD            548 IOPRT1    LDA   #<COUT1        ;set I/O page
FEE0:80 C9    FEAB    549           BRA   IOPRT2         ;=>go set output hook
FEE2:                 550 *
FEE2:                 551 * DECCH decrements the current cursor
FEE2:                 552 * CLRCH sets all cursors to 0
FEE2:                 553 * SETCUR sets cursors to value in Acc.
FEE2:                 554 * See explanatory note with GETCUR
FEE2:                 555 *
FEE2:5A               556 DECCH     PHY                  ;(from $FC10)
FEE3:20 9D CC         557           JSR   GETCUR         ;get current CH
FEE6:88               558           DEY                  ;decrement it
FEE7:80 05    FEEE    559           BRA   SETCUR1        ;go update cursors
FEE9:                 560 *
FEE9:A9 01            561 CLRCH     LDA   #1             ;set all cursors to 0
FEEB:3A               562 WDTHCH    DEC   A              ;dec window width (from $FC17)
FEEC:5A               563 SETCUR    PHY                  ;save Y
FEED:A8               564           TAY                  ;need value in Y
FEEE:20 AD CC         565 SETCUR1   JSR   GETCUR2        ;save new CH
FEF1:7A               566           PLY                  ;restore Y
FEF2:AD 7B 05         567           LDA   OURCH          ;and get new CH into acc
FEF5:60               568           RTS                  ;(Need LDA to set flags)
FEF6:                 569 *
FEF6:20 00 FE         570 CRMON     JSR   BL1            ;HANDLE CR AS BLANK
FEF9:68               571           PLA                  ; THEN POP STACK
FEFA:68               572           PLA                  ; AND RETURN TO MON
FEFB:D0 6C    FF69    573           BNE   MONZ           ;(ALWAYS)
FEFD:                 574 *
FEFD:60               575 READ      RTS                  ;Tape read not needed
FEFE:                 576 *
FEFE:                 577 * OPTBL is a table containing the new opcodes that
FEFE:                 578 * wouldn't fit into the existing lookup table.
FEFE:                 579 *
FEFE:12               580 OPTBL     DFB   $12            ;ORA (ZPAG)
FEFF:14               581           DFB   $14            ;TRB ZPAG
```

```
FF00:1A             582           DFB   $1A           ;INC A
FF01:1C             583           DFB   $1C           ;TRB ABS
FF02:32             584           DFB   $32           ;AND (ZPAG)
FF03:34             585           DFB   $34           ;BIT ZPAG,X
FF04:3A             586           DFB   $3A           ;DEC A
FF05:3C             587           DFB   $3C           ;BIT ABS,X
FF06:52             588           DFB   $52           ;EOR (ZPAG)
FF07:5A             589           DFB   $5A           ;PHY
FF08:64             590           DFB   $64           ;STZ ZPAG
FF09:72             591           DFB   $72           ;ADC (ZPAG)
FF0A:74             592           DFB   $74           ;STZ ZPAG,X
FF0B:7A             593           DFB   $7A           ;PLY
FF0C:7C             594           DFB   $7C           ;JMP (ABS,X)
FF0D:89             595           DFB   $89           ;BIT IMM
FF0E:92             596           DFB   $92           ;STA (ZPAG)
FF0F:9C             597           DFB   $9C           ;STZ ABS
FF10:9E             598           DFB   $9E           ;STZ ABS,X
FF11:B2             599           DFB   $B2           ;LDA (ZPAG)
FF12:D2             600           DFB   $D2           ;CMP (ZPAG)
FF13:F2             601           DFB   $F2           ;SBC (ZPAG)
FF14:FC             602           DFB   $FC           ;??? (the unknown opcode)
FF15:       0016    603 NUMOPS    EQU   *-OPTBL-1     ;number of bytes to check
FF15:               604 *
FF15:               605 * INDX contains pointers to the mnemonics for each of
FF15:               606 * the opcodes in OPTBL.  Pointers with BIT 7
FF15:               607 * set indicate extensions to MNEML or MNEMR.
FF15:               608 *
FF15:38             609 INDX      DFB   $38
FF16:FB             610           DFB   $FB
FF17:37             611           DFB   $37
FF18:FB             612           DFB   $FB
FF19:39             613           DFB   $39
FF1A:21             614           DFB   $21
FF1B:36             615           DFB   $36
FF1C:21             616           DFB   $21
FF1D:3A             617           DFB   $3A
FF1E:F8             618           DFB   $F8
FF1F:FA             619           DFB   $FA
FF20:3B             620           DFB   $3B
FF21:FA             621           DFB   $FA
FF22:F9             622           DFB   $F9
FF23:22             623           DFB   $22
FF24:21             624           DFB   $21
FF25:3C             625           DFB   $3C
FF26:FA             626           DFB   $FA
FF27:FA             627           DFB   $FA
FF28:3D             628           DFB   $3D
FF29:3E             629           DFB   $3E
FF2A:3F             630           DFB   $3F
FF2B:FC             631           DFB   $FC           ;???
FF2C:00             632           BRK
FF2D:               633 *
FF2D:A9 C5          634 PRERR     LDA   #$C5           ;PRINT 'ERR', THEN FALL INTO
FF2F:20 ED FD       635           JSR   COUT           ; FWEEPER.
FF32:A9 D2          636           LDA   #$D2
FF34:20 ED FD       637           JSR   COUT
FF37:20 ED FD       638           JSR   COUT
FF3A:               639 *
```

```
FF3A:A9 87          640 BELL     LDA  #$87          ;MAKE A JOYFUL NOISE, THEN RETURN.
FF3C:4C ED FD       641          JMP  COUT
FF3F:               642 *
FF3F:A5 48          643 RESTORE  LDA  STATUS        ;RESTORE 6502 REGISTER CONTENTS
FF41:48             644          PHA                ; USED BY DEBUG SOFTWARE
FF42:A5 45          645          LDA  A5H
FF44:A6 46          646 RESTR1   LDX  XREG
FF46:A4 47          647          LDY  YREG
FF48:28             648          PLP
FF49:60             649          RTS
FF4A:               650 *
FF4A:85 45          651 SAVE     STA  A5H           ;SAVE 6502 REGISTER CONTENTS
FF4C:86 46          652 SAV1     STX  XREG          ; FOR DEBUG SOFTWARE
FF4E:84 47          653          STY  YREG
FF50:08             654          PHP
FF51:68             655          PLA
FF52:85 48          656          STA  STATUS
FF54:BA             657          TSX
FF55:86 49          658          STX  SPNT
FF57:D8             659          CLD
FF58:60             660          RTS
FF59:               661 *
FF59:20 84 FE       662 OLDRST   JSR  SETNORM       ;SET SCREEN MODE
FF5C:20 2F FB       663          JSR  INIT          ; AND INIT KBD/SCREEN
FF5F:20 93 FE       664          JSR  SETVID        ; AS I/O DEVS.
FF62:20 89 FE       665          JSR  SETKBD
FF65:               666 *
FF65:D8             667 MON      CLD                ;MUST SET HEX MODE!
FF66:20 3A FF       668          JSR  BELL          ;FWEEPER.
FF69:A9 AA          669 MONZ     LDA  #$AA          ;'*' PROMPT FOR MONITOR
FF6B:85 33          670          STA  PROMPT
FF6D:20 67 FD       671          JSR  GETLNZ        ;READ A LINE OF INPUT
FF70:20 C7 FF       672          JSR  ZMODE         ;CLEAR MONITOR MODE, SCAN IDX
FF73:20 A7 FF       673 NXTITM   JSR  GETNUM        ;GET ITEM, NON-HEX
FF76:84 34          674          STY  YSAV          ; CHAR IN A-REG.
FF78:A0 13          675          LDY  #SUBTBL-CHRTBL  ; X-REG=0 IF NO HEX INPUT
FF7A:88             676 CHRSRCH  DEY
FF7B:30 E8   FF65   677          BMI  MON           ;COMMAND NOT FOUND, BEEP & TRY AGAIN.
FF7D:D9 CD FF       678          CMP  CHRTBL,Y      ;FIND COMMAND CHAR IN TABLE
FF80:D0 F8   FF7A   679          BNE  CHRSRCH       ;NOT THIS TIME
FF82:20 BE FF       680          JSR  TOSUB         ;GOT IT! CALL CORRESPONDING SUBROUTINE
FF85:A4 34          681          LDY  YSAV          ;PROCESS NEXT ENTRY ON HIS LINE
FF87:4C 73 FF       682          JMP  NXTITM
FF8A:               683 *
FF8A:A2 03          684 DIG      LDX  #$03
FF8C:0A             685          ASL  A
FF8D:0A             686          ASL  A             ;GOT HEX DIGIT,
FF8E:0A             687          ASL  A             ; SHIFT INTO A2
FF8F:0A             688          ASL  A
FF90:0A             689 NXTBIT   ASL  A
FF91:26 3E          690          ROL  A2L
FF93:26 3F          691          ROL  A2H
FF95:CA             692          DEX                ;LEAVE X=$FF IF DIG
FF96:10 F8   FF90   693          BPL  NXTBIT
FF98:A5 31          694 NXTBAS   LDA  MODE
FF9A:D0 06   FFA2   695          BNE  NXTBS2        ;IF MODE IS ZERO,
FF9C:B5 3F          696          LDA  A2H,X         ; THEN COPY A2 TO A1 AND A3
FF9E:95 3D          697          STA  A1H,X
```

```
FFA0:95 41          698             STA   A3H,X
FFA2:E8             699 NXTBS2      INX
FFA3:F0 F3    FF98  700             BEQ   NXTBAS
FFA5:D0 06    FFAD  701             BNE   NXTCHR
FFA7:A2 00          702 GETNUM      LDX   #$00          ;CLEAR A2
FFA9:86 3E          703             STX   A2L
FFAB:86 3F          704             STX   A2H
FFAD:B9 00 02       705 NXTCHR      LDA   IN,Y          ;GET CHAR
FFB0:C8             706             INY
FFB1:20 99 C3       707             JSR   UPSHIFT0      ;upshift if necessary (set high bit)
FFB4:49 B0          708             EOR   #$B0
FFB6:C9 0A          709             CMP   #$0A
FFB8:90 D0    FF8A  710             BCC   DIG           ;it's a digit
FFBA:80 37    FFF3  711             BRA   GETHEX        ;check for other digits
FFBC:00             712             BRK
FFBD:00             713             BRK
FFBE:               714 *
FFBE:A9 FE          715 TOSUB       LDA   #<GO          ;DISPATCH TO SUBROUTINE, BY
FFC0:48             716             PHA                 ; PUSHING THE HI-ORDER SUBR ADDR,
FFC1:B9 E0 FF       717             LDA   SUBTBL,Y      ; THEN THE LO-ORDER SUBR ADDR
FFC4:48             718             PHA                 ; ONTO THE STACK,
FFC5:A5 31          719             LDA   MODE          ; (CLEARING THE MODE, SAVE THE OLD
FFC7:A0 00          720 ZMODE       LDY   #$00          ; MODE IN A-REG),
FFC9:84 31          721             STY   MODE
FFCB:60             722             RTS                 ; AND 'RTS' TO THE SUBROUTINE!
FFCC:               723 *
FFCC:EA             724             NOP
FFCD:               725 *
FFCD:BC             726 CHRTBL      DFB   $BC           ;^C  (BASIC WARM START)
FFCE:B2             727             DFB   $B2           ;^Y  (USER VECTOR)
FFCF:BE             728             DFB   $BE           ;^E  (OPEN AND DISPLAY REGISTERS)
FFD0:EF             729             DFB   $EF           ;V   (MEMORY VERIFY)
FFD1:C4             730             DFB   $C4           ;^K  (IN#SLOT)
FFD2:A9             731             DFB   $A9           ;^P  (PR#SLOT)
FFD3:BB             732             DFB   $BB           ;^B  (BASIC COLD START)
FFD4:A6             733             DFB   $A6           ;'-' (SUBTRACTION)
FFD5:A4             734             DFB   $A4           ;'+' (ADDITION)
FFD6:06             735             DFB   $06           ;M   (MEMORY MOVE)
FFD7:95             736             DFB   $95           ;'<' (DELIMITER FOR MOVE, VFY)
FFD8:07             737             DFB   $07           ;N   (SET NORMAL VIDEO)
FFD9:02             738             DFB   $02           ;I   (SET INVERSE VIDEO)
FFDA:05             739             DFB   $05           ;L   (DISASSEMBLE 20 INSTRS)
FFDB:00             740             DFB   $00           ;G   (EXECUTE PROGRAM)
FFDC:93             741             DFB   $93           ;':' (MEMORY FILL)
FFDD:A7             742             DFB   $A7           ;'.' (ADDRESS DELIMITER)
FFDE:C6             743             DFB   $C6           ;'CR' (END OF INPUT)
FFDF:99             744             DFB   $99           ;BLANK
FFE0:               745 *
FFE0:               746 * Table of low order monitor routine
FFE0:               747 * dispatch addresses.
FFE0:               748 *
FFE0:B2             749 SUBTBL      DFB   >BASCONT-1
FFE1:C9             750             DFB   >USR-1
FFE2:BE             751             DFB   >REGZ-1
FFE3:35             752             DFB   >VERIFY-1
FFE4:8C             753             DFB   >INPRT-1
FFE5:96             754             DFB   >OUTPRT-1
FFE6:AF             755             DFB   >XBASIC-1
```

```
FFE7:17              756            DFB   >SETMODE-1
FFE8:17              757            DFB   >SETMODE-1
FFE9:2B              758            DFB   >MOVE-1
FFEA:1F              759            DFB   >LT-1
FFEB:83              760            DFB   >SETNORM-1
FFEC:7F              761            DFB   >SETINV-1
FFED:5D              762            DFB   >LIST-1
FFEE:B5              763            DFB   >GO-1
FFEF:17              764            DFB   >SETMODE-1
FFF0:17              765            DFB   >SETMODE-1
FFF1:F5              766            DFB   >CRMON-1
FFF2:03              767            DFB   >BLANK-1
FFF3:                768 *
FFF3:69 88           769 GETHEX     ADC   #$88
FFF5:C9 FA           770            CMP   #$FA
FFF7:B0 91    FF8A   771            BCS   DIG
FFF9:60              772            RTS
FFFA:                773 *
FFFA:FB 03           774            DW    NMI             ;NON-MASKABLE INTERRUPT VECTOR
FFFC:62 FA           775            DW    RESET           ;RESET VECTOR
FFFE:03 C8           776 IRQVECT    DW    NEWIRQ          ;INTERRUPT REQUEST VECTOR
```

```
   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          C8FF ACDONE     04FF ACIABUF
 C988 ACIADONE   C900 ACIAINT      C908 AC.ATST    FDD1 ADD
 FD84 ADDINP     FBF8 ADV2       ?FBF4 ADVANCE   C94B AIEATIT
 C943 AINOFLSH   C94D AIPASS       C922 AIPORT2    C91C AITST2
 C01E ALTCHARSET ?03F5 AMPERV      FD03 APPLE2C    FB60 APPLEII
 0438 ASTAT      C6A2 BADRD1       C6D3 BADREAD    C77C BANGER
   2B BAS2H        2A BAS2L        FBC1 BASCALC    FBD0 BASCLC2
 FEB3 BASCONT      29 BASH         E003 BASIC2     C324 BASICENT
 C79F BASICIN    C317 BASICINIT    E000 BASIC        28 BASL
 FD71 BCKSPC     FAA3 BEEPFIX    ?FBDD BELL1     FF3A BELL
 FBE4 BELL2      0215 BINH         0214 BINL       C329 BINPUT
 FE00 BL1        FE04 BLANK        FCD0 BLAST        4F BOOTDEV
 07DB BOOTSCRN     3C BOOTTMP    ?C326 BPRINT    ?FA4C BREAK
 03F0 BRKV       ?FC10 BS          04 BUTMODE      C061 BUTN0
 C062 BUTN1      CFC2 C03        C307 C3COUT1    ?C300 C3ENTRY
 C305 C3KEYIN    FD62 CANCEL       CA76 CDONE2     CA3C CDONE
?CD7D CGO        F9BA CHAR1        F9B4 CHAR2      CDCD CHK80
 FBD9 CHKBELL    C528 CHKMOU       CB4E CHKRT      FF7A CHRSRCH
   24 CH         C132 CHOK         FFCD CHRTBL     CA28 CKDIG
 FC9E CLEOLZ     FC46 CLEOP1       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
 C1DD CLRCOL     FC9C CLREOL       FC44 CLREOP2    FC42 CLREOP
 FC5D CLREOP1    CBCF CLRHALF      CD9B CLRIT      CC97 CLRKBD
 FCA0 CLRLIN     CC04 CLRPORT    ?CFFF CLRROM      F838 CLRSC2
 F83C CLRSC3   ?F832 CLRSCR        F836 CLRTOP     CA7D CMDB
 CA5D CMDCR        BF CMDCUR       CA79 CMDD       CA68 CMDI2
 CA67 CMDI       CA67 CMDK         CA14 CMDLOOP    CA67 CMDL
 C9DE CMDLIST    CA5D CMDN         CAB0 CMDP2      CA78 CMDP
 CAC4 CMDQ       CAB5 CMDR         CA99 CMDS       CAC6 CMDT
 CB05 CMDT2      CB17 CMDT3        C9C7 CMDTABLE   CA55 CMDZ
 CA25 CMDZ2      CA4D CMFOUND      C555 CMLOK      C538 CMLOOP
 C577 CMNOINT    C58E CMNOVBL      C57B CMNOY      C55D CMNT0
 C562 CMRGHT     C56F CMROK        CA43 CMSET      C542 CMXMOV
 CFB7 C01        0738 COL          FCCA COLDSTART    30 COLOR
 FCE7 COM1       FCF6 COM2         FCFC COM3       CA36 COMINIT
 C9EB COMMAND    C266 COMMPORT     C263 COMOUT     C200 COMSLOT
 CB28 COMTBL     C338 COPYROM      C348 COPYROM2   FDF6 COUTZ
 FDED COUT       FDF0 COUT1        FEF6 CRMON    ?FD8B CROUT1
 FD8E CROUT      FC62 CR           FC85 CRRTS        37 CSWH
   36 CSWL       CD2A CTLADR       CD58 CTLCHAR    CD54 CTLCHAR0
 FCA4 CTLD0      CD6F CTLDONE      CD71 CTLGO      CD80 CTLGO1
   14 CTLNUM     CD91 CTLOFF       CD95 CTLON      CD15 CTLTAB
 07FB CURSOR     C51D CVBUT        C516 CVMOVED    C4ED CVNOVBL
   25 CV         FDB6 DATAOUT      FBBC DCX        FEE2 DECCH
 C2C8 DEFAULT    C2F1 DEFCOM       C2D9 DEFFF      C2FC DEFIDX
 C2CE DEFLOOP    C6D9 DENIB1       C6D7 DENIBL     C885 DEVNO
 FF8A DIG        CA30 DIGLOOP      0356 DNIBL      CBC2 DOCLR
 FBB4 DOCOUT1    FB54 DOCTL        C6FB DODRV2     C188 DONE
 FD20 DONXTCUR   FECE DOPR0      ?C701 DRV2BOOT    C60B DRV2ENT
 C5C3 DV10LOOP   C5CA DV10LT       C111 ENTR1      C230 ENTR
 F8A1 ERR          9B ESC          CCD7 ESC0     ?CCE3 ESC1
 CCE5 ESC2       CCC0 ESC3       CD0C ESCCHAR      0638 ESCHAR
 0013 ESCNUM     CCED ESCRDKEY     CCF8 ESCTAB     C28C EXIT1
```

```
 C28A EXITX        ?C65C EXTENT       C63D EXTENT1       05F9 EXTINT2
 0538 EXTINT        F800 F8ORG        FBB3 F8VERSION     C142 FIXCH
?FA9B FIXSEV        06B8 FLAGS        CB1B FLUSH         F962 FMT1
 F9A6 FMT2         CD67 FNDCTL         2E FORMAT        C64B FUG1
?C648 FUGIT        F847 GBASCALC       27 GBASH           26 GBASL
 F856 GBCALC       C8F8 GBEMPTY      C8DB GBNOOVR       C393 GETALT1
 C398 GETALT2      C37C GETALT       C8CC GETBUF        C3A6 GETCOUT
 GCA7 GETCUR1      CC9D GETCUR       CCAD GETCUR2       CCB7 GETCUR3
 CCBF GETCURX      F8A5 GETFMT       FFF3 GETHEX        FC80 GETINDX
?FD6F GETLN1       FD67 GETLNZ      ?FD6A GETLN         FFA7 GETNUM
 CB57 GETST        CEFA GETX        ?CF06 GETY          CF38 GKEY
?FD25 GOTKEY       FEB6 GO           C89F GOBREAK         06 GOODF8
 C28F GOREMOTE     C290 GOTERM       F8CC GOTONE          2C H2
 C64E HANGING      C5E3 HDDONE       C5BE HDLOOP        C5B8 HDPOS2
?FCC9 HEADR        C5AC HEXDEC2      C59B HEXTODEC     ?C057 HIRES
 F81C HLINE1      ?F819 HLINE        FC58 HOME          CDA5 HOMECUR
 CE1B HOOKITUP     CE20 HOOKUP       F897 IEVEN         0200 INBUF
 CA0C INCMD        FF15 INDX        ?F88C INSDS2        0200 IN
 C405 INENT        FB2F INIT         C41C INITMOUSE    ?FE8B INPORT
 FE8D INPRT        F882 INSDS1       F8D0 INSTDSP       CC12 INVERT
   32 INVFLG       CC1C INVX         C000 IOADR         FE9B IOPRT
 FEDE IOPRT1       FEAB IOPRT2       FF58 IORTS         C058 IOU
 C078 IOUDSBL      C079 IOUENBL      C806 IRQ1          C827 IRQ2
 C831 IRQ3         C83B IRQ4         C850 IRQ5          C85E IRQ6
 C861 IRQ7         C873 IRQ8         C88C IRQDNE1       C88F IRQDNE2
 C899 IRQDNE3      C882 IRQDONE     ?03FE IRQLOC        C989 IRQTBLE
 FFFE IRQVECT     ?FA40 IRQ          C663 ISMRK1        C22F ISRDY
 CFF9 JMPDEST      C32C JPINIT       C32F JPREAD        C335 JPSTAT
 C332 JPWRITE      C010 KBDSTRB      FB88 KBDWAIT       C000 KBD
 FD1B KEYIN       ?FD18 KEYIN0         39 KSWH            38 KSWL
 C08B LCBANK1      C083 LCBANK2      2F LENGTH          FC66 LF
 0400 LINE1        FE5E LIST         FE63 LIST2           2C LMNEM
   00 LOC0           01 LOC1         FD38 LOOKPICK      C056 LORES
 FE22 LT2          FE20 LT          ?  40 M.40            20 M.CTL2
   08 M.CTL          10 M.CURSOR       08 M.GOXY          01 M.MOUSE
   80 M.PASCAL       04 M.VMODE        44 MACSTAT       C709 MAKTBL
   2E MASK         C9D4 MASK1        C9D9 MASK2         05F8 MAXH
 04F8 MAXL         077D MAXXH        067D MAXXL        ?07FD MAXYH
?06FD MAXYL        C400 MBASIC       C79B MBBAD         0578 MINH
 0478 MINL         057D MINXH        047D MINXL        ?05FD MINYH
?04FD MINYL        C8AB MIRQLP       C8C2 MIRQSTD       C4F1 MISTAT
?C052 MIXCLR       C053 MIXSET       F9C0 MNEML         FA00 MNEMR
 F8BE MNNDX1       F8C2 MNNDX2       F8C9 MNNDX3        FDAD MOD8CHK
   31 MODE         FF65 MON          FF69 MONZ          067C MOUARM
 C063 MOUBUT       C048 MOUCLR      ?C058 MOUDSBL      ?C059 MOUENBL
 07FC MOUMODE      C4D5 MOUSEINT     CD9F MOUSOFF       CD99 MOUSON
 077C MOUSTAT      0478 MOUTEMP      C066 MOUX1         057C MOUXH
 C015 MOUXINT      047C MOUXL        C067 MOUY1         05FC MOUYH
 C017 MOUYINT      04FC MOUYL          20 MOVARM        CF86 MOVEAUX
 CF99 MOVEC2M      C8A2 MOVEIRQ      CF9F MOVELOOP      FE2C MOVE
 CFCB MOVERET      CF9F MOVESTRT       02 MOVMODE       C7DE MPADDLE
 C72F MSG          CAA6 MSLOOP       07F8 MSLOT         CAA4 MSWAIT
 0300 NBUF1        FBB0 NEWADV1      FBA0 NEWADV        FA47 NEWBRK
 FC99 NEWC1        FC90 NEWCLEOLZ    FC8D NEWCLREOL     FC73 NEWCR
 CCCC NEWESC       C803 NEWIRQ      ?FA81 NEWMON        FC38 NEWOP1
 FC35 NEWOPS       FC86 NEWVTAB      FC88 NEWVTABZ      CFA9 NEXTA1
 03FB NMI          CA09 NOCMD        C46B NOERROR      ?FD45 NOESC1
 FD4A NOESC2       FD44 NOESCAPE     C26B NOESC         FACF NOFIX
```

```
 C725 NOPATRN      C371 NOREAD       CA93 NOSHIFT      C4F9 NOSTAT2
 C36A NOT1         C8FE NOTACIA      FD5F NOTCR1       FD4D NOTCR
?CC68 NOTINV1      CC6B NOTINV2      CC53 NOTINV       FEA7 NOTPRT0
 C22E NOTRDY       FB94 NOWAIT       047F NUMBER       0016 NUMOPS
 FCBA NXTA1        FCB4 NXTA4        FF98 NXTBAS       FF90 NXTBIT
 FFA2 NXTBS2       FD75 NXTCHAR      FFAD NXTCHR      ?F85F NXTCOL
 077B NXTCUR       FF73 NXTITM       FA59 OLDBRK       047B OLDCH
 067A OLDCUR2      0679 OLDCUR      ?FF59 OLDRST       FEC2 OPRT0
 FEFE OPTBL        057B OURCH        05FB OURCV        C407 OUTENT
?FE95 OUTPORT      FE97 OUTPRT       C1E4 P1INIT       C1F3 P1READ2
 C1EE P1READ       C1FB P1STATUS     C1F6 P1WRITE      C211 P2INIT
 C213 P2READ       C217 P2STATUS     C215 P2WRITE      C064 PADDL0
 CF71 PASCALC     ?CF7F PASCLC2      CC0B PASINVERT    CF35 PASREAD
 C850 PASSKIP1     C97C PBFULL       C973 PBOK        ?F954 PCADJ2
 F95C PCADJ4       F953 PCADJ        F956 PCADJ3         3B PCH
   3A PCL          CF19 PCTL         C7F6 PDOK         C7EB PDON
 CC3D PICK1        CC33 PICK2        CC3F PICK3        CC4A PICK4
   95 PICK         CC1D PICKY        CF41 PINIT        CEBC PIORDY
 F800 PLOT         F80E PLOT1        CEC0 PNOTRDY      C402 PNULL
 FD92 PRA1         F910 PRADR1       F914 PRADR2       F926 PRADR3
 F92A PRADR4       F930 PRADR5       F94A PRBL2       ?F94C PRBL3
 F948 PRBLNK       FDDA PRBYTE      ?FB1E PREAD        FB25 PREAD2
?FF2D PRERR        CEF7 PRET        ?FDE3 PRHEX        FDE5 PRHEXZ
 F8F5 PRMN1        F8F9 PRMN2        C168 PRNOW       ?F941 PRNTAX
 F8DB PRNTBL       F8D4 PRNTOP       F940 PRNTYX       C14C PRNT
?F944 PRNTX          33 PROMPT       FD96 PRYX2        CF66 PS1
 CF51 PSETUP       CF54 PSETUP2      CF30 PSETX        C222 PSTAT2
 CEB1 PSTATUS      CEBE PSTERR      ?C070 PTRIG        C967 PUTBUF
 C7DA PUTINBUF     CE3B PVMODE       04B8 PWDTH        CEDD PWR1
 FAFD PWRCON       03F4 PWREDUP      CEF4 PWRET        CEC2 PWRITE
 CEF1 PWRITERET    FB12 PWRUP2       FAA6 PWRUP        C506 QLOOP
 C5E8 QTBL         CE45 QUIT         CE44 QX          ?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
?F938 RELADR       FA62 RESET        FABD RESET.X      C354 RESETLC
 FF3F RESTORE     ?FF44 RESTR1       C641 RETRY1       C657 RETRY
 FADA RGDSP1       FB02 RGDSP2         2D RMNEM          4F RNDH
   4E RNDL         C081 ROMIN        C37B ROMOK        0478 ROMSTATE
 FAD2 RTBL         F80C RTMASK       F87F RTMSKZ       F831 RTS1
 FBEF RTS2B        F961 RTS2         FB2E RTS2D        FBFC RTS3
 FCC8 RTS4B       ?FDC5 RTS4C       ?FC34 RTS4         FE17 RTS5
?FCB3 RTS6        ?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         CBB9 SCRL3        CB9B SCRLEVEN
 CBA2 SCRLFT       CB6D SCRLIN       CBB0 SCRLODD     ?F871 SCRN
 F879 SCRN2        CE80 SCRN48       CE53 SCRN84       CB30 SCROLLDN
 CB38 SCROLLIT     CB35 SCROLLUP    ?FC70 SCROLL       BFF8 SDATA
 C61F SEEKZERO     C296 SERIN        C11E SERISOUT     03B8 SERMODE
 C18C SEROUT       C191 SEROUT2      C19D SEROUT3      C117 SERPORT
 C100 SERSLOT      C146 SERVID       CDC0 SET40        C001 SET80COL
```

```
 CDBE SET80          C00D SET80VID       C00F SETALTCHAR     C009 SETALTZP
?C059 SETAN0        ?C05B SETAN1        ?C05D SETAN2         C05F SETAN3
 C184 SETCH         ?F864 SETCOL         FEEE SETCUR1        FEEC SETCUR
 CB67 SETDBAS       ?FB40 SETGR          CE23 SETHOOKS       FE86 SETIFLG
 FE80 SETINV        ?C454 SETIOU         CDA1 SETIT          FE89 SETKBD
 FE1D SETMDZ         FE18 SETMODE        FE84 SETNORM       ?FAA9 SETPG3
 FAAB SETPLP        ?FB6F SETPWRC        C360 SETROM         CB88 SETSRC
 C008 SETSTDZP       CACD SETTERM       ?FB39 SETTXT         C233 SETUP
 CB83 SETUP2         FE93 SETVID         FB4B SETWND         CE1A SETX
 CBC1 SEV1           CC4C SHOWCUR        C45E SILOOP         C2BE SINOMOD
 C205 SIN            C465 SINOCH         CBA8 SKPLFT         CBB4 SKPRT
   2B SLOTZ            C1 SLTDMY         C46C SMINVALID      C18B SOCMD
 C1E2 SODONE        03F2 SOFTEV          C1BF SOMAIN         C1A9 SORDY
 C1C1 SORDY2         C207 SOUT           C030 SPKR             49 SPNT
 BFF9 SSTAT          CF29 STARTXY          48 STATUS         CAE6 STCLR
 FB65 STITLE        ?FE0B STOR           FBF0 STORADV        C3B8 STORCH
 C3DB STORE1        ?C3F7 STORE4         C3C1 STORE          C3EE STORE2
?C3F2 STORE3         C3F9 STORE5         C3B3 STORY          CB21 STRTS
 CAED STSET          CAF6 STWASOK        FFE0 SUBTBL         C246 SUDODEF
 C25C SUDONE         C249 SUNODEF        C257 SUOUT          C160 TAB
?FB5B TABV          ?C020 TAPEOUT        C740 TBL1           C749 TBL2
 C71B TBLLOOP2       C70D TBLLOOP        04F8 TEMP1          06F8 TEMP
 0578 TEMPA          05F8 TEMPY          C293 TERM1            DF TERMCUR
 C275 TESTKBD        0800 THBUF         ?FB09 TITLE          C15E TOOFAR
 FFBE TOSUB          06FF TRKEY          067F TRSER          05FF TWKEY
 057F TWSER          C050 TXTCLR         C054 TXTPAGE1       C055 TXTPAGE2
 C051 TXTSET         05FA TYPHED         CC91 UD2            C39B UPSHIFT
 FC1A UP             CC70 UPDATE         C399 UPSHIFT0       03F8 USRADR
 FECA USR              2D V2             C070 VBLCLR         C019 VBLINT
   0C VBLMODE        FE36 VERIFY         067B VFACTV         FE58 VFYOK
 CE31 VIDMODE        FC04 VIDOUT1        FBFD VIDOUT         FB78 VIDWAIT
 F826 VLINEZ         F828 VLINE          04FB VMODE          FC30 VTAB40
 FC22 VTAB           FB59 VTAB23         FC24 VTABZ          FCA8 WAIT
 FCA9 WAIT2          FCAA WAIT3          FEEB WDTHCH         CDD5 WIN0
 CDE0 WIN1           CDED WIN2           CDF2 WIN3           CE02 WIN4
 CDD2 WIN40          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       CDB7 X.SI           CDB0 X.SO           C3A5 X.UPSHIFT
 FDB3 XAM            FDA3 XAM8           FDC6 XAMPM          FEB0 XBASIC
 C9AD XBITKBD        C9C0 XBKB1          C9C2 XBKB2          06FB XCOORD
 CFF6 XFERAZP        CFE0 XFERC2M        CFE6 XFERZP         CFCD XFER
 C752 XLOOP1         C780 XMBASIC        C78D XMBOUT         C490 XMCDONE
 C4B0 XMCLAMP        C484 XMCLEAR        C59A XMDONE         C473 XMH2
 C471 XMHLOOP        C46D XMHOME         C4CF XMINT          C763 XMODE
 C495 XMREAD         C50C XMSKIP         C4FC XMTSTINT       C9A8 XNOKEY
 C8F0 XNOSBUF          91 XON            C758 XPAGE          C4A4 XRBUT
 C4AB XRBUT2         C98F XRDKBD         C8C5 XRDSER         C8FF XRDSNO
   46 XREG           C9A0 XRKBD1         C423 XRLOOP         C766 XRSET
 C76E XRST1          C43D XSETMOU        C452 XSOFF         ?C100 XXX
 0008 YHI              47 YREG             35 YSAV1            34 YSAV
 FFC7 ZMODE          C7FB ZZNM1          CB24 ZZNM2          CE4D ZZQUIT
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 15-JAN-84 21:28
** TOTAL LINES ASSEMBLED  4406
** FREE SPACE PAGE COUNT   47
```

Glossary

65C02: The microprocessor used in the Apple IIc computer.

ACIA: Asynchronous Communications Interface Adapter. A single chip that converts data from parallel to serial form, and vice versa, and handles serial transmission and reception and RS-232-C signals, under the control of its internal registers set and changed by firmware or software.

accumulator: The register in the 6502 and 65C02 microprocessors where most computations are performed.

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 used to identify something, such as a location in the computer's memory.

analog: Represented in terms of a physical quantity that can vary smoothly and continuously over a range of values. For example, a conventional 12-hour clock face is an analog device that represents the time of day in terms of the angles of the clock's hands. Compare **digital**.

analog-to-digital converter: A device that converts quantities from analog to digital form. For example, the Apple IIc's hand control converts the position of the control dial (an analog quantity) into a discrete number (a digital quantity) that changes in steps even when the dial is turned smoothly.

AND: A logical operator that produces a true result if both of its operands are true, a false result if either or both of its operands are false; compare **OR, exclusive OR, NOT**.

Apple IIc: A personal computer in the Apple II family, manufactured and sold by Apple Computer, Inc.

Applesoft: An extended version of the BASIC programming language used with the Apple IIc computer. The firmware for interpreting and executing programs in Applesoft is included in the Apple IIc ROM.

ASCII: American Standard Code for Information Interchange; a code in which the numbers from 0 to 127 stand for text characters, used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

assembler: A language translator that converts a program written in assembly language into an equivalent program in machine language.

assembly language: A low-level programming language in which individual machine-language instructions are written in a symbolic form more easily understood by a human programmer than machine language itself.

asserted: Made true (positive in positive-true logic; negative in negative-true logic).

asynchronous: Having a variable time interval between characters.

back panel: The rear face of the Apple IIc computer, which includes the power switch, the power connector, and connectors for two serial devices, a video display device, an external disk drive, and a mouse or hand control.

bandwidth: A measure of the range of frequencies a device can handle. In the case of a video monitor, greater bandwidth enables it to display more information; 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.

baud: A unit of signaling speed equal to the number of discrete conditions or signal events per second. Often equated (though not precisely) with bits per second.

binary: The representation of numbers in terms of powers of two, using the two digits 0 and 1. Commonly used in computers, since the values 0 and 1 can easily be represented in physical form 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.

bit: A binary digit (0 or 1); the smallest possible unit of information, consisting of a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing.

board: See **printed-circuit board**.

boot: To start up a computer by loading a program into memory from an external storage medium such as a disk. Often accomplished by first loading a small program whose purpose is to read the larger program into memory. The program is said to *pull itself up by its own bootstraps*.

bootstrap: See **boot**.

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 timesharing service.

BRK: A 65C02 instruction that causes the microprocessor to halt.

buffer: An area of the computer's memory used as a *holding area* where information can be stored by one program or device and then read out by another at a different speed.

bus: A group of wires that transmit related information from one part of a computer system to another. In the Apple IIc, the address bus has 16 wires, and the data bus has eight.

byte: A unit of information consisting of a fixed number of bits; on the Apple IIc, one byte consists of eight bits and can represent any value between 0 and 255.

carriage return: An ASCII character (decimal 13; Appendix H) that ordinarily causes a printer or display device to place the subsequent character on the left margin. On a manual typewriter, this movement is combined with line feed (the advancement of the paper to the next line). With computers, carriage return and line feed are separate, causing hair-raising problems for the user.

carrier: The background signal on a communication channel that is modified to *carry* the 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: The C bit in the 65C02 processor status register, used to hold the *carry bit* in addition and subtraction.

cathode-ray tube: An electronic device, such as a television picture tube, that produces images on a screen coated with phosphors that emit light when struck by a focused beam of electrons.

central processing unit: See **processor**.

character: A letter, digit, punctuation mark, or other symbol used in printing, displaying or transferring information.

character code: A number used to represent a text character for processing by a computer system.

chip: The small piece of semiconducting material (usually silicon) on which an integrated circuit is fabricated.

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.

code: (1) A number or symbol used to represent some piece of information in a compact or easily processed form. (2) The statements or instructions making up a program.

cold start: The process of starting up the Apple IIc 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, then loading and running a program. Compare **warm start**.

command: A communication from the user to a computer system (usually typed from the keyboard) directing it to perform some action.

command character: An ASCII character, usually CONTROL-A or CONTROL-I, that causes the serial port firmware to interpret subsequent characters as a command.

command register An ACIA location (at address $C09A for port 1 and $C0AA for port 2) that stores parity type and RS-232-C signal characteristics.

communication mode: An operating state in which serial port 2 (or 1, if so set) is prepared to exchange data and signals with a DCE (such as a modem).

compiler: A language translator that converts a program written in a high-level programming language into an equivalent program in some lower-level language (such as machine language) 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.

computer: An electronic device for performing predefined (programmed) computations at high speed and with great accuracy.

computer system: A computer and its associated hardware, firmware, and software.

connector: A physical device such as a plug, socket, or jack, used to connect two devices to one another.

control character: A character that controls or modifies the way information is printed or displayed. Control characters have ASCII codes between $00 and $1F (or between $80 and $9F if the high-order bit is set). You can generate them at the Apple IIc keyboard by holding down CONTROL while typing one of the letter keys or @ [\ ] ^ or _.

control register: An ACIA location (at address $C09B for port 1, or $C0AB for port 2) that stores data format and baud rate selections.

CPU: Central processing unit; see **processor**.

CRT: See **cathode-ray tube**.

cursor: A symbol displayed on the screen that marks 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.

data bit: One of five to eight bits representing a character.

Data Carrier Detect: An RS-232-C signal from a DCE (such as a modem) to a DTE (such as an Apple IIc) indicating that a communication connection has been established.

Data Communication Equipment: As defined by the RS-232-C standard, any device that transmits or receives information. Usually this is a modem. However, when a Modem Eliminator is used, the Apple IIc itself looks like a DCE to the other device, and the other device looks like a DCE to the Apple IIc.

data format: The form in which data is stored, manipulated or transferred. Serial data transmitted and received by port 1 or 2 has a data format of: one start bit, five to eight data bits, an optional parity bit, and one, one and a half, or two stop bits.

Data Set Ready: An RS-232-C signal from a DCE to a DTE indicating that the DCE has established a connection.

Data Terminal Equipment: As defined by the RS-232-C standard, any device that generates or absorbs information, thus acting as a terminus of a communication connection.

Data Terminal Ready: An RS-232-C signal from a DTE to a DCE indicating a readiness to transmit or receive data.

DCD: See **Data Carrier Detect**.

DCE: See **Data Communication Equipment**.

debug: To locate and correct an error or the cause of a problem or malfunction in a computer system. Typically used to refer to software-related problems.

decimal: The common form of number representation used in everyday life, in which numbers are expressed in terms of powers of ten, using the ten digits 0 to 9.

default: A value, action, or setting that is assumed or set in the absence of explicit instructions otherwise.

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 sound emitted by a speaker.

device: (1) A physical apparatus for performing a particular task or achieving a particular purpose. (2) In particular, a hardware component of a computer system.

digit: (1) One of the characters 0 to 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 to 9 and A to F in hexadecimal.

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

digital-to-analog converter: A device that converts quantities from digital to analog form.

DIP: See **dual in-line package**.

disassembler: A language translator that converts a machine-language program into an equivalent program in assembly language, more easily understood by a human programmer. 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, similarly to the way sounds are recorded on tape.

disk drive: A device that writes and reads information on the surface of a magnetic disk.

diskette: A term sometimes used for the small (5-1/4-inch) flexible disks used with the Apple Disk II drive.

Disk II drive: A model of disk drive made and sold by Apple Computer for use with the Apple IIe computer; uses 5-1/4-inch flexible (*floppy*) disks.

Disk Operating System: An optional software system for the Apple IIe that enables the computer to control and communicate with one or more Disk II drives.

display: (1) Information exhibited visually, especially on the screen of a display device. (2) To exhibit information visually. (3) A display device.

display device: A device that exhibits information visually, such as a television receiver or video monitor.

display screen: The glass or plastic panel on the front of a display device, on which images are displayed.

DOS: See **Disk Operating System**.

DSR: See **Data Set Ready**.

DTE: See **Data Terminal Equipment**.

DTR: See **Data Terminal Ready**.

dual in-line package: An integrated circuit packaged in a narrow rectangular box with a row of metal pins along each side; similar in appearance to an armored centipede.

echo: To send an input character to a video display, printer, or other output device.

edit: To change or modify; for example, to insert, remove, replace, or move text in a document.

editor: A program that enables the user to create and edit information of a particular form; for example, a *text editor* or a *graphics editor*.

effective address: In machine-language programming, the address of the memory location on which a particular instruction actually operates, which may be arrived at by indexed addressing or some other addressing method.

emulation mode: A manner of operating in which one computer or interface imitates another.

even parity: Use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits (among the data bits plus the parity bit) an even number.

error message: A message displayed or printed to notify the user of an error or problem in the execution of a program.

escape mode: A state of the Apple IIe computer, entered by pressing the ESC key, in which certain keys on the keyboard 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 ESC, used for positioning the cursor and controlling the display of text on the screen.

exclusive OR: A logical operator that produces a true result if one of its operands is true and the other false, a false result if its operands are both true or both false; compare **OR, AND, NOT**.

execute: To perform or carry out a specified action or sequence of actions, such as those described by a program.

firmware: Software stored permanently in hardware: programs in read-only memory (ROM). Such programs (for example, the Applesoft interpreter and the Apple IIc 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**.

flexible disk: A disk made of flexible plastic; often called a *floppy* disk. Compare **rigid disk**.

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**.

form feed: An ASCII character (decimal 12; Appendix H) that causes a printer or other paper-handling device to advance to the top of the next page.

framing error: In serial data transfer, absence of the expected stop bit(s) at the end of a received character. The serial port 1 and 2 ACIAs record this error by setting bit 1 (FRM) of its status register to 1. The ACIA checks and records each framing error separately: if the next character is OK, the FRM bit is cleared.

full duplex: Capable of simultaneous two-way communication.

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: Capable of communication in one direction at a time.

hand control: An optional peripheral device that can be connected to the Apple IIc's hand control connector and has a rotating dial and a pushbutton; typically used to control game-playing programs, but can be used in more serious applications as well.

hand control connector: A 9-pin connector on the Apple IIc's back panel, used for connecting hand controls to the computer.

hardware: Those components of a computer system consisting of physical (electronic or mechanical) devices. Compare **software, firmware**.

hertz: The unit of frequency of vibration or oscillation, also called cycles per second; named for the physicist Heinrich Hertz and abbreviated Hz. The Apple IIc's 65C02 microprocessor operates at a clock frequency of 1 million hertz, or 1 megahertz (MHz).

hexadecimal: The representation of numbers in terms of powers of sixteen, using the sixteen digits 0 to 9 and A to F. Hexadecimal numbers are easier for humans to read and understand than binary numbers, but can be converted easily and directly to binary form: each hexadecimal digit corresponds to a sequence of four binary digits, or bits.

high-level language: A programming language that is relatively easy for humans to understand. A single statement in a high-level language typically corresponds to several instructions of machine language.

high-order byte: The more significant half of a memory address or other two-byte quantity. In the Apple IIc's 65C02 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

high-resolution graphics: The display of graphics on the Apple IIc's display screen as a six-color array of points, 280 columns wide and 192 rows high.

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**.

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 of specifying memory addresses used in machine-language programming.

index register: A register in a computer processor that holds an index for use in indexed addressing. The Apple IIc's 65C02 microprocessor has two index registers, called the X register and the Y register.

input: (1) Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem. (2) The act or process of transferring such information.

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, with no fractional part; represented inside the computer in fixed-point form.

integrated circuit: An electronic component consisting of many circuit elements fabricated on a single piece of semiconducting material, such as silicon; see **chip**.

interface: The devices, rules, or conventions by which one component of a system communicates with another.

interpreter: A language translator that reads a program written in a particular programming language and immediately carries out the actions that the program describes. Compare **compiler**.

interrupt: A temporary suspension in the execution of a program by a computer in order 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 black dots on a white (or other single phosphor color) background, instead of the usual white dots on a black background.

I/O: Input/output; the transfer of information into and out of a computer. 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 Apple IIc Monitor program.

K: Two to the tenth power, or 1024 (from the Greek root *kilo*, meaning one thousand); for example, 64K equals 64 times 1024, or 65,536.

keyboard: The set of keys built into the Apple IIc computer, similar to a typewriter keyboard, for typing information to the computer.

keystroke: The act of pressing a single key or a combination of keys (such as CONTROL-C) on the Apple IIc keyboard.

kilobyte: A unit of information consisting of 1K (1024) bytes, or 8K (8192) bits; see **K**.

KSW: The symbolic name of the location in the Apple IIc's memory where the standard input link is stored; stands for *keyboard switch*. See **I/O link**.

language: See **programming language**.

language translator: A system program that reads a 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**.

least significant bit: The right-hand bit of a binary number as written down; its positional value is 0 or 1.

line feed: An ASCII character (decimal 10; Appendix H) that ordinarily causes a printer or video display to advance to the next line.

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: Nearby; capable of direct connection using wires only.

location: See **memory location**.

logical operator: An operator, such as **AND**, that combines logical values to produce a logical result.

low-level language: A programming language that is relatively close to the form that the computer's processor can execute directly. Low-level languages available for the Apple IIc include 65C02 machine language and 65C02 assembly language.

low-order byte: The less significant half of a memory address or other two-byte quantity. In the Apple IIc's 65C02 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

low-power Schottky: A type of TTL integrated circuit having lower power and higher speed than a conventional TTL integrated circuit.

low-resolution graphics: The display of graphics on the Apple IIc's display screen as a sixteen-color array of blocks, 40 columns wide and 48 rows high.

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 65C02 microprocessor used in the Apple IIc) has its own form of machine language.

main memory: The memory component of a computer system that is built into the computer itself and whose contents are directly accessible to the processor.

MARK parity: A bit of value 1 appended to a binary number for transmission. The receiving device can then check for errors by looking for this value on each character.

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 IIc, a memory location holds one byte, or eight bits, of information.

MHz: Megahertz; one million hertz. See **hertz**.

microcomputer: A computer, such as the Apple IIc, whose processor is a microprocessor.

microprocessor: A computer processor contained in a single integrated circuit, such as the 65C02 microprocessor used in the Apple IIc.

microsecond: One millionth of a second; abbreviated us.

millisecond: One thousandth of a second; abbreviated ms.

mode: A state of a computer or system that determines its behavior.

modem: Modulator/demodulator; a peripheral device that enables the computer to transmit and receive information over a telephone line; a DCE that connects a DTE to communication lines.

modem eliminator: The physical crossing of wires that replaces a pair of modems for direct connection of two DTEs.

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 Apple IIc in firmware, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level.

most significant bit: The leftmost bit of a binary number as written down. This bit represents 0 or 1 times 2 to the power one less than the total number of bits in the binary number. For example, in the binary number 10000, which contains five digits, the *1* represents 1 times two to the fourth power—or sixteen.

nanosecond: One billionth (in British usage, one thousand-millionth) of a second; abbreviated ns.

network: A collection of interconnected, individually controlled computers, together with the hardware and software used to connect them.

nibble: A unit of information equal to half a byte, or four bits; can hold any value from 0 to 15. Sometimes spelled *nybble*.

NOT: A unary logical operator that produces a true result if its operand is false, a false result if its operand is true; compare **AND, OR, exclusive OR**.

NTSC: (1) 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: Use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an odd number.

opcode: See **operation code**.

operand: A value to which an operator is applied; the value on which an opcode operates.

operating system: A software system that organizes the computer's resources and capabilities and makes them available to the user or to application programs running on the computer.

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.

OR: A logical operator that produces a true result if either or both of its operands are true, 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.

overrun: A condition that occurs when the Apple IIc processor does not retrieve a received character from the ACIAs receive data register 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, consisting on the Apple IIc 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 bytes.

page zero: See **zero page**.

parallel interface: An interface in which many bits of information (typically eight bits, or one byte) are transmitted simultaneously over different wires or channels. Compare **serial interface**.

parity: Maintenance of a sameness of level or count, usually the count of 1 bits in each character, for error checking.

parity error: Absence of the correct parity bit value in a received character. The serial port ACIAs record this error by setting bit 0 (PAR) of their status registers to 1.

PC board: See **printed-circuit board**.

phase: (1) A stage in a periodic process; a point in a cycle; for example, the 65C02 microprocessor uses a clock cycle consisting of two phases called PHI0 and PHI1. (2) The relationship between two periodic signals or processes; for example, in NTSC color video, the color of a point on the screen is expressed by the instantaneous phase of the video signal relative to the color reference signal.

pipelining: A feature of a processor that enables it to begin fetching the next instruction before it has finished executing the current instruction. All other things equal, processors that have this feature run faster than those without it.

pointer: An item of information consisting of the memory address of some other item.

pop: To remove the top entry from a stack.

port: The point of connection, usually a physical connector, between a computer and a peripheral device, another computer, or a network.

power supply: The hardware component of a computer that draws electrical power from a power outlet and converts it to the forms needed by some other hardware component.

printed-circuit board: A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly fiberglass, from which all conducting material except the desired circuits is etched, and to which integrated circuits and other electronic components are connected.

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.

program: (1) 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. (2) To write a program.

programming language: A set of rules or conventions for writing programs.

prompt: To remind or signal the user that some action is expected, typically by displaying a distinctive symbol, a reminder message, or a menu of choices on the display screen.

prompt character: A text character displayed on the screen to prompt the user for some action. Often also identifies the program or component of the system that is doing the prompting; for example, the prompt character] is used by the Applesoft BASIC interpreter, > by Integer BASIC, and * by the system Monitor program.

prompt message: A message displayed on the screen to prompt the user for some action.

protocol: A predefined exchange of control signals between devices enabling them to prepare for and carry out coordinated data transfers.

push: To add an entry to the top of a stack.

radio-frequency modulator: A device for converting the video signals produced by a computer to a form that can be accepted by a television receiver.

RAM: See **random-access memory**.

random-access memory: Memory in which the contents of individual locations can be referred to in an arbitrary or random order.

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 dots on the individual lines of the raster.

read: To transfer information into the computer's memory from a source external to 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: Memory whose contents can be read but not written; used for storing firmware. Information is written into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off, and can never be erased or changed. Compare **read-write memory, random-access memory, write-only memory**.

read-write memory: Memory whose contents can be both read and written; often misleadingly called *random-access memory*, or *RAM*. 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 more permanent storage medium, such as a disk. Compare **read-only memory, random-access memory, write-only memory**.

receive data register: A read-only register in each serial port ACIA (at location $C098 for port 1 and $C0A8 for port 2) that stores the most recent character successfully received.

register: A location in a computer processor where an item of information, such as a byte, is held and modified under program control. Registers in the 65C02 microprocessor include the accumulator (A), two index registers (X and Y), the stack pointer (S), the processor status register (P), and the program counter (PC). The PC register holds two bytes (sixteen bits); the other registers hold one byte (eight bits) each.

remote: Too distant for direct connection using wires or cables only.

Request To Send: An RS-232-C signal from a DTE to a DCE to prepare the DCE for data transmission.

return address: The point in a program to which control returns on completion of a subroutine.

RF modulator: See **radio-frequency modulator**.

RI: See **Ring Indicator**.

rigid disk: A disk made of a hard, nonflexible material. Compare **flexible disk**.

Ring Indicator: An optional RS-232-C signal from a DCE to a DTE that indicates the arrival of a call.

ROM: See **read-only memory**.

routine: A part of a program that accomplishes some task subordinate to the overall task of the program.

RS-232-C: A standard created by the Electronic Industries Association (EIA) to allow devices of different manufacturers to exchange serial data—particularly via telephone lines.

RTS: See **Request To Send**.

run: (1) To execute a program. (2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

save: To transfer information from main memory to a peripheral storage medium for later use.

screen: See **display screen**.

scroll: To change the contents of all or part of the display screen by shifting information out at one end (most often the top) to make room for new information appearing at the other end (most often the bottom), producing an effect like that of moving a scroll of paper past a fixed viewing window. See **viewport, window**.

serial interface: An interface in which information is transmitted sequentially, one 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**.

silicon: A non-metallic, semiconducting chemical element from which integrated circuits are made.

soft switch: A means of changing some feature of the Apple IIc from within a program; specifically, a location in memory that produces some special effect whenever its contents are read or written.

software: Those components of a computer system consisting of programs that determine or control the behavior of the computer. Compare **hardware, firmware**.

source code: See **source program**.

source program: The original 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 of value 0 appended to a binary number for transmission. The receiving device can look for this value on each character as a means of error checking.

stack: A list in which entries are added or removed at one end only (the top of the stack), causing them to be removed in LIFO (last-in-first-out) order.

start bit: A transition from a MARK signal to a SPACE signal for one bit-time, indicating that the next string of bits represents a character.

status register: A register in an ACIA (at location $C099 for port 1 and $C0A9 for port 2) that stores the state of two of the 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.

stop bit: A MARK signal following a string of data bits (or their optional parity bit) to indicate the end of a character.

string: An item of information consisting of a sequence of text characters.

strobe: (1) An event, such as a change in a signal, that triggers some action. (2) A signal whose change is used to trigger some action.

subroutine: A part of a program that can be executed on request from any point in the program, and which returns control to the point of the request on completion.

television receiver: A display device capable of receiving broadcast video signals (such as commercial television) by means of an antenna. Can be used in combination with a radio-frequency modulator as a display device for the Apple IIc computer. Compare **video monitor**.

television set: See **television receiver**.

terminal: A device consisting of a typewriterlike keyboard and a display device, used for communicating between a computer system and a human user. Personal computers such as the Apple IIc typically have all or part of a terminal built into them.

terminal mode: An operating state of the Apple IIc communication port in which the firmware makes the computer act like a simple ASCII terminal.

text: (1) Information presented in the form of characters readable by humans. (2) The display of characters on the Apple IIc's display screen. Compare **graphics**.

text window: An area on the Apple IIc's display screen within which text is displayed and scrolled.

transistor-to-transistor logic: (1) A family of integrated circuits used in computers and related devices. (2) A standard for interconnecting such circuits that defines the voltages used to represent logical zeros and ones.

transmit data register: A write-only register in one of the serial port ACIAs (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 in a computer system. Typically used to refer to hardware-related problems; compare **debug**.

TTL: See **transistor-to-transistor logic**.

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.

user: The person operating or controlling a computer system.

user interface: The rules and conventions by which a computer system communicates with the person operating it.

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 capable of receiving video signals by direct connection only, and which cannot receive broadcast signals such as commercial television. Can be connected directly to the Apple IIc computer as a display device. Compare **television receiver**.

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) that the program is working on. Compare **window**.

warm start: The process of restarting the Apple IIc after the power is already on, without reloading the operating system into main memory and often without losing the program or information already in main memory. 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 of a fixed size that is treated as a unit; the number of bits in a word is a characteristic of each particular computer.

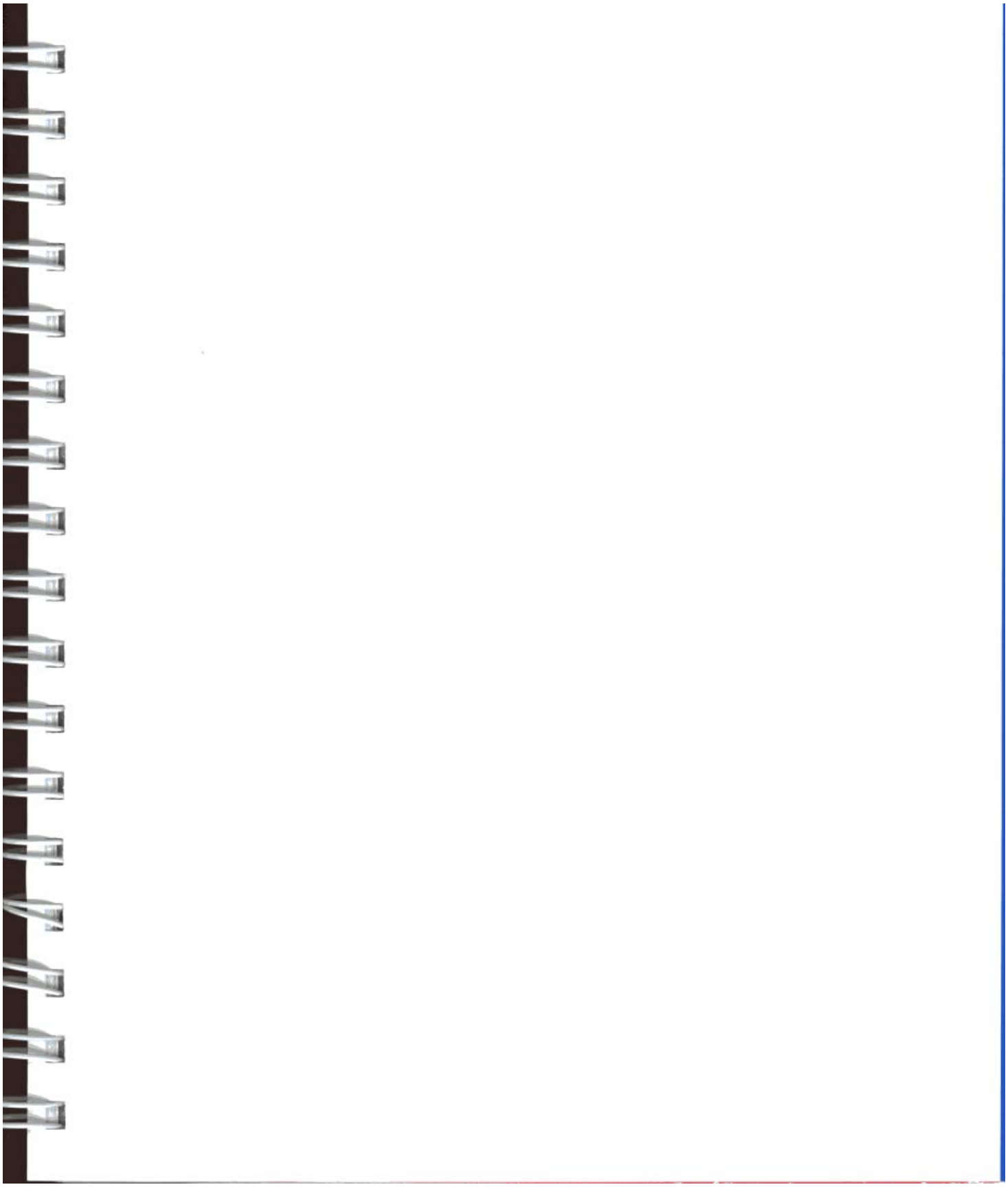
wraparound: The automatic continuation of text from the end of one line to the beginning of the next, as on the display screen or a printer.

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).

X register: One of the index registers in the 65C02 microprocessor.

Y register: One of the index registers in the 65C02 microprocessor.

zero page: The first page (256 bytes) of the Apple IIc's memory, 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; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.



Bibliography

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.

Apple IIe Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1982.

Addendum to the Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1984.

Apple IIe Reference Manual. Cupertino, Calif.: Apple Computer, Inc., 1982.

Applesoft BASIC Programmer's Reference Manual, Volumes 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.

Leventhal, Lance. *6502 Assembly Language Programming*. Berkeley, Calif.: Osborne/McGraw-Hill, 1979.

Synertek Hardware Manual. 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 Manual. Santa Clara, Calif.: Synertek, Incorporated, 1976.

The only currently available manufacturer's programming manual for 6500 series microcomputers.

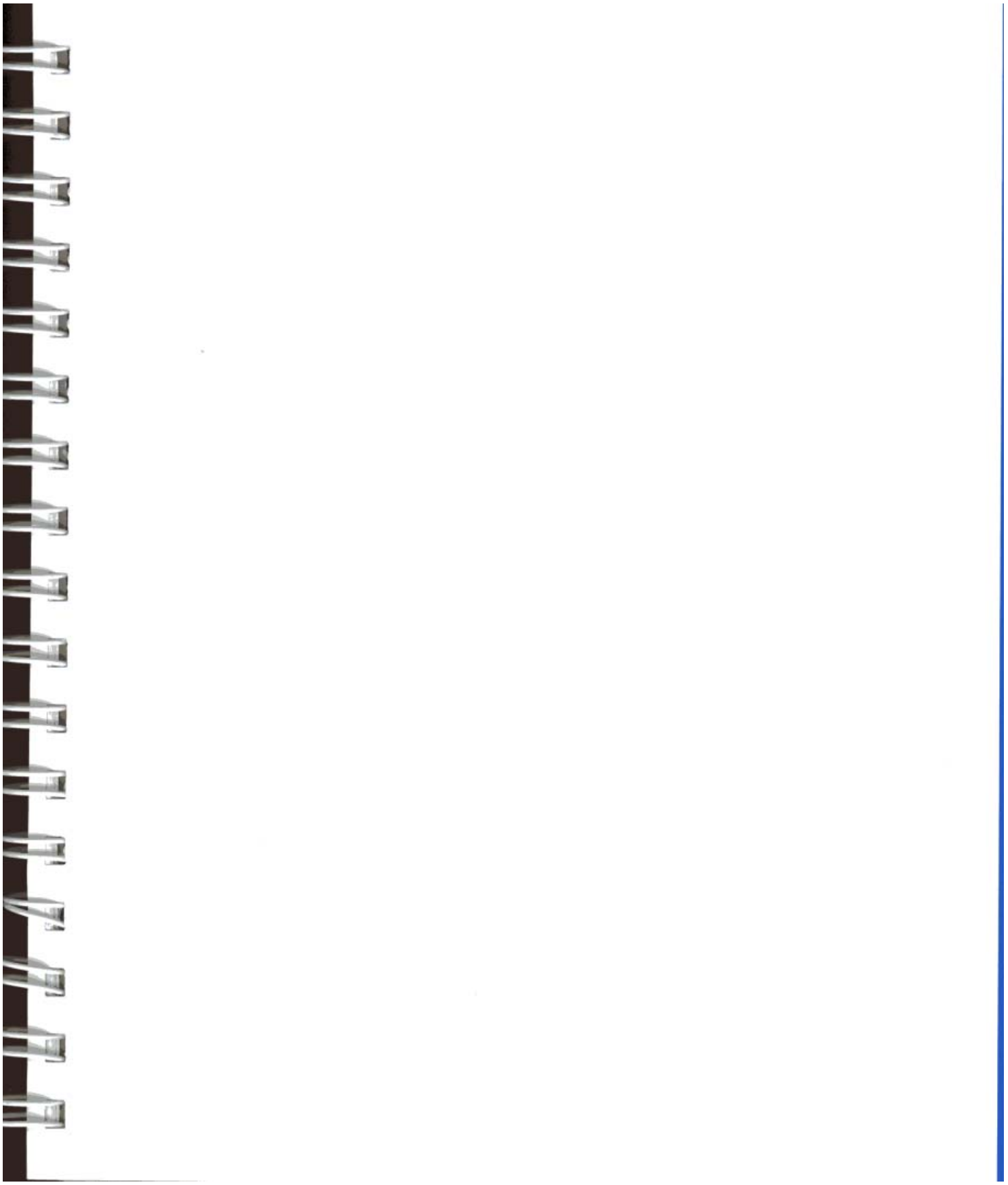
Watson, Allen, III. "A Simplified Theory of Video Graphics, Part I." *Byte* Vol. 5, No. 11 (November, 1980).

———. "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).



Index

References to entries in Volume 2 are in square brackets [ ].

Cast of Characters

A

D

E

F

J

K

L

M

Q

R

S

T

U

V

W

X

Y

Z

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576

030-1022-A

Printed in U.S.A.