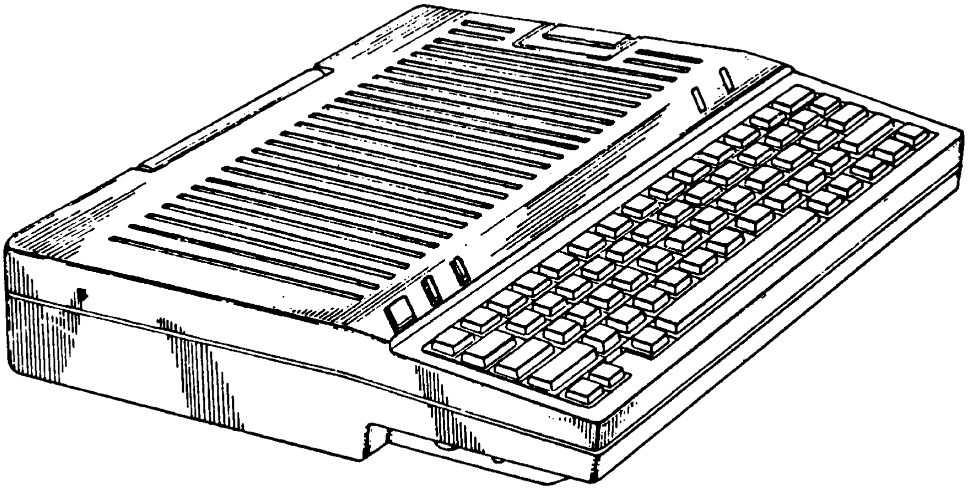


Apple //c Computer Preliminary Technical Reference Manual

AUTHOR

Joe R. Meyers
Apple Computer, Inc.

DOCUMENT DATES OF RECORD

December 1983

(This page is not part of the original document)

INTRODUCTION

What is this Document ?

This document contains a technical discussion of the Apple //c personal computer (code named LOLLY in this document) aimed at third party software and hardware developers. The document Foreword describes its purpose as:

> **This is the reference manual for the Lolly personal computer. It contains detailed descriptions of all the hardware and firmware that make up the Lolly.**

This manual was provided to internal Apple staff for review in January 1984 and is marked COMPANY CONFIDENTIAL.

This document is preliminary and is missing many tables and figures and some sections entirely.

This is assigned Apple product number A2L4030 and document number 030-0814-A.

Facts about this Document

Author:

Apple Computer, Inc.

Document Dates of Record:

December 1983

Owner:

Organization:	DigiBarn Computer Museum	(www.digibarn.com)
Curator:	Druce Damer	(http://www.damer.com/)

Document scanned by David Craig (shirlgato@cybermesa.com) in September 2007.

(This page is not part of the original document)

Property Statement

Disclaimer

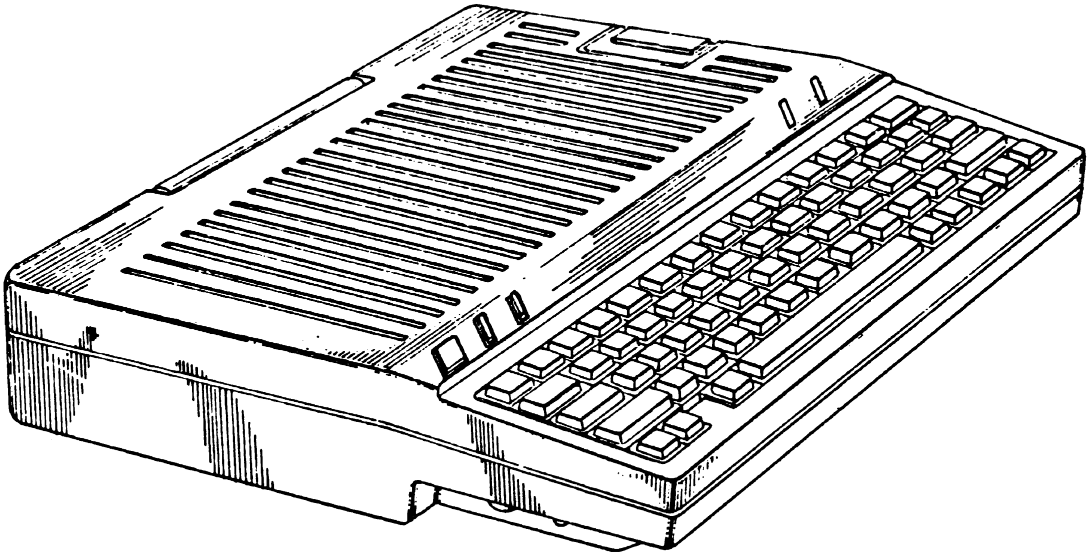
Apple //c Computer Preliminary Technical Reference Manual

Written by
Joe R. Meyers • Apple Computer, Inc • December 1983

Note

The Apple //c computer is given the code name LOLLY by this manual. Other code names were Chels, ET, IIb (book), IIp (portable), Pippin, VLC (very low cost), Elf, Yoda, Teddy, Jason, Sherry and Zelda.

The official name "Apple //c" appears on page 2-29 in this manual.



Lolly

REFERENCE MANUAL

Final Draft 11/83

file=lrmbl:front J R Meyers Final Draft 12/83

To: Distribution From: Joe Meyers

Date: January 5, 1984 Re: Final Lolly Reference Manual

Here is the final draft of the Lolly Reference Manual. Actually, it is in the form required for developers: Lolly is the name used in this draft, instead of the final public name. Also, page numbers are by chapter instead of sequential, and many Script codes should be added so the editor doesn't have to add them.

This draft incorporates comments and suggestions kindly furnished by:

Peter Baum
Bill Goldberg
Toni Calavas
Joe Ennis
Ernie Beernink
Lee Collings
Daunna Minnich
Steve Glass
Bob Etheredge
Conrad Rogers
Allen Watson
Beta Rvw Mtg
Ken Victor
Dick Huston
Rich Williams
Problem Res Grp

and countless anonymous letters and phone calls. Thank you all.

I do not yet know if Creative Services will let us use printed tabs as suggested in the beta draft review meeting. I have numbered the sections to make finding things easier even without tabs.

I have not completed typing of a number of tables, notably those in Appendixes B through F. Also, I still need to merge the pertinent entries from the Apple IIe Reference Manual and the Super Serial Card manual.

Only minor changes are planned before this document goes to the edit staff. Please let me know right away if you find any substantial errors. Please have comments to me no later than Friday, January 13, 1984. The manual is scheduled to go to the editors January 16.

This document is COMPANY CONFIDENTIAL. You are responsible for its security. Do not copy or distribute it without consulting me.

FINAL DRAFT DISTRIBUTION LIST

Document: Lolly Reference Manual   Date: January 5, 1984

Writer: Joe Meyers   Ext. 5176   M/S: 22-K

Name	Position	Department	Approval	M/S
Lee Collings	Product Manager	Marketing	_______	22-P
Peter Quinn	Project Leader	HW Engineering	_______	22-P
Ken Victor	Project Leader	FW Engineering	_______	22-P
Toni Calavas	People Manager	User Education	_______	22-K
Daunna Minnich	Project Supv	User Education	_______	22-K

S. Espinosa	22-K	F. Sommers	12-A	R. Cochran (2)	22-K
J. Hoyt (2)	18-B	B. Grimm	22-W	D. Hanttula	TR-1
J. Thompson	22-K	R. Broenen	18-K	N. Hecht	22-K
L. Sims	22-K	L. Curry	22-G	A. Huang	22-H
M. Matthews	TR-2	R. Rice	22-P	J. Ennis	22-P
P. Pahl	22-V	J. MacDougall	22-P	E. Beernink	22-P
K. Grey	22-E	R. Huston	22-P	R. Williams	22-P
S. Kerner	22-B	R. Beam	22-P	D. Larson	22-F
D. Marson	20-C	D. Farrar	22-Q	C. Rogers	22-P
J. Wallace	22-Q	A. Watson	22-K	P. Baum	22-W
S. Clark	22-G	J. Guzman	18-B	V. Scheiderer(2)	18-B
R. Etheredge	22-P	S. Glass (2)	22-G	B. Marks	22-H
P. Papageorge	22-V	S. Morningstar	22-G	G. Adams	22-Q
K. Walker	22-G	S. Hix	22-K	C. File	22-K

DEADLINE FOR COMMENTS: January 13, 1984

NOTICE

Apple Computer, Inc. reserves the right to make improvements in the product described in this manual at any time and without notice.

DISCLAIMER OF ALL WARRANTIES AND LIABILITY

[standard disclaimer goes here]

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

APPLE Product #A2L4030

[Class B RFI WARNING appears here]

030-0814-A

Title Page

Lolly @@@

REFERENCE MANUAL @@@@

Final Draft

file=lrmbl:front J. Meyers Final Draft 12/83

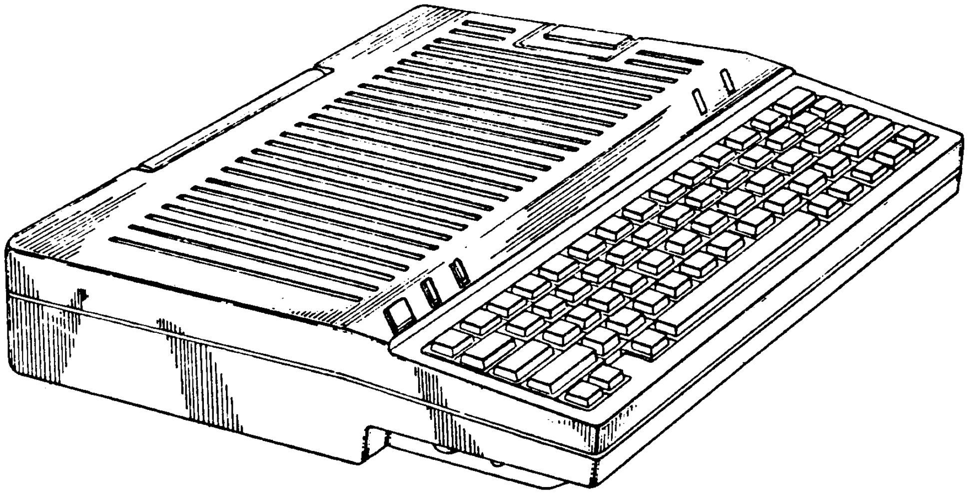
RADIO AND TELEVISION INTERFERENCE

[R & TVI blurb goes here]

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

TABLE OF CONTENTS



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Contents

**************************** Volume I ****************************

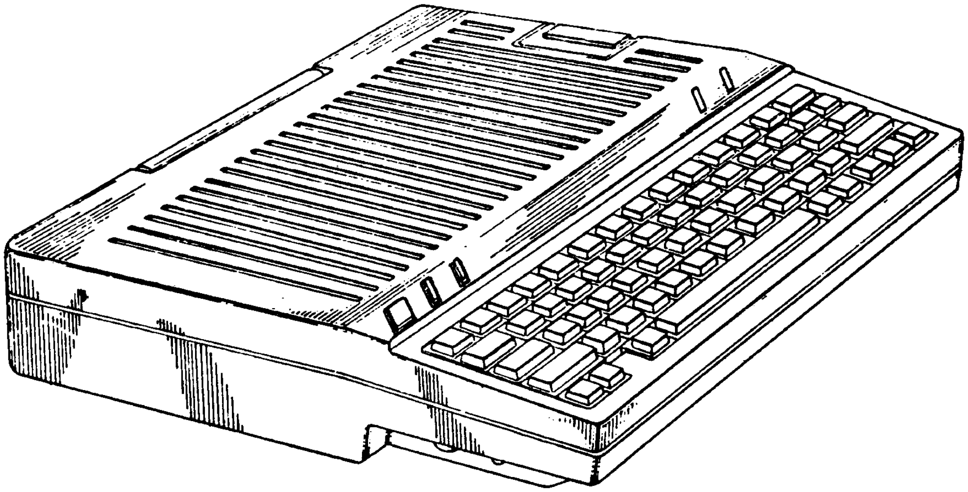
Chapter 2: Memory Organization and Control

**************************** Volume 2 ****************************

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

FOREWORD



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Foreword

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

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

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

<< Head 1 >>

Contents of This Manual

This manual presents first the physical characteristics of the Lolly (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 appendices contain summary tables and information relating to other Apple products in relation to the Lolly.

Chapter 1 identifies the main physical features of the Lolly.

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 Lolly 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 Lolly works.

Additional reference information appears in the appendices. 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 Lolly.

Appendix B is a memory map of the Lolly, including detailed tables of pages $00, $03 and $C0.

Appendix C lists the "published" firmware entry points, sorted by alphabet and 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 <u>Applesoft BASIC Reference Manual</u> (in two volumes).

Appendices D and E discuss what operating systems and languages, respectively, can run on the Lolly, and how they use Lolly's memory management and I/O firmware.

Appendix F contains an overview of the differences between the Lolly and the Apple IIe. This appendix also compares the Lolly with its serial ports to an Apple IIe with a Super Serial Card installed in it.

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

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

The glossary defines many of the technical terms used in this manual. These terms are also printed in bold face type where they first appear in the text.

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

Following the bibliography is a listing of the source code for the Monitor, enhanced video firmware, and input/output firmware contained in the Lolly. The listings do not include the built-in Applesoft interpreter, which is discussed in the Applesoft BASIC Reference Manual.

<< Head 1 >>

Symbols Used 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 Box >>---------------------

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

------------------------<< End Box >>----------------------

-----------------------<< Gray Box >>----------------------

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

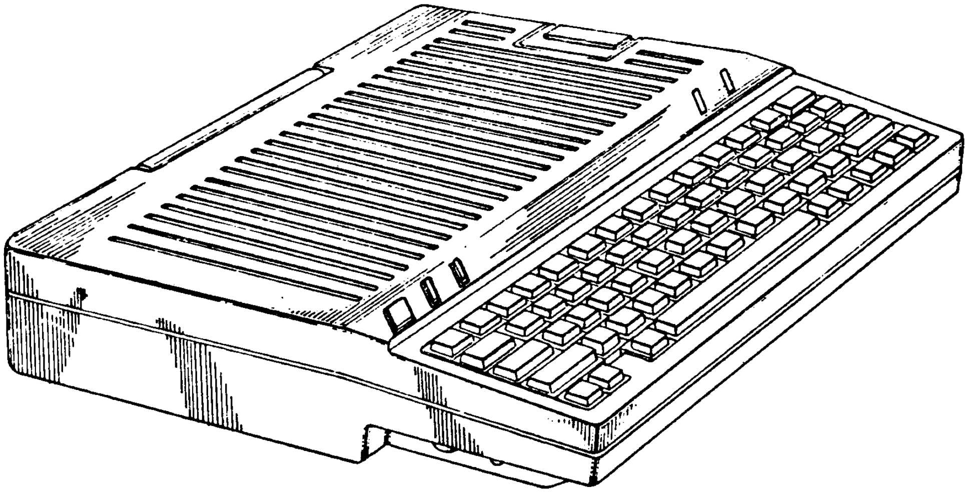
------------------------<< End Box >>----------------------

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

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 1 • INTRODUCTION



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 1

Introduction

Chapter 1

Introduction

This first chapter introduces you to the Lolly. 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.

<< Head 1 >>

1.1 Outside of Machine

Figure 1-1 is a photograph of a Lolly with a Standard USA keyboard. This chapter discusses both the Standard (Sholes) and Simplified (Dvorak) USA keyboards, as well as the other lights and switches on the front of the machine. Appendix G illustrates and discusses several international keyboard layouts.

------------------------------<< Figure >>------------------------------

[Figure 1-1]

`Figure 1-1.` Photograph of Lolly
with Standard USA Keyboard

Figure 1-2 is a diagram of the parts of the computer that you can see, hear or access from the outside. The Lolly 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 Lolly, but expansion is possible and easy with the connectors on the Lolly back panel.

------------------------------<< Figure >>------------------------------

[Figure 1-2]

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

<< Head 2 >>

1.1.1 The Keyboard

The front of the Lolly includes the keyboard (Figure 1-3), the computer's primary input device. It has a typewriter layout, upper and lowercase, and can generate all 128 of the characters in the ASCII character set, including control characters. The front of the computer also has a RESET key, 40/80 Column Switch, Keyboard switch, Power light, and Disk Active light.

-------------<< Gloss >>------------

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.

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

------------------------------<< Figure >>------------------------------

[Figure 1-3]

`Figure 1-3.` Keyboard and Front of Lolly

-------------------------------<< Table >>------------------------------

Number of keys:	63
Character encoding:	ASCII
Number of codes:	128
Features:	Automatic repeat, 2-key rollover
Special function keys:	RESET, OPEN-APPLE, SOLID-APPLE
Cursor movement keys:	LEFT-ARROW, RIGHT-ARROW, DOWN-ARROW, UP-ARROW RETURN, DELETE, TAB
Modifier keys:	CONTROL, SHIFT, CAPS LOCK, ESC
Front-panel switches:	40/80 Column switch, Keyboard switch
Front-panel lights:	Power light, Disk Active light

`Table 1-1.` Lolly Keyboard Specifications

<< Head 3 >>

Features

The Lolly 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 2-key rollover, which means if you type up to two keys before releasing a prior key, the new character will enter the computer the same as if the previous key was released first. (This is important for fast touch typists.)

<< Head 3 >>
Special Function Keys

The Lolly has three special function keys: the RESET key, and two keys marked with apples, one outlined (OPEN-APPLE), and one filled in (SOLID-APPLE).

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

The RESET key has a direct line to the 65C02 microprocessor: holding down the CONTROL key while pressing the RESET key causes the Lolly to restart processing with a program that puts the machine in a known state (Chapter 2).

-------------<< Gloss >>------------
So you don't accidentally destroy current work, the reset takes effect only if you hold down the CONTROL key while pressing RESET.

If you hold down both the CONTROL and the OPEN-APPLE key while pressing RESET, the computer will start up as if you had just turned it on.

-------------<< Gloss >>------------
Chapter 2 describes the results of the various reset procedures.

<< Head 3 >>
Cursor Movement Keys

Four of the cursor movement keys have arrows on them: left, right, down and up. The other three are RETURN, DELETE and TAB. All of them 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 the others generate.

<< Head 3 >>
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.

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

The CONTROL key, when pressed in combination with letter keys or certain other keys, produces ASCII control characters.

-------------<< Gloss >>------------
The other keys to use with the CONTROL key are: @ [\ ] ^ _ and their international equivalents (Appendix G).

The SHIFT key works the same on the Lolly as on an ordinary typewriter: it selects uppercase letters and the upper characters on the keys.

The CAPS LOCK key, in its down position, changes the letter keys to uppercase, but does not affect other keys.

The ESC key 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 Chapter 4). Many programs, including the built-in Monitor program, then interpret specific other keys as designating an `escape sequence.`

<< Head 3 >>

The 40/80 Column Switch

The 40/80 Column 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.

-------------<< Gloss >>------------
This switch takes effect only if the program or operating system you are using actually checks it. Chapter 4 describes the way programs can read this switch.

-----------------------<< Gray Box >>-----------------------

`Note:` Not all programs check this switch. Even programs that do check the switch may read it wrong unless you set it before running the program.

------------------------<< End Box >>-----------------------

<< Head 3 >>
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 Lolly, 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.

-------------<< Gloss >>------------
If you plan to use the Dvorak keyboard, you can change the keycaps to the layout shown in Figure 1-5. Carefully (so you don't break a key stalk and void your warranty) pry up the keys whose positions are changed, push them onto the stalks in their new positions, and press down the Keyboard switch.

------------------------------<< Figure >>------------------------------

[Figure 1-4]

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

-----------------------------<< Figure >>-----------------------------

[Figure 1-5.]

'Figure 1-5.' Simplified or 'Dvorak'
Keyboard (Keyboard Switch Down)

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 characters and key layout.

-------------<< Gloss >>------------

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

<< Head 3 >>

Power and Disk Active Lights

The green power light glows when normal power is present at the Lolly's internal power supply.

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

<< Head 2 >>

1.1.2 The Speaker

The Lolly has a loudspeaker in the bottom of the case, as shown in Figure 1-6. The speaker enables Lolly 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 Lolly case, and a mini-phono jack for connecting headphones or an external speaker.

The jack accepts either one-channel (monaural) or two-channel (stereo) plugs, although speaker output is monaural only. Inserting a plug disconnects the built-in speaker.

The way programs control the speaker is described in Chapter 4.

------------------------------<< Figure >>------------------------------

[Figure 1-6]

'Figure 1-6.' Lolly Speaker, Volume Control and Phone Jack

<< Head 2 >>

1.1.3 The Built-in Disk Drive

The Lolly 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, 16-sector disks. The drive door is on the right side of the Lolly case. Chapter 6 describes how to use the Lolly's disk I/O hardware and firmware.

-------------<< Gloss >>------------

I/O means input (information coming into the computer) and output (information going out of the computer).

-----------------------------<< Figure >>-----------------------------

[Figure 1-7]

'Figure 1-7.' Lolly Built-in Disk Drive

<< Head 2 >>

1.1.4 The Back Panel

The back panel of the Lolly (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 or a mouse
- 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 15 volt DC 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 >>------------------------------

[Figure 1-8]

'Figure 1-8.' Back Panel Connectors

<< Head 1 >>

1.2 Inside of Machine

Figure 1-9 is a diagram of the main components inside the Lolly computer. Chapter 11 discusses these components and how they work in further detail.

------------------------------<< Figure >>------------------------------

[Figure 1-9]

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

<< Head 2 >>

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 Lolly (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 >>-----------------------------

[Figure 1-10]

`Figure 1-10.` Lolly Power Supply
and Voltage Converter

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 it and the rest of the Lolly against short circuits and other mishaps.

-------------<< Gloss >>------------

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

<< Head 2 >>

1.2.2 The Main Logic Board

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

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

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

-----------------------------<< Figure >>------------------------------

[Figure 1-11]

`Figure 1-11.` Lolly Main Logic Board

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 Lolly, the 65C02 runs at 1 MHz and performs up to 500,000 eight-bit operations per second. The specifications of the 65C02 are given in Chapter 11; the 65C02 instruction set is given in Appendix A.

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 ROM in turn sends ASCII key codes to the processor. These devices are described in Chapter 11.

The character generator ROM converts display information to a form that display devices can use. This process is also described in Chapter 11.

The other ROMs contain the Monitor, the Applesoft BASIC interpreter, enhanced video firmware, and other input/output firmware. These ROMs are described in Chapter 11; the firmware they contain is described throughout this manual.
The Applesoft language interpreter is described in the Applesoft Tutorial and the Applesoft BASIC Programming Reference Manual.

Five of the large IC's are custom-made for the Lolly:

- The Memory Management Unit (MMU) contains most of the logic that controls memory addressing in the Lolly, which is described in Chapter 2.
- The Input/Output Unit (IOU) contains most of the logic that controls the built-in input and output features of the Lolly, which are described in Chapters 3 through 9.
- 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.
- The Integrated "Woz" Machine (IWM) is a single-chip version of the Apple Disk II controller card.

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

<< Head 2 >>

1.2.3 The Other Circuit Boards

The Lolly 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. They contain no user-serviceable parts. In particular, adjustment of disk drive speed must be done by an authorized Apple Service Center.

----------------------<< Warning Box >>---------------------

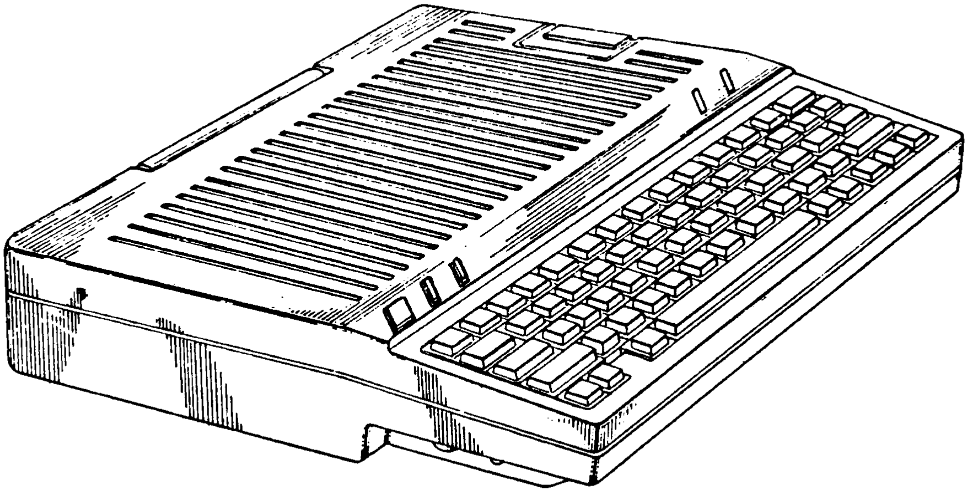
`Warning`
Do not attempt to adjust the speed of your built-in disk drive. If you do, you may damage it and you will void your warranty.

------------------------<< End Box >>-----------------------

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 2 • MEMORY



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

Chapter 2

Memory Organization and Control

Chapter 2

Memory Organization and Control

This chapter introduces you 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.

<< Head 1 >>

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 >>-------------------------------

[Figure 2-1]

`Figure 2-1.` Block Diagram Model of 65C02

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

------------<< Gloss >>------------
Each location holds 8 bits (one byte), so the 65C02 is called an 8-bit processor.

The five 8-bit registers in the 65C02 are

- The accumulator, or A register. The accumulator is like a work table 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 hand 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.

------------<< Gloss >>------------
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.

<< Head 1 >>

2.2 Overview of the Address Space

The Lolly's 65C02 microprocessor can address 65,536 (64K) memory locations. All of the Lolly's RAM, ROM and input and output (I/O) devices are accessed using addresses in this 64K address range. Some functions share the same addresses--but not at the same time. The Lolly controls its shared addresses using soft switches, which are described in Sections 2.4 and 2.5.

-------------<< Gloss >>------------
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.

All input and output in the Lolly 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.

Blocks of 256 address locations are called pages. 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.

-------------<< Gloss >>------------
Note that the page number equals the first two digits of a four-digit hexadecimal address. There are 256 pages of 256 bytes each in the address space. This kind of page is different from the display areas in the Lolly, which are sometimes referred to as Page 1 and Page 2.

<< Head 1 >>

2.3 Memory Map and Memory Switching

Figure 2-2 is a map of the Lolly'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 >>-----------------------------

[Figure 2-2]

Figure 2-2. Lolly Memory Map

<< Head 2 >>

2.3.1 Main RAM Addresses ($0000 - $FFFF)

The area labelled 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.

<< Head 2 >>

2.3.2 Auxiliary RAM Addresses ($0000 - $FFFF)

Auxiliary RAM is built into the Lolly. Some or all of auxiliary memory is present in an Apple IIe with one of the 80-Column cards installed (Appendix F). 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.

<< Head 2 >>

2.3.3 ROM Addresses ($C100 - $FFFF)

ROM addresses contain the built-in Lolly firmware. Addresses $C100 through $CFFF belong exclusively to ROM. Addresses $D000 through $FFFF are shared with main 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 Lolly:

- 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 80-column video support firmware occupies pages $C8 through $CF.

- Mouse firmware entry points are on page $C4.

- Disk I/O firmware entry points are on page $C6; external startup disk firmware begins on page $C7.

-----------------------<< Gray Box >>-----------------------

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

-----------------------<< End Box >>-----------------------

Pages $D0 through $EF (addresses $D000 through $EFFF) contain the Applesoft Interpreter firmware. The operation of this firmware is described in the Applesoft Reference Manual.

Pages $F0 through $FF (addresses $F000 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.

<< Head 2 >>

2.3.4 Hardware Addresses ($C000 - $C0FF)

The Lolly's built-in input and output functions, and its address space controls, 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) controls. Chapters 3 through 9 describe the Lolly's input and output locations. Appendix B lists all of these locations in address order, rather than by function.

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

- Data inputs. The only data input is location $C000, where the low-order seven bits (bits 6 to 0) represent the keyboard key just pressed.

- 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 hand controller 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 Lolly 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.

- Soft switches. Soft switches are two-position switches turned on by accessing one address and turned off by accessing another address. Most of these switches have a third address associated with them, for reading the state of the switch.

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

<< Head 1 >>

2.4 Bank-Switched Memory

The memory areas described in this section (Figure 2-3) are called bank-switched memory because so many banks (ranges) of addresses--one bank of ROM and up to four banks of RAM--occupy the same group of addresses near the upper end 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.

-------------<< Gloss >>------------

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.

-----------------------------<< Figure >>-----------------------------

[Figure 2-3]

`Figure 2-3.` Bank-switched Memory

<< Head 2 >>

2.4.1 Page Allocations

Pages zero and one are used by many of the 65C02 instructions, as discussed in Section 2.1. The ROM and RAM addresses in bank-switched memory are usually occupied by system software such as interpreters, compilers, and operating systems.

<< Head 3 >>

Page $00 (One-byte Addresses)

Several of the 65C02 microprocessor's addressing modes require the use of addresses in page zero, also called zero page. The Monitor, the interpreters, and the operating systems all make extensive use of page zero (Appendix B).

To use indirect addressing in assembly-language programs, you must store base addresses in page zero. At the same time, you must avoid interfering with the other programs that use page zero--the Monitor, the language interpreters, and the operating systems. One way to avoid conflicts is to use only those page-zero locations not already used by these other programs. Refer to Table B-1 in Appendix B.

As you can see from the table, 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: save the contents of part of page zero, use that part, then restore the previous contents before you pass control to another program.

<< Head 3 >>

Page $01 (The 65C02 Stack)

The 65C02 microprocessor uses page 1 as its stack--a place where it can store subroutine return addresses stored, 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.

The stack can hold only 256 bytes of information at a time. When you store the 257th byte in the stack, the stack pointer repeats itself, or wraps around, so that the new byte replaces the first byte stored, which is now lost. This writing over old data is called stack overflow, and when it happens, the program continues to run normally until the lost information is needed, whereupon the program produces unpredictable results.

<< Head 3 >>

Pages $D0 through $FF (ROM and RAM)

The memory address space from $D000 through $FFFF is used for both ROM and RAM. The 12K bytes of ROM (read-only memory) 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 another 16K bytes of auxiliary RAM in this 12K space, again with double occupancy in the address range $D000 through $DFFF.

-------------<< Gloss >>------------

RAM comes in 16K chips, but only 12K addresses are available in the high range of memory: addresses $C000 through $CFFF are set aside for hardware and I/O. So switches were added to the hardware page to select a second range of $D000 space for main and auxiliary RAM.

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

<< Head 2 >>

2.4.2 Using Bank Selector Switches

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

1. Select either RAM or ROM in this memory space.
2. Enable or inhibit writing to the RAM (write-protect) 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 Box >>---------------------

`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.

-----------------------<< End Box >>-----------------------

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.

------------------------------<< Table >>------------------------------

Switch Address	Write RAM	Read RAM	Read ROM	4K RAM Bank: First	Second	Notes
$C080		x			x	
$C081	x		x		x	*
$C082			x		x	
$C083	x	x			x	*
$C088		x		x		
$C089	x		x	x		*
$C08A			x	x		
$C08B	x	x		x		*

$C011	Read whether $D000 bank 2 (bit 7 = 1) or $D000 bank 1 (bit 7 = 0) is currently selected.
$C012	Read whether RAM (bit 7 = 1) or ROM (bit 7 = 0) is currently selected for reading.
$C009	Select alternate RAM addresses for bank-switched memory.
$C008	Select main RAM addresses for bank-switched memory.
$C016	Read status of $C009/$C008 (ALTZP) switch on bit 7 (1 = auxiliary RAM selected).

`Table 2-1.` Bank Select Switches.
* To write-enable RAM, read this
address twice.

------------------------------<< Figure >>------------------------------

[Figure 2-4]

`Figure 2-4.` Read ROM

------------------------------<< Figure >>------------------------------

[Figure 2-5]

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

------------------------------<< Figure >>------------------------------

[Figure 2-6]

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

------------------------------<< Figure >>------------------------------

[Figure 2-7]

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

-----------------------------<< Figure >>-----------------------------

[Figure 2-8]

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

-----------------------------<< Figure >>-----------------------------

[Figure 2-9]

`Figure 2-8.` Read and Write RAM and
Use First $D0 Bank

-----------------------------<< Figure >>-----------------------------

[Figure 2-10]

`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.

<< Head 1 >>

2.5 48K Memory

The major portion of the Lolly's memory space--48K of it--is available for your programs and data. The 48K RAM memory extends from location $200 to location $BFFF (Figure 2-11). The actual amount of free space depends on what language or operating system you are using, and what video display needs your program has.

----------------------------<< Figure >>------------------------------

[Figure 2-11]

`Figure 2-11.` 48K Memory Map

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.

<< Head 2 >>

2.5.1 Page Allocations

Most of the Lolly'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 BASIC interpreters, and whatever video display you want to use.

----------------------<< Gray Box >>----------------------

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

-----------------------<< End Box >>----------------------

<< Head 3 >>

Page $02 (The Input Buffer)

The GETLN input routine (described in Chapter 3) uses page 2 as its keyboard-input buffer. The size of this buffer (256 bytes) sets the maximum size of input strings. 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.

-------------<< Gloss >>------------
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 speed.

<< Head 3 >>
Page $03 (Global Storage and Vectors)

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

-------------<< Gloss >>------------
Global storage refers to an area reserved for information that programs pass to one another. 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.

<< Head 3 >>
Pages $04 Through $07 (Text and Low-Res 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.

Text and Low-res Page 1X (TLP1X) is an identical display page occupying auxiliary memory pages $08 through $0B. This pair of Text and Low-Res graphics pages are interleaved to produce 80-column text display (Chapter 5).

There are 128 locations in this area (64 in main RAM, 64 in auxiliary RAM) that are not displayed on the screen. These locations are called `screen holes` (Section 3.4.6).

----------------------<< Warning Box >>---------------------

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

------------------------<< End Box >>-----------------------

<< Head 3 >>
Pages $08 Through $0B (Text and Low-Res Page 2)

Display Page 2, the alternate text and low-resolution-graphics display buffer, occupies main memory pages $08 through $0B. Most programs do not use Page 2 for displays.

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

-----------------------<< Gray Box >>-----------------------

Note: Lolly firmware does not provide a way to use the second pair of Text and Low-Res graphics pages for 80-column text display.

------------------------<< End Box >>-----------------------

<< Head 3 >>
Page $08 (Communication Port Buffers)

Serial port 2 (Chapter 8) 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 an output buffer. These buffers increase the data transfer rates possible with the serial communication port.

<< Head 3 >>
Pages $20 Through $3F (High-Res Page 1)

The primary high-resolution-graphics display buffer, called High-resolution Page 1 (HRP 1), 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 (HRP 1X) is an identical display page occupying auxiliary memory pages $20 through $3F.

The Lolly can display double-high-resolution graphics (Chapter 5) by interleaving HRP 1 and HRP 1X.

<< Head 3 >>

Pages $40 Through $5F (High-Res 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 available as a second high-resolution that your program can prepare for display while HRP 1 is being displayed.

High-resolution_graphics Page 2X (HRP 2X) occupies auxiliary memory pages $0 through $5F.

----------------------<< Gray Box >>----------------------

Note: Lolly firmware does not provide a way to use the second pair of High-resolution graphics pages for double-high-resolution display.

------------------------<< End Box >>---------------------- -

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

<< Head 2 >>

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 Box >>---------------------

`Warning`

This discussion assumes that the 80STORE switch, used to control display memory, is off. For details, refer to Section 2.5.4.

-----------------------<< End Box >>-----------------------

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 2-11 and 2-12 illustrate how the switches work.

------------------------------<< Table >>------------------------------

Name	Function	Location Hex	Decimal		Notes
RAMRD	Read auxiliary memory	$C003	49155	-16381	Write
	Read main memory	$C002	49154	-16382	Write
	Read RAMRD (1 = aux)	$C013	49171	-16365	Read
RAMWRT	Write auxiliary memory	$C005	49157	-16379	Write
	Write main memory	$C004	49156	-16380	Write
	Read RAMWRT (1 = aux)	$C014	49172	-16354	Read

`Table 2-2.` 48K Memory Switches.
Note: 80STORE must be off
(Table 2-6).

----------------------------<< Figure >>------------------------------

[Figure 2-11]

`Figure 2-11.` 48K RAM Selection:
Split Pairs

----------------------------<< Figure >>----------------------------

[Figure 2-12]

`Figure 2-12.` 48K RAM Selection:
One Side Only

<< Head 2 >>

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.

----------------------<< Gray Box >>----------------------

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.

-----------------------<< End Box >>----------------------

You use these built-in subroutines and all of the I/O subroutines in the same way: by making subroutine calls to their starting locations. Those locations are shown in Table 2-3.

------------------------------<< Table >>------------------------------

Routine Name	Location	Description
MOVEAUX	$C311	Moves data blocks between main and auxiliary memory
XFER	$C314	Transfers program control between main and auxiliary memory

----

`Table 2-3.` 48K RAM Transfer Routines

<< Head 3 >>

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 Box >>---------------------

`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.

-----------------------<< End Box >>-----------------------

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 Lolly's built-in routines. The addresses of these byte pairs are shown in Table 2-4.

------------------------------<< Table >>------------------------------

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

`Table 2-4.` Parameters for MOVEAUX Routine

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 registers 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.

<< Head 3 >>

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 >>------------------------------

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

`Table 2-5.` Parameters for XFER Routine

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 (CLV instruction) 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. XFER saves the accumulator and the transfer address on the current stack, then sets up the soft switches for the parameters you have selected and jumps to the new program.

----------------------<< Warning Box >>---------------------

`Warning`
It is your responsibility as the programmer to save the current stack pointer somewhere in the current memory space before using XFER and to restore it after regaining control. Failure to do so will cause program errors.

-----------------------<< End Box >>----------------------

<< Head 2 >>

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 circumatances your program may want to control video display pages separately. The switches discussed in this section override the effects of RAMRD and RAMWRT for display pages only.

Three switches are involved in the display page selection process. Each of them has three locations assigned to it: one to turn it on, one to turn it off, and a third to read its state (Table 2-6).

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.

-------------<< Gloss >>------------

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.

Here is how these switches work.

- 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 (Figures 2-13 and 2-14).
- If 80STORE is on, it overrides RAMRD and RAMWRT with respect to the display pages selected by HIRES and PAGE2 (Figure 2-15)..

------------------------------<< Table >>------------------------------

Name	Function	Location Hex	Location Decimal		Notes
80STORE	On: PAGE2 selects 1&1X	$C001	49153	-16383	Write
	Off: PAGE2 selects 1&2	$C000	49152	-16384	Write
	Read 80STORE (1 = on)	$C018	49176	-16360	Read
PAGE2	TLP2, HRP2; TLP1X, HRP1X	$C055	49237	-16299	1
	TLP1; HRP1	$C054	49236	-16300	1
	Read PAGE2 (1 = 2 or 1X)	$C01C	49180	-16356	Read
HIRES	On: PAGE2 selects HRPs	$C057	49239	-16297	1
	Off: PAGE2 selects TLPs	$C056	49238	-16298	1
	Read HIRES (1 = on)	$C01D	49181	-16355	Read

`Table 2-2.` Display Memory Switches

------------------------------<< Figure >>------------------------------

[Figure 2-13]

`Figure 2-13.` PAGE2 Selections with
80STORE Off and HIRES Off

----------------------------<< Figure >>------------------------------

[Figure 2-14]

`Figure 2-14.` PAGE2 Selections with
80STORE Off and HIRES On

----------------------------<< Figure >>------------------------------

[Figure 2-15]

`Figure 2-15.` PAGE2 Selections with
80STORE On

<< Head 1 >>

2.6 The Reset Routine

A procedure called the reset routine (Figure 2-16) puts the Lolly into a known state when it has just been turned on or you hold down the CONTROL key while pressing RESET. The reset routine puts the Lolly into its normal operating mode and restarts the resident program.

---------------------------<< Figure >>------------------------------

[Figure 2-16]

`Figure 2-16.` RESET Routine Flowchart

When you initiate a reset, hardware in the Lolly 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 sets the keyboard and display as the standard input and output devices (Chapter 3). It masks mouse interrupts and sets mouse defaults (Chapter 9). Finally, it enables DHIRES switch access, clears the keyboard strobe, and sounds the speaker.

The Lolly 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, as described below in the section, "The Reset Vector". 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.

<< Head 2 >>

2.6.1 The Cold-start Procedure (Power On)

If the reset vector is not valid, either the Lolly 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 below, 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 has been completed, the reset routine will continue without using the disk, and pass control to the built-in Applesoft interpreter.

<< Head 2 >>

2.6.2 The Warm-start Procedure (CONTROL-RESET)

Whenever you press CONTROL-RESET when the Lolly 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 resident program--which is the built-in Applesoft interpreter if no other interpreter or operating system is present.

If the resident program is indeed Applesoft, 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.

----------------------<< Gray Box >>----------------------

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

-----------------------<< End Box >>-----------------------

<< Head 2 >>

2.6.3 Forced Cold Start (OPEN-APPLE CONTROL-RESET)

If a program has loaded 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 the CONTROL key and the OPEN-APPLE key, then pressing and releasing the RESET key.

-----------------------<< Gray Box >>-----------------------

When you want to stop a program unconditionally--for example, to start up the Lolly with some other program--you should use the forced cold-start reset, CONTROL-OPEN-APPLE-RESET, instead of turning the power off and on.

------------------------<< End Box >>-----------------------

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.

<< Head 2 >>

2.6.4 The Reset Vector

When you reset the Lolly, the reset routine transfers control to the resident program by means of an address stored in page 3 of main RAM. This address is called a vector because it directs program control to a specified destination. There are several other vector addresses stored in page 3, as shown in Table 2-7.

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 1010 and 1011 (hexadecimal $3F2 and $3F3). It then stores a validity-check byte, also called the power-up byte, at location 1012 (hexadecimal $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 Lolly, 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 Lolly, 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.

The reset routine has a subroutine that generates the validity-check byte for the current reset vector. Chapter 10 describes how to use this subroutine (which is at location $FB6F). When your program finishes, it can return the Lolly to normal operation by restoring the original reset vector and again calling the subroutine to fix up the validity-check byte.

------------------------------<< Table >>------------------------------

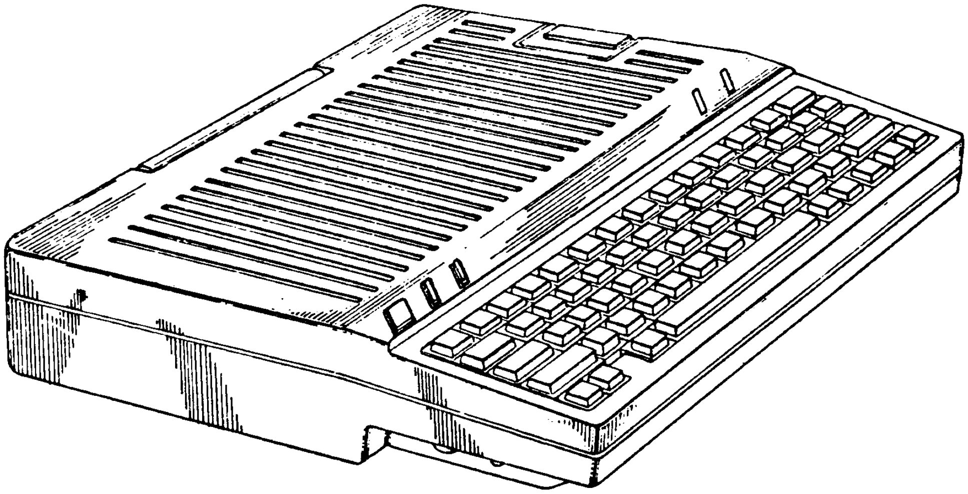
Vector address Hex	Vector function
$3F0 $3F1	Address of the subroutine that handles BRK requests (normally $59, $FA).
$3F2 $3F3	Reset vector (see text).
$3F4	Power-up byte (see text).
$3F5 $3F6 $3F7	Jump instruction to the subroutine that handles Applesoft "&" commands (normally $4C, $58, $FF).
$3F8 $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.
$3FE $3FF	Interrupt vector (address of the subroutine that handles interrupt requests).

`Table 2-7.` Page 3 Vectors

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 3 • INTRODUCTION TO I/O



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 3

Introduction to Lolly I/O

Chapter 3

Introduction to Lolly I/O

This chapter is an introduction to the input/output capabilities of the Lolly: 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: direct I/O, and using interrupts.

<< Head 1 >>

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 a Lolly 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 (checkerboard cursor), and of C3KEYIN and C3COUT1 if that firmware is on (flashing solid cursor).

<< Head 2 >>

3.1.1 Changing the Standard I/O Links

The standard I/O links are two pairs of locations in the Lolly that are used for controlling character input and output.

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 Lolly is currently using for single-character output. This address is normally $FDF0, the address of routine COUT1.

When you issue a \PR#n\ from BASIC or an \n CONTROL-P\ from the Monitor (Chapter 10), the Lolly 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.

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 Lolly 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 Box >>---------------------

`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 command with a carriage return (to be sure that what follows begins a new line), then PRINT \CONTROL-D\ followed immediately by the IN# or PR# command.

------------------------<< End Box >>----------------------

After changing CSW or KSW, assembly-language programs 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.

-----------------------<< Gray Box >>-----------------------

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

-----------------------<< End Box >>-----------------------

<< Head 1 >>

3.2 Standard Input Features

The Lolly'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"). By making repeated calls to RDKEY, GETLN accepts a sequence of characters terminated with a carriage return. Thus GETLN allows line-oriented input using the current input routine.

-------------<< Gloss >>------------

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

<< Head 2 >>

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 above). The cursor displayed by RDKEY is a flashing version of whatever character happens to be at that position on the screen. It is usually a space, so the cursor appears as a blinking rectangle.

<< Head 2 >>

3.2.2 KEYIN Input Subroutine

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

KEYIN handles the problem of displaying a cursor without using flashing format. 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, KEYIN displays a steady inverse space (rectangle).

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.

<< Head 2 >>

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, described below in the section Editing with GETLN.

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

--------------------------------<< Table >>--------------------------------

Prompt Character	Program requesting input
?	User's BASIC program (INPUT statement)
]	Applesoft BASIC (Appendix E)
>	Integer BASIC (Appendix E)
*	Firmware Monitor (Chapter 10)
!	Mini-assembler (DOS Toolkit, Appendix D)

Table 3-1. Prompt Characters

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

-------------<< Gloss >>------------

Control characters echoed by GETLN are not executed.

------------------------<< End Box >>----------------------

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 the RETURN key (or CONTROL-X to cancel the line: see Table 3-4); then it clears the remainder of the line the cursor is on, stores the carriage-return code in the buffer, sends the carriage-return code to the display, and returns to the calling program.

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

<< Head 2 >>

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 the ESC key, releasing it, and then pressing some other key, as shown in Table 3-2.

-------------<< Gloss >>------------
Be sure to release the ESC key 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 cursor-motion keys I, J, K and M without pressing the ESC key again. This enables you to perform repeated cursor moves by holding down the appropriate key.

When GETLN is in escape mode, it displays a plus sign in inverse format as the cursor. You leave escape mode by typing any key other than a cursor-motion key.

-------------<< Gloss >>------------
The escape codes with the arrow keys are the standard cursor-motion keys on the Lolly. The escape codes with the I, J, K, and M keys are the standard cursor-motion keys on the Apple II and II Plus, and are present on the Lolly for compatability.

-------------------------------<< Table >>-------------------------------

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 up 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 right one line and exits from escape mode	
ESC D or d	Moves the cursor down 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	
ESC I or i ESC up-arrow	Moves the cursor up one line; remains in escape mode	1
ESC J or j ESC left-arrow	Moves the cursor left one space; remains in escape mode	1
ESC K or k ESC right-arrow	Moves the cursor right one space; remains in escape mode	1
ESC M or m ESC down-arrow	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-Q	Deactivates the enhanced video firmware; sets links to KEYIN and COUT1; restores normal window size (Table 3-5): exits from escape mode	2

`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 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.

<< Head 2 >>

3.2.5 Editing with GETLN

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

<< Head 3 >>

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.

<< Head 3 >>

Backspace

When you press the LEFT-ARROW key (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 display position and the cursor back one space. If you type another character now, it will replace the character you backspaced over, both on the display and in the line buffer.

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

-------------<< Gloss >>------------

The cursor does not move if the deleted character was an (invisible) control character.

<< Head 3 >>

Retype

The RIGHT-ARROW key (or CONTROL-U) has a function that is complementary to the backspace function. When you press the RIGHT-ARROW key, GETLN picks up the character at the display position just as if it had been typed on the keyboard. You can use this procedure to pick up characters that you have just deleted by backspacing across them. You can use the backspace and retype functions with the cursor-motion functions to edit data on the display (see Section 3.2.4).

<< Head 1 >>

3.3 Standard Output Features

The standard output routine is named COUT, pronounced C-out, which stands for character out. 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.

<< Head 2 >>

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 left-most position on the next line down. If this would move the cursor position past the end of the last line in the window, COUT1 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 below do, and they use this cursor position. If some other routine displays a cursor, it will not necessarily put it in the cursor position used by COUT1 or C3COUT1.

----------------------<< Warning Box >>----------------------

`Warning`
When the video firmware is active, the value of CH is kept at 0 and the true horizontal position is stored at $57B. Either of these locations can be updated to move the cursor; to read the cursor position, however, use only $57B.

------------------------<< End Box >>------------------------

<< Head 2 >>

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, as described in Section 3.2.4. The stop-list function, described separately, can only be invoked from the keyboard.

-------------------------------<< Table >>------------------------------

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

Table 3-3. Control Characters with COUT1.

<< Head 2 >>

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 from the keyboard, either by typing the control character listed or by using the appropriate escape code, as described in the section "Escape Codes with GETLN". The stop-list function, described separately, can only be invoked from the keyboard.

------------------------------<< Table >>------------------------------

Control Character	ASCII Name	Lolly Name	Action Taken by C3COUT1	Notes
CONTROL-G	(BEL)	bell	Produces a 1000Hz 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)	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	(DS3)	stop-list	Stops listing characters on the display until another key is pressed.	2

`Table 3-4a.` 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.

------------------------------<< Table >>------------------------------

Control Character	ASCII Name	Lolly Name	Action Taken by C3COUT1	Notes
CONTROL-U	(NAK)	quit	Deactivates enhanced video firmware, homes cursor, and clears screen.	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-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
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-^	(RS)	gotoXY	Using the next two characters, minus 32, as one-byte X and Y values, moves the cursor position to CH=X, CV=Y.	1,2

`Table 3-4b.` Control Characters with C3COUT1, continued. (1) Only available when video firmware is active. (2) gotoXY is supported under Pascal, but not under BASIC.

<< Head 2 >>

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 the CONTROL key while pressing the S key). Whenever COUT1 or C3COUT1 gets a carriage return from the program, it checks for CONTROL-S. If it is in the input string, COUT1 or C3COUT1 stops and waits for another keypress. When you want COUT1 or C3COUT1 to resume, press another key; COUT1 or C3COUT1 will send the carriage return it got earlier to the display, then continue normally. The character code of the key you pressed to resume displaying is ignored unless it is a 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.

<< Head 2 >>

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 Box >>---------------------

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

-----------------------<< End Box >>-----------------------

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.

----------------------<< Warning Box >>---------------------

`Warning`
Any time you change the boundaries of the text window, you should make sure that the current cursor position (stored at CH and CV) is inside the new window. If it is outside, it is possible for COUT1 or C3COUT1 to put characters into memory locations outside the display page, possibly destroying programs or data.

-----------------------<< End Box >>-----------------------

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

----------------------<< Warning Box >>---------------------

`Warning`
Window width adjustments are not supported under Pascal.

-----------------------<< End Box >>-----------------------

------------------------------<< Table >>------------------------------

Window Parameter	Location		Minimum Value		Normal values: 40col.		80col.		Maximum Values: 40col.		80col	
	Dec	Hex	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

`Table 3-5.` Text Window Memory Locations

<< Head 2 >>

3.3.6Normal, 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 Lolly displays what is called the primary character set; if the video firmware is on, the Lolly displays what is called the alternate character set.

-------------<< Gloss >>------------
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 (TM). 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
```

<< Head 3 >>

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.

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.

-------------<< Gloss >>------------
For a brief explanation of logical
functions, refer to Appendix H.

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.

------------------------<< Gray Box >>------------------------

To avoid unusual character display results, use only the three values discussed in the preceding paragraph.

------------------------<< End Box >>------------------------

Character values from $80 through $9F are interpreted as control characters and are not displayed.

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

<< Head 3 >>

Alternate Character Set Display

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 (Chapter 5).

First the character is logically ORed with the value $80, thus setting bit 7 to 1 if it is not already on. If the resulting value is in the range $80 through $9F, it is interpreted as a control character and not displayed. Values $A0 through $FF are displayed.

Only bit 7 of the inverse flag (at location $32) is used to further modify the character value. If inverse flag 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 uppercase set. If Mousetext has been turned on, the values $40 through $5F are left unchanged, and they display as Mousetext icons (Chapter 5).

<< Head 1 >>

3.4 Port I/O

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

<< Head 2 >>

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#1 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 >>------------------------------

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/hand control	Mouse and hand controllers	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

`Table 3-6.` Port Characteristics

<< Head 2 >>

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 Lolly ports except the disk port conform to this protocol.

------------------------------<< Table >>------------------------------

Address	Value	Description
$Cn05	$38	Pascal firmware card/port identifier (also SEC instruction; BASIC input entry point)
$Cn07	$18	Pascal firmware card/port identifier (also CLC instruction: BASIC output entry point)
$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 Lolly): $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
$CnFF	11	Firmware revision level

`Table 3-7.` Firmware Protocol Locations

<< Head 2 >>

3.4.3 Port I/O Space

By a convention used in other Apple II series machines, each port or slot has exclusive use of sixteen memory locations of the form #C800 + $n00, 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 Lolly.

-------------<< Gloss >>------------

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

------------------------------<< Table >>------------------------------

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

`Table 3-8.` Port I/O Locations

<< Head 2 >>

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 Lolly, this space is dedicated to port firmware. However, I/O ROM space in the Lolly 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.

<< Head 2 >>

3.4.5 Expansion ROM Space

The 2K-byte memory space from $C800 to $CFFF in the Lolly--called expansion ROM space on Apple II, II Plus and IIe--contains the enhanced video firmware and some miscellaneous firmware routines.

<< Head 2 >>

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.

---------------------<< Warning Box >>---------------------

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

-----------------------<< End Box >>-----------------------

------------------------------<< Table >>------------------------------

Base Address	Port 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

`Table 3-9.` Port Screen-hole Memory Locations.

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

<< Head 1 >>

3.5 Interrupts

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) command, control is transferred through the BRK vector ($3F0- - $3F1). Otherwise, control is transferred through the IRQ vector ($3FE - $3FF).

Table 3-10 lists the types of IRQ-causing interrupts implemented on the Lolly, and the locations for interrupt enable, disable, read and reset, and their associated vectors (addresses of interrupt handling routines).

----------------------<< Warning Box >>---------------------

`Warning`
You are strongly urged to use the interrupt-handling capabilities built into the Lolly firmware. If you do write your own, be sure to study the interrupt routines in the firmware listings (Volume II).

------------------------<< End Box >>----------------------

-------------<< Gloss >>------------
The P register is the processor status register. The 65C02 sets bit 4 if it decodes a BRK instruction ($00).

-----------------------------<< Table >>-----------------------------

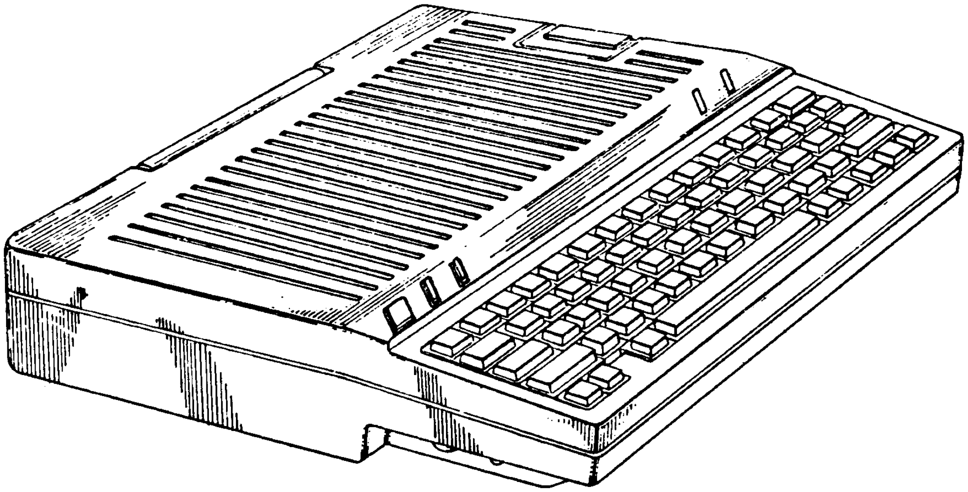
Name	Cause	Enable/ Disable	Read	Reset	Vector
BRK	BRK instruction	none	P bit 4	autom	$3F0-$3F1
TXE	Serial transmit buffer empty	(handled by I/O firmware)			
RXF	Serial receive buffer full	(handled by I/O firmware)			
DCD	Serial Data Carrier Detect	(handled by I/O firmware			
VBLINT (1)	Vertical Blanking	$C05B/ $C05A	$C041	$C019	
XINT (1,2)	X0 (mouse)	$C059/ $C058	$C040	$C015	
YINT (1,2)	Y0 (mouse)	$C058/ $C059	$C040	$C017	
NMI	(non-maskable interrupt)	-------not used-------			
EXTINT	(external interrupt)	-------not used-------			
KSTRB	(keyboard interrupt)	no	$C000	$C010	

`Table 3-10.` Interrupts and Their Associated Addresses (1) Only if IOUDIS is off. (2) If you read $C040, bit 7 is 1 if either X0 or Y0 interrupt has occurred. Reading or writing address $C048 resets both XINT and YINT.

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 4 • KEYBOARD & SPEAKER



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 4

Keyboard and Speaker

Chapter 4

Keyboard and Speaker

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

<< Head 1 >>

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 >>------------------------------

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 it).

Addresses

Hardware locations:

Location	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	OPEN-APPLE key status on bit 7; 1 = pressed (also game input switch 0, Chapter 9)

$C062	SOLID-APPLE key status on bit 7; 1 = pressed (also game input switch 1, Chapter 9)

Monitor firmware routines:

Location	Name	Description
$FD6A	GETLN	Gets an input line with prompt (Chapter 3).
$FD67	GETLNZ	Gets an input line.
$FD6F	GETLN1	Gets an input line, but with no preceding prompt.
$FD1B	KEYIN	The keyboard input subroutine (Chapter 3).
$FD35	RDCHAR	Gets an input character or escape code.
$FD0C	RDKEY	The standard character input subroutine (Chapter 3).

Use of other pages:

Page 2:	The standard character string input buffer. (See GETLN description in Chapter 3.)

`Table 4-1.` Keyboard Input Characteristics

<< Head 2 >>

4.1.1 Reading the Keyboard

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

Your programs can get the code for the last key pressed by reading the keyboard-data location. The low-order seven bits of the byte at the keyboard location contain the character code; the high-order bit of this byte is the strobe bit, described below.

Your program can find out whether any key is down (except the APPLE keys, CONTROL, SHIFT, CAPS LOCK or RESET) by reading from location $C010. The high-order bit (bit 7) of the byte you read at this location is called Any-key-down (AKD); it is 1 if a key is down, and 0 if no key is down.

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

------------------------<< Gray Box >>------------------------

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

------------------------<< End Box >>------------------------

After the keyboard strobe has been cleared, it remains low until another key is pressed. Even after you have cleared the strobe, you can still read the character code at the keyboard location. The data byte has a different value, because the high-order bit is no longer set, but the ASCII code in the seven low-order bits is the same until another key is pressed. Tables 4-2a and 4-2b show the ASCII codes for the keys on the Lolly keyboard.

-----------------------------<< Table >>-----------------------------

Key	Normal	Char	Control	Char	Shift	Char	Both	Char
DELETE	7F	DEL	7F	DEL	7F	DEL	7F	DEL
L-ARROW	08	BS	08	BS	08	BS	08	BS
TAB	09	HT	09	HT	09	HT	09	HT
D-ARROW	0A	LF	0A	LF	0A	LF	0A	LF
U-ARROW	0B	VT	0B	VT	0B	VT	0B	VT
RETURN	0D	CR	0D	CR	0D	CR	0D	CR
R-ARROW	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	36	6	5E	^	5E	^
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	|	7F	DEL

`Table 4-2a.` Keys and ASCII Codes.
Codes are shown here in hexadecimal;
to find the decimal equivalents, use
Tables 4-3a and 4-3b.

------------------------------<< Table >>------------------------------

Key	Normal	Char	Control	Char	Shift	Char	Both	Char
]}	5D	]	1D	GS	7D	}	1D	GS
`~	60	`	60	`	7E	~	7E	~
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB

X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

`Table 4-2b. Keys and ASCII Codes, continued. Codes are shown here in hexadecimal; to find the decimal equivalents, use Tables 4-3a and 4-3b.

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

Another key that doesn't generate a code is the RESET key, located at the upper-left corner of the keyboard; it is connected directly to the Lolly's circuits. Pressing the RESET key with the CONTROL key depressed normally causes the system to stop whatever program it's running and restart itself. This restarting process is called the Reset Cycle, and it is described in Chapter 2.

------------------------------<< Table >>------------------------------

Control			Special			Uppercase			Lowercase		
Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o

`Table 4-3a.` The ASCII Character Set

------------------------------<< Table >>------------------------------

Control			Special			Uppercase			Lowercase		
Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

'Table 4-3b.' The ASCII Character Set, continued.

<< Head 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.

<< Head 3 >>

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.

<< Head 3 >>

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 a CONTROL-X, then GETLN1 issues the prompt at location $33 when it gets another line.

<< Head 3 >>

RDCHAR

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

<< Head 1 >>

4.2 Speaker Output

The Lolly 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 4-4 describes the speaker output characteristics. Electrical specifications of the speaker circuit appear in Chapter 11.

------------------------------<< Table >>------------------------------

Port number:	None
Commands:	Some programs sound the speaker in response to CONTROL-G.
Initial characteristics:	Reset routine sounds the speaker.

Addresses

Hardware locations:

Location	Description
$C030	Toggle speaker (read only)

Monitor FW routines:

Location	Name	Description
$FBDD	BELL1	Sends a beep to the speaker.
$FF3A	BELL	Sends CONTROL-G to the current output device.

'Table 4-4.' Speaker Input Characteristics

<< Head 2 >>

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.

<< Head 2 >>

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 (Chapter 3).

<< Head 3 >>

BELL1

BELL1 (at address $FDBB) makes a beep through the speaker by generating a 1kHz tone in the Lolly's speaker for 0.1 second. This routine scrambles the A and X registers.

<< Head 3 >>

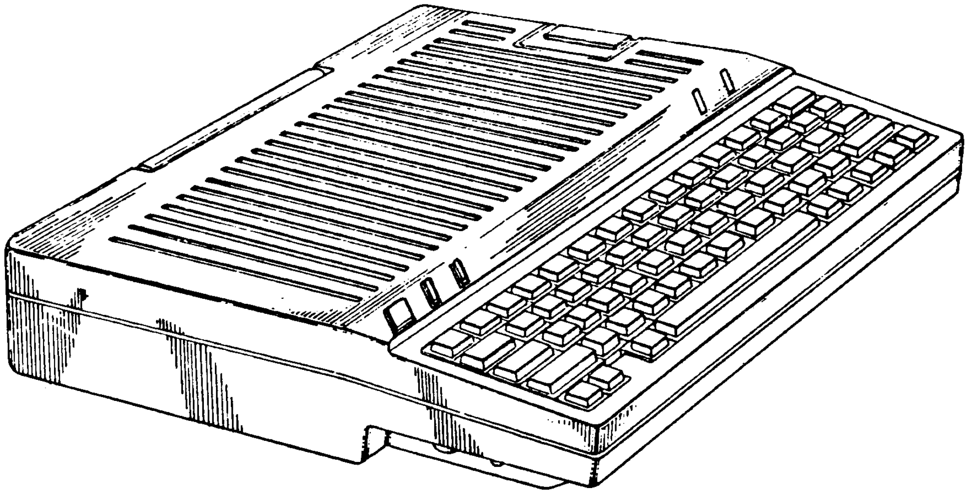
BELL

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.

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 5 • VIDEO DISPLAY OUTPUT



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 5

Video Display Output

Chapter 5

Video Display Output

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

-------------<< Gloss >>------------

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 Lolly; if it does not, you may need to attach an RF video modulator between the Lolly and the television set.

----------------------<< Gray Box >>----------------------

The Lolly 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.

-----------------------<< End Box >>----------------------

Table 5-1 is a summary of the video output port's characteristics and a guide to other information in this Chapter.

------------------------------<< Table >>------------------------------

Port number:	Output port 3
Commands:	Figure 5-2
Initial characteristics:	Figure 5-2

Note: At startup, if the 40/80-column switch is up and the program checks it, enhanced video firmware and 80-column display are turned on right away.

Addresses

Hardware locations:	See Table 5-7.
Monitor firmware routines:	See Table 5-8.
I/O firmware entry points:	See Table 5-9.

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

<< Head 1 >>

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 >>------------------------------

Display modes:	40-column text (map: Figure 5-4) 80-column text (map: Figure 5-5) Low-resolution color graphics (map: Figure 5-6) High-resolution color graphics (map: Figure 5-7) Double-high-resolution color graphics (map: Figure 5-8)
Text capacity:	24 lines by 80 columns (character positions)
Character set:	96 ASCII characters (uppercase and lowercase)
Display formats:	Normal, Inverse, Flashing, Mousetext (TM)(Table 5-2)
Low-resolution graphics:	16 colors (Table 5-3) 40 horizontal by 48 vertical (map: Figure 5-6)
High-resolution graphics:	6 colors (Table 5-5) 280 horizontal by 192 vertical (map: Figure 5-7)
Double-high-res graphics:	16 colors (Table 5-3) 560 horizontal by 192 vertical (map: Figure 5-8)

`Table 5-2.` Video Display Specifications

The video signal produced by the Lolly is NTSC-compatible composite color video. It is available at two places: the RCA-type phono jack on the back of the Lolly, 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 to connect video expansion hardware (Section 11.9.5).

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

<< Head 1 >>

5.2 Text Modes

The text characters displayed include the upper- 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.

<< Head 2 >>

5.2.1 Text Character Sets

The Lolly 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; and
- flashing, alternating between normal and inverse.

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

The alternate character set sacrifices the flashing format for a complete inverse format. With the alternate character set, the Lolly can display uppercase letters, lowercase letters, numbers, and special characters in either normal format or inverse format. It can also display Mousetext (TM), as described in Section 5.2.2.

You select the character set by means of the alternate-text soft switch, described below in the section "Display Mode Switching". Table 5-3 shows the character codes in decimal and hexadecimal for the Lolly 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).

------------------------------<< Table >>------------------------------

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	Uppercase letters Mousetext (Section 5.2.2)	Inverse
$60 - $7F	Special characters	Flashing	Lowercase letters	Inverse
$80 - $9F	Uppercase letters	Normal	Uppercase letters	Normal
$A0 - $BF	Special characters	Normal	Special characters	Normal
$C0 - $DF	Uppercase letters	Normal	Uppercase letters	Normal
$E0 - $FF	Lowercase letters	Normal	Lowercase letters	Normal

`Table 5-3.` The Display Character Sets. To identify particular characters and values, refer to Tables 4-2a and 4-2b.

<< Head 2 >>

5.2.2 Mousetext

The character-generator ROM can display 32 graphics characters called Mousetext (TM) in place of the uppercase series from $40 through $5F. These graphics are especially convenient to use with a mouse, since they can be moved around faster than bit-mapped characters. To use Mousetext characters, do the following:

- Turn on the enhanced video firmware: issue PR#3 or ESC 8.
- Turn on the Mousetext feature: PRINT CHR$(27) or pass $1B to COUT in the accumulator.
- Set inverse mode: use the INVERSE command or put $3F in location $32 and call COUT1.
- Print the uppercase letter (or other character in that group: @ [\ ] ^ or _) that corresponds to the Mousetext character you want.
- Turn off the Mousetext feature: PRINT CHR$(24) or pass $18 to COUT1 in the accumulator.

- Set normal mode: use the NORMAL command or put $FF in location $32 and call COUT1.

Here is a sample BASIC program that prints all the graphics characters:

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

<< Head 2 >>

5.2.3 40-column versus 80-column Text

The Lolly 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 5-1. 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 >>------------------------------

[Figure 5-1]

`Figure 5-1.` 40-column and 80-column Text Display (with Alternate Character Set)

Figure 5-2 illustrates the methods of switching text display modes, and the characteristics of each.

----------------------------<< Figure >>----------------------------

[Figure 5-2]

`Figure 5-2.` Text Mode
Characteristics and Switching

<< Head 1 >>

5.3 Graphics Modes

The Lolly 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 Lolly's memory.

<< Head 2 >>

5.3.1 Low-resolution Graphics

In the low-resolution graphics mode, the Lolly 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.

------------------------------<< Table >>------------------------------

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	Grey 2
3	$3	Purple	11	$B	Pink
4	$4	Dark Green	12	$C	Light Green
5	$5	Grey 1	13	$D	Yellow
6	$6	Medium Blue	14	$E	Aquamarine
7	$7	Light Blue	15	$F	White

`Table 5-4.` Low-resolution Graphics Colors. Colors may vary, depending upon the controls on the monitor or television set.

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

<< Head 2 >>

5.3.2 High-resolution Graphics

In the high-resolution graphics mode, the Lolly 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 ($2000) 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 Lolly's memory.

The Lolly high-resolution graphics display is bit-mapped: each dot on the screen corresponds to a bit in the Lolly'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 5-3. 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 >>-----------------------------

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

`Table 5-5.` High-resolution Graphics Colors. 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 Lolly 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 Lolly produces color on a TV set, see Chapter 11. For information about the way NTSC color television works, see the magazine articles listed in the bibliography.

-----------------------------<< Figure >>-----------------------------

[Figure 5-3.]

Figure 5-3. High-resolution Display Bits

<< Head 2 >>

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. There are 560 dots per line.

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 5-4 that corresponds to the window's position (Figure 5-8).

<< Head 1 >>

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.

<< Head 1 >>

5.5 Display Pages

The Lolly 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 Section 5.6). The built-in firmware I/O routines described in Chapter 3 take care of this extra addressing automatically; that is one reason to use those routines for all normal text output.

----------------------<< Gray Box >>----------------------

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

----------------------<< End Box >>----------------------

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 5-6.

----------------------<< Gray Box >>----------------------

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

----------------------<< End Box >>----------------------

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 >>------------------------------

Display mode	Display Page	Lowest Address	Highest Address
40-column Text, Low-resolution Graphics	1	$400 1024	$7FF 2047
	2	$800 2048	$BFF 3071
80-column Text	1	$400 1024	$7FF 2047
	2*	$800 2048	$BFF 3071
High-resolution Graphics	1	$2000 8192	$3FFF 16383
	2	$4000 16384	$5FFF 24575
Double-high-res Graphics	1**	$2000 8192	$3FFF 16383
	2**	$4000 16384	$5FFF 24575

Table 5-6. Video Display Page Locations. *Note: This is not supported by firmware; for instructions on how to switch pages, refer to Section 5.6. **See section 5.3.3.

<< Head 1 >>

5.6 Display Mode Switching

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

Table 5-7 shows the reserved locations for the soft switches that control the different display modes. For example, to switch from mixed-mode to full-screen graphics in an assembly-language program, you could use the instruction:

```
STA     $C052
```

----------------------<< Gray Box >>----------------------

You may not need to deal with these functions by reading and writing directly to the memory locations in this table. Many of the functions shown here are selected automatically if you use the display routines in the various high-level languages on the Lolly.

----------------------<< End Box >>----------------------

Some of the soft switches in Table 5-7 are marked read or write. Those soft switches share their locations with the keyboard data and strobe functions. To perform the function shown in the table, use the operation listed there. Soft switches that are not marked may be accessed by either a read or a write. When writing to a soft switch, it doesn't matter what value you write; the action occurs when you address the location, and the value is ignored.

------------------------ Table >>------------------------------

Name	Function	Location			Notes
		Hex	Decimal		
ALTCHARSET	Alternate char. set on	$C00F	49167	-16369	Write
	Alternate char. set off	$C00E	49166	-16370	Write
	Read ALTCHARSET switch	$C01E	49182	-16354	Read
TEXT	Text mode on	$C051	49233	-16303	
	Text mode off (graphics)	$C050	49232	-16304	
	Read TEXT switch	$C01A	49178	-16358	Read
MIXED	Mixed-mode on	$C053	49235	-16301	1
	Mixed-mode off	$C052	49234	-16302	1
	Read MIXED switch	$C01B	49179	-16357	Read
PAGE2	Page 2 on	$C055	49237	-16299	2
	Page 2 off (Page 1)	$C054	49236	-16300	2
	Read PAGE2 switch	$C01C	49180	-16356	Read
HIRES	Hi-res mode on	$C057	49239	-16297	1
	Hi-res mode off	$C056	49238	-16298	1
	Read HIRES switch	$C01D	49181	-16355	Read
80COL	80-column display on	$C00D	49165	-16371	Write
	80-column display off	$C00C	49164	-16372	Write
	Read 80COL switch	$C01F	49183	-16353	Read
80STORE	Store in auxiliary memory	$C001	49153	-16383	Write,3
	Store in main memory	$C000	49152	-16384	Write,3
	Read 80STORE switch	$C018	49176	-16360	Read
DHIRES	Double-high-res on	$C05E	49246	-16290	Read,4
	Double-high-res off	$C05F	49247	-16289	Read
	Read DHIRES switch	$C079	49279	-16257	Read

`Table 5-7.` Display Soft Switches. (1) This mode is only effective when graphics-mode switch is ON. (2) This switch has a different function when the auxiliary text page is enabled for writing. Refer to the next section. (3) This switch changes the function of the

PAGE2 switch for address[illegible] auxiliary text memory [illegible] text section describes h[illegible] do this.
(4) IOUDIS must be [illegible] (Write $C07E).

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

<< Head 1 >>

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 Box >>---------------------

`Warning`
Never call any firmware with 80STORE and PAGE2 both on. If you do, the firmware will not function properly. As a general rule, always leave PAGE2 off unless you shut off interrupts.

-----------------------<< End Box >>-----------------------

The display memory maps are shown in Figures 5-4 through 5-8. All of the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes ($80 hexadecimal). For a full description of the way the Lolly 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).

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

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 5-8).

------------------------------<< Figure >>------------------------------

[Figure 5-4]

`Figure 5-4.` Map of 40-column Text Display

------------------------------<< Figure >>------------------------------

[Figure 5-5]

`Figure 5-5.` Map of 80-column Text Display

-----------------------------<< Figure >>-----------------------------

[Figure 5-6]

`Figure 5-6.` Map of Low-resolution Graphics Display

-----------------------------<< Figure >>-----------------------------

[Figure 5-7]

`Figure 5-7.` Map of High-resolution Graphics Display

----------------------------<< Figure >>----------------------------

[Figure 5-8]

`Figure 5-8.` Map of Double-high-resolution Graphics Display

<< Head 2 >>

5.8 Monitor Firmware Support

Table 5-8 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 >>-----------------------------

Location	Name	Description
$FC9C	CLREOL	Clears to end of line.
$FC9E	CLEOLZ	Clears to end of line using BASL.
$FC42	CLREOP	Clears to bottom of window.
$F832	CLRSCR	Clears the low-resolution screen.
$FF836	CLRTOP	Clears top 40 lines of low-res 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 screen and puts cursor in upper left corner of screen.
$F800	PLOT	Plots a single 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 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 block on the screen.
$F864	SETCOL	Sets the color for plotting in low-res.
$F828	VLINE	Draws a vertical line of blocks.

`Table 5-8.` Monitor Firmware Routines

<< Head 2 >>

5.9 I/O Firmware Support

Lolly 80-column 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 5-9.

-----------------------------<< 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)

Table 5-9. I/O Firmware Protocol Table

<< Head 3 >>
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.

<< Head 3 >>
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.

<< Head 3 >>
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 then performs the CONTROL-| function (see Table 5-10).

- carries out control functions as shown in Table 5-10.

------------------------------<< Table >>------------------------------

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, then does CONTROL-J.
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.)
Q or q	11	Switches to 40-column display.
R or r	12	Switches to 80-column display.
U or u	15	Turns off 80-column firmware. (Not available to Pascal.)
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, also 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.

Table 3-x. Pascal Video Control Functions

When PWRITE has completed this, it

- turns the cursor back on (if it was not intentionally turned off)

- puts a zero in the X register (IORESULT = GOOD) and returns to the calling program.

<< Head 3 >>

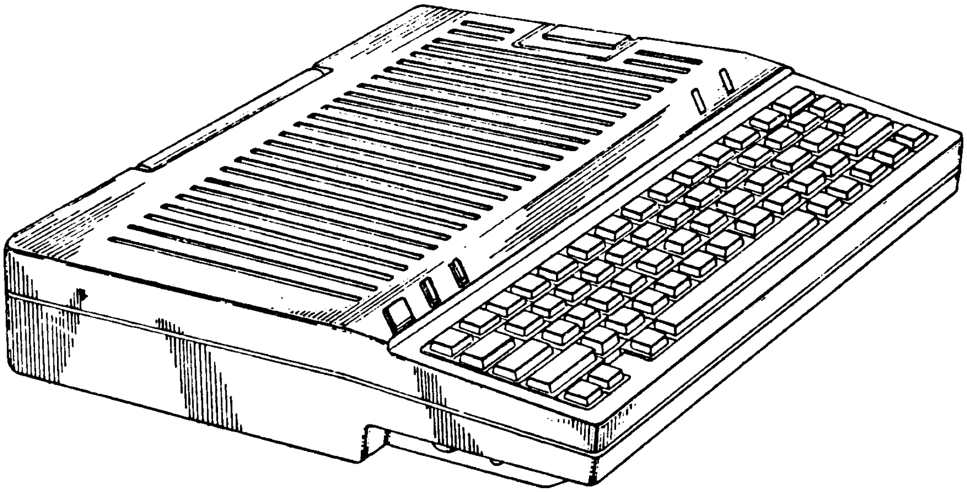
PSTATUS

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).

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 6 • DISK I/O



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 6

Disk Input and Output

Chapter 6

Disk Input and Output

The Lolly 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 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, \Chech Disk Drive\ on the display screen.

------------------------------<< Table >>------------------------------

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 6 CONTROL-K or 6 CONTROL-P (if there is no operating system in RAM)
Initial characteristics:	All resets except CONTROL-RESET with a valid reset vector eventually pass control to the built-in disk drive.

Addresses

Hardware locations:

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.

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

<< Head 1 >>

6.1 Startup

A power-on startup, an OPEN-APPLE-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.

<< Head 1 >>

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 Lolly with a diagnostic program in the event that the built-in drive does not work.

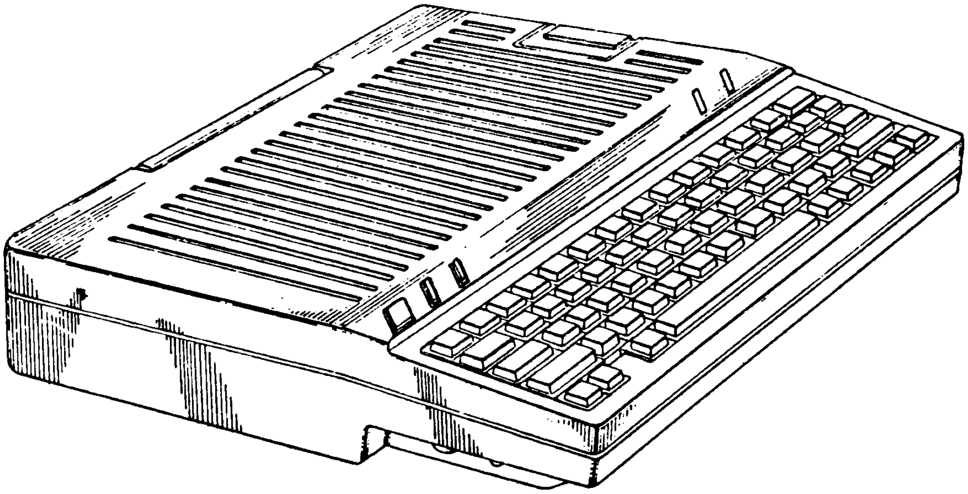
To restart using the external drive, issue a PR#7 command from the keyboard with a ProDOS disk in the external drive.

-------------<< Gloss >>------------
Remember that external drive startup works with ProDOS-based and many RAM-based games, but not with Pascal 1.1 or 1.0, or with DOS.

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 7 • SERIAL PORT 1



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 7

Serial Port 1

Chapter 7

Serial I/O Port 1

Serial port 1 is the first of two serial I/O ports available on the Lolly. 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 Universal Utilities Disk. If you need to change port characteristics from a program, read Section 7.5 for the methods to use.

----------------------<< Warning Box >>---------------------

`Warning`
Although the Lolly 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.

------------------------<< End Box >>-----------------------

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.

-------------<< Gloss >>------------
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 >>-----------------------------

Port number:	Serial port 1
Commands:	Keyboard commands: PR#1 BASIC commands: PR#1 Monitor command: 1 CONTROL-P (if there is an operating system in RAM, follow this command with 3 CONTROL-P) All other commands: Table 7-2
Initial characteristics:	Table 7-3
Addresses	
Hardware locations:	Table 7-4
Monitor FW routines:	None
I/O FW entry points:	Table 7-5
Use of screen holes:	Table 7-6
Use of other pages:	None

`Table 7-1.` Serial Port 1
Characteristics

<< Head 1 >>
7.1 Using Serial Port 1

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

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

------------<< Gloss >>-----------

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

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.

----------------------<< Gray Box >>----------------------

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

----------------------<< End Box >>----------------------

------------------------------<< Table >>------------------------------

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td>nnn</td><td>Set new line width of nnn (from 1 through 255).</td></tr>
<tr><td>nnB</td><td>Set baud rate to value corresponding to nn:
<table>
<tr><th>nn</th><th>Rate</th><th>nn</th><th>Rate</th></tr>
<tr><td>1</td><td>50</td><td>9</td><td>1800</td></tr>
<tr><td>2</td><td>75</td><td>10</td><td>2400</td></tr>
<tr><td>3</td><td>110 (109.92)</td><td>11</td><td>3600</td></tr>
<tr><td>4</td><td>135 (134.58)</td><td>12</td><td>4800</td></tr>
<tr><td>5</td><td>150</td><td>13</td><td>7200</td></tr>
<tr><td>6</td><td>300</td><td>14</td><td>9600</td></tr>
<tr><td>7</td><td>600</td><td>15</td><td>19200</td></tr>
<tr><td>8</td><td>1200</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>nD</td><td>Set data format to values corresponding to n:
<table>
<tr><th>n</th><th>Data bits</th><th>Stop bits</th></tr>
<tr><td>0</td><td>8</td><td>1</td></tr>
<tr><td>1</td><td>7</td><td>1</td></tr>
<tr><td>2</td><td>6</td><td>1</td></tr>
<tr><td>3</td><td>5</td><td>1</td></tr>
<tr><td>4</td><td>8</td><td>2</td></tr>
<tr><td>5</td><td>7</td><td>2</td></tr>
<tr><td>6</td><td>6</td><td>2</td></tr>
<tr><td>7</td><td>5</td><td>2</td></tr>
</table></td></tr>
<tr><td>I</td><td>Echo printer output on the screen.</td></tr>
<tr><td>K</td><td>Disable automatic line feed after carriage return.</td></tr>
<tr><td>L</td><td>Generate line feed after carriage return.</td></tr>
<tr><td>nnnN</td><td>Set line width to nnn (from 1 through 255); do not echo printer output on the screen.</td></tr>
</table>

	Note: 0N is ignored; to disable automatic generation of carriage return, use Z command.
nP	Set parity corresponding to n:

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

R	Reset port 1 ACIA (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.

`Table 7-2.` Printer Port Commands

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 type CONTROL-I CONTROL-V. (CONTROL-V and CONTROL-W are the recommended substitute control characters.) To change the command character back again, type CONTROL-V CONTROL-I.

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

Examples of valid commands and command sequences:

CONTROL-I I	echo output to the display screen
CONTROL-I K CONTROL-I 72N	set line width 72, disable LF & echo
CONTROL-I CONTROL-V RETURN CONTROL-V (command) RETURN	change control character to CONTROL-V (for example, so you can send CONTROL-I as part of a character stream)

---------------------<< Warning Box >>--------------------

`Warning`
Once the printer has begun, do not issue PR#0, PR#3, ESC 4 or ESC 8 until it has finished printing. Any one of these commands turns off the printer port.

-----------------------<< End Box >>----------------------

<< Head 1 >>

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 >>-----------------------------

* 9600 baud

* 8 data bits, no parity bits, 2 stop bits

* 80-column line width; no echo to display screen

* firmware supplies linefeed after carriage return

* command character is set to CONTROL-I (see below)

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

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.

----------------------<< Gray Box >>----------------------

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

-----------------------<< End Box >>----------------------

<< Head 1 >>

7.3 Hardware Page Locations

Table 7-4 lists the addresses and bit assignments of serial port 1's hardware registers on page $C0. The registers are internal to a 6551 ACIA; their bit assignments are described in Chapter 11.

-------------<< Gloss >>------------

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 >>-----------------------------

Location	Description
$C090	Reserved
$C091	Reserved
$C092	Reserved
$C093	Reserved
$C094	Reserved
$C095	Reserved
$C096	Reserved
$C097	Reserved
$C098	ACIA transmit/receive data register
$C099	ACIA status register
$C09A	ACIA command register
$C09B	ACIA control register
$C09C	Reserved
$C09D	Reserved
$C09E	Reserved
$C09F	Reserved

`Table 7-4.` Serial Port 1 Hardware Page Locations

<< Head 1 >>

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.x describes how to use this protocol.

--------------------------------<< Table >>------------------------------

Address	Value	Description
$C105	$38	Pascal ID byte (opcode SEC)
$C107	$18	Pascal ID byte (opcode CLC)
$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

`Table 7-5.` Port 1 I/O Firmware Protocol

<< Head 1 >>

7.5 Screen Hole Locations

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.

-------------<< Gloss >>------------

The ACIA register bits are defined in Chapter 11.

-----------------------<< End Box >>----------------------

----------------------------<< Table >>----------------------------

Location	Description
Auxiliary memory screen holes (firmware loads at power-up reset)	
$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
0	1 = communication port; 0 = serial printer port

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

bit	interpretation
7-0	printer width (0 = do not insert CR)

Location	Description
Main memory screen holes	
$479	Reserved
$4F9	Reserved
$579	Printer width (1 - 255; 0 = disable formatting)
$5F9	Temporary storage location
$679	Bit 7 = 1 if and only if the firmware is currently 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 linefeed after carriage return.
$7F9	Current printer column

Table 7-6. Serial Port 1 Screen Hole Locations

<< Head 1 >>

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 serial port firmware moves the predefined set of port characteristics listed in Table 7-2 from ROM into the auxiliary memory screen holes listed in Table 7-6.

- If you specify new characteristics using the Universal Utilities Disk, the UUD software changes the values in the auxiliary memory screen holes.

- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either OPEN-APPLE CONTROL-RESET or a simple CONTROL-RESET. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up.

- 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 comands listed in Table 7-2 to change them.

----------------------------<< Figure >>----------------------------

[Figure 7-1]

<< Head 2 >>

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 a zero, the receiver presumes it is a start bit--that is, the bit that designates the beginning of a word (byte) 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. After one or two bit-intervals, the next start bit begins transfer of the next word 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. Even parity is similar. 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 plus 7 data plus 1 parity bit plus 1 stop bit, then the actual transfer rate is about 30 characters per second.

--------------------------<< Figure >>-----------------------------

[Figure 7-2]

`Figure 7-2.` Data Format

<< Head 2 >>

7.6.2 Carriage Return and Linefeed

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 linefeed. Most typewriters perform a linefeed automatically after a carriage return, and so the two seem to be one--but they are not.

Carriage return and linefeed are separate ASCII codes. Carriage return is sometimes denoted CR; it is ASCII code 13 ($0D). Linefeed, sometimes denoted LF, is ASCII code 10 ($0A). The DOWN-ARROW key on the Lolly keyboard generates a LF.

Some printers can supply a linefeed automatically after detecting a carriage return; others cannot. If the printer does not supply LF after CR 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 Lolly 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 OK, then there is probably not enough time for the print mechanism to return to the left margin in response to CR. However, the lLolly cfirmware cannot supply a delay after CR, so you have to use a lower baud rate with such a printer.

<< Head 2 >>

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 Lolly 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 command.

<< Head 2 >>

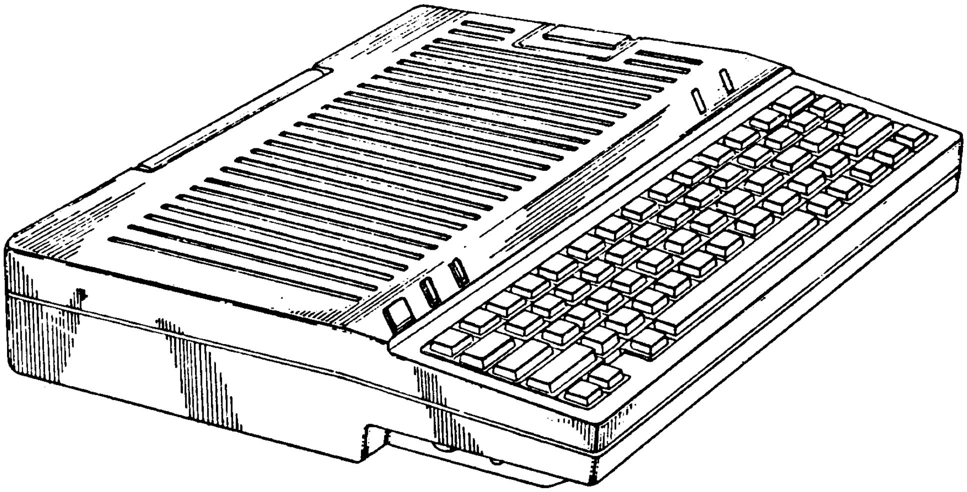
7.6.4 Displaying Output on the Screen

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.

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 8 • SERIAL PORT 2



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 8

Serial Port 2

Chapter 8

Serial I/O Port 2

Serial port 2 is the second of two serial I/O ports available on the Lolly. It is intended primarily as a communication port for modems. You can change it to an serial printer port (like port 1) using the Universal Utilities Disk. If you need to change port characteristics from a program, read Section 8.5 for methods to use.

----------------------<< Warning Box >>----------------------

`Warning`
Although the Lolly 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.

------------------------<< End Box >>------------------------

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

-------------<< Gloss >>------------
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.

------------------------<< End Box >>------------------------

----------------------------<< Table >>----------------------------

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 to output.
	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	
Addresses		
Hardware locations:	Table 8-4	
Monitor FW routines:	None	
I/O FW 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 an output buffer.	

`Table 8-1.` Serial Port 2 Characteristics

<< Head 1 >>

8.1 Using Serial Port 2

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

----------------------<< Gray Box >>----------------------

Note: 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).

-----------------------<< End Box >>----------------------

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

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

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

----------------------<< Gray Box >>----------------------

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

-----------------------<< End Box >>----------------------

------------------------------<< Table >>------------------------------

Command	Description
nnn	Set new line width of nnn (from 1 through 255).
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		

Command	Description
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

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 is ignored; to disable automatic generation of carriage return, use Z command.
nP	Set parity corresponding to n:

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

Command	Description
Q	Quit terminal mode.

R	Reset port 2 ACIA (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 Lolly 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.

`Table 8-2.` Modem Port Commands

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 type CONTROL-A CONTROL-V. (CONTROL-V and CONTROL-W are the recommended substitute control characters.) To change the command character back again, type CONTROL-V CONTROL-A.

Do not use CONTROL-B, -C, -H, -I, -J, -L, -M or -Y: Lolly firmware may intercept these control characters, causing unpredictable results.

Examples of valid commands and command sequences:

CONTROL-A I	enable echo to screen
CONTROL-A B	send a BREAK character to remote device
CONTROL-A CONTROL-V CONTROL-V (command)	change control character to CONTROL-V (for example, so you can send CONTROL-A as part of a character stream)

----------------------<< Warning Box >>---------------------

`Warning`
Once the transmission has begun, do not issue PR#0, PR#3, ESC 4 or ESC 8 until the transmission is over. Any one of these commands turns off the communication port.

------------------------<< End Box >>-----------------------

<< Head 1 >>

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 >>------------------------------

* 300 baud

* 7 data bits, no parity bits, 1 stop bit

* firmware does not supply linefeed 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

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

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 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.

<< Head 1 >>

8.3 Hardware Page Locations

Table 8-4 lists the addresses of serial port 2's hardware registers on page $C0. The registers are internal to a 6551 ACIA; their bit assignments are described in Chapter 11.

-------------<< Gloss >>------------

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.

-----------------------<< End Box >>----------------------

------------------------------<< Table >>-----------------------------

Location	Description
$C0A0	Reserved
$C0A1	Reserved
$C0A2	Reserved
$C0A3	Reserved
$C0A4	Reserved
$C0A5	Reserved
$C0A6	Reserved
$C0A7	Reserved
$C0A8	ACIA transmit/receive data register
$C0A9	ACIA status register
$C0AA	ACIA command register
$C0AB	ACIA control register
$C0AC	Reserved
$C0AD	Reserved
$C0AE	Reserved
$C0AF	Reserved

`Table 8-4.` Serial Port 2 Hardware Page Locations

<< Head 1 >>

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.x describes how to use this protocol.

------------------------------<< Table >>------------------------------

Address	Value	Description
$C205	$38	Pascal ID byte (opcode SEC)
$C207	$18	Pascal ID byte (opcode CLC)
$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

`Table 8-5.` Port 2 I/O Firmware Protocol

<< Head 1 >>

8.5 Screen Hole Locations

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.

-------------<< Gloss >>------------
The ACIA register bits are defined in Chapter 11.

------------------------------<< Table >>------------------------------

Location	Description
Auxiliary memory screen holes (firmware loads values shown at power up)	
$47C	$16 (ACIA control reg: 7 data + 1 stop bit, 300 baud)
$47D	$0B (ACIA command reg: no parity)
$47E	$01 (flags: no echo, an auto LF after CR, comm port)

bit	interpretation
7	echo output on display (0 = no echo)
6	generate LF after CR (0 = no LF)
5-1	always = 0
0	1 = communication port; 0 = serial printer port

Location	Description
$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

Location	Description
$47A	Reserved
$4FA	Reserved
$57A	Line width (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 linefeed after carriage return.
$7FA	Current column

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

<< Head 1 >>

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 serial port 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 Universal Utilities Disk, the UUD software changes the values in the auxiliary memory screen holes.

- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either OPEN-APPLE CONTROL-RESET or a simple CONTROL-RESET. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up.

- IN#2 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main memory screen holes.

- A program can change values in the main memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location $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 >>-----------------------------

[Figure 8-1]

<< Head 2 >>

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 Lolly and its firmware, with the baud rate, data format and other characteristics you have selected
- the cable from the Lolly 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 >>----------------------------

[Figure 8-2]

`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 Lolly.

<< Head 2 >>

8.6.2 Carriage Return and Linefeed

If you are communicating with a computer or terminal, carriage return and linefeed 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.

<< Head 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. Figure 8-3 shows the patterns of information flow you can select. The following subsections tell you how to use them.

----------------------------<< Figure >>----------------------------

[Figure 8-3]

`Figure 8-3.` Serial Port 2
Information Flow

<< Head 3 >>
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. A computer information service, for example, typically operates half duplex. This means, among other things, that the service does not echo what the Lolly sends it back to the Lolly. For this kind of operation (Figure 8-4) the Lolly should echo its output to the display; otherwise, when you type information, it will seem as though nothing was typed.

----------------------------<< Figure >>----------------------------

[Figure 8-4]

`Figure 8-4.` Half Duplex Operation

<< Head 3 >>

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.

If your Lolly is the terminal in full duplex operation, do not echo output to the screen (Figure 8-5). If the Lolly does echo output to the screen in this case, everything you type will appear twice on the screen: once from the Lolly, and once from the host computer.

-----------------------------<< Figure >>-----------------------------

[Figure 8-5]

`Figure 8-5.` Full Duplex—Lolly Is Terminal

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 Lolly output as echoed by the host.

If your Lolly is the host computer communicating with a terminal on the other end, full duplex operation requires the Lolly to echo its input to output for the benefit of the terminal (Figure 8-6). However, if the Lolly 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 Lolly, requiring a RESET to get out.

-----------------------------<< Figure >>------------------------------

[Figure 8-6]

`Figure 8-6.` Full Duplex--Lolly Is
Host

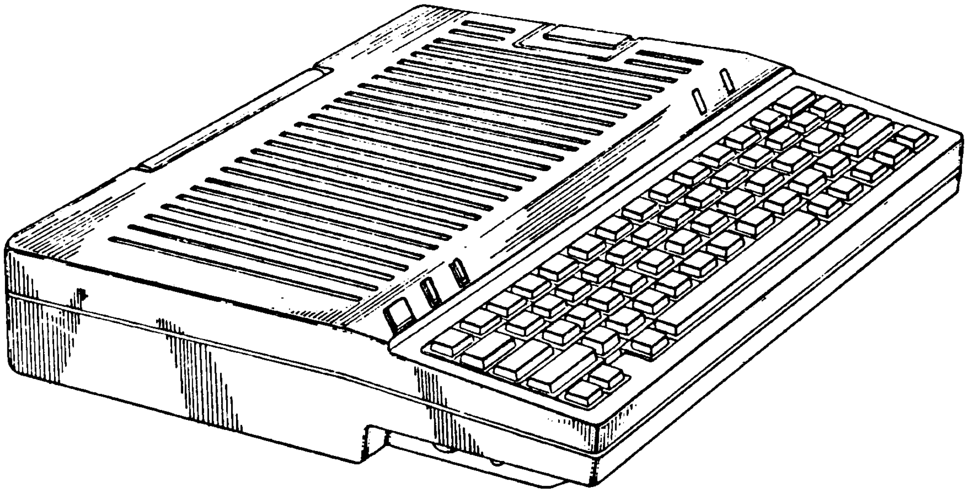
<< Head 3 >>
Terminal Mode

Terminal mode makes the Lolly act like what is known as a dumb terminal--one that just sends and receives information, but does not process it. This is advantageous for communication with information services, where control characters might cause trouble if the Lolly attempts to act on them. You can use terminal mode with half duplex or full duplex operation.

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 9 • MOUSE & GAME INPUT



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 9

Mouse and Game Input

Chapter 9

Mouse and Game Input

This chapter describes the mouse port and hand control (game) input capabilities of the Lolly. 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.

-------------<< Gloss >>------------

There is a section in this chapter titled "Using the Mouse as a Hand Control" that clarifies this difference.

<< Head 1 >>

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 Box >>---------------------

`Warning`

If you want to insure 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.

------------------------<< End Box >>-----------------------

--------------------------------<< Table >>-------------------------------

Port number	Mouse input port 4
Keyboard commands	Turn on mouse: PR#4 1 RETURN Turn off mouse interrupts: PR#4 0 RETURN Read mouse: IN#4 RETURN
BASIC commands	Turn on mouse: PRINT"PR#4"PRINT CHR$(1) Turn off mouse interrupts: PRINT"PR#4" PRINT CHR$(0) Turn on graphics character set: Chapter 5
Initial characteristics	After a reset, all mouse interrupts are off, and the rising edge of X0 and Y0 are selected for interrupts.
Addresses	
Hardware locations:	Table 9-2
Monitor FW routines:	None
I/O FW entry points:	Table 9-3 (Pascal) Table 9-4 (BASIC or assembly language)
Use of screen holes:	Table 9-5

`Table 9-1.` Mouse Input Port Characteristics

<< Head 2 >>

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 11-18.

<< Head 2 >>

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.

<< Head 3 >>
Transparent Mode

In this mode, the mouse behaves like a polled mouse such as the mouse on an Apple IIe. In reality, however, an interrupt routine in the Lolly firmware updates mouse position counters each time the mouse is moved, then returns control to the main program task.

<< Head 3 >>
Movement Interrupt Mode

On the Lolly, 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 either direction of either axis. 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 call READMOUSE (Table 9-4) to get mouse status and its current X-Y position. The routine can also change the mouse mode and position if desired.

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. Any computer's mouse cursor will go nuts if the user moves the mouse extremely fast.

-------------<< Gloss >>------------
Chapter 5 contains recommendations
for using Mousetext (TM) characters
with a mouse.

<< Head 3 >>
Button Interrupt Mode

The Lolly 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.

<< Head 3 >>

Movement/Button 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.

<< Head 3 >>

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.

<< Head 2 >>

9.1.3 Mouse Hardware 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.

----------------------<< Warning Box >>---------------------

`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 II of this manual. Using the built-in firmware is much easier and guarantees compatibility with all other

Apple II series computers.

-----------------------<< End Box >>----------------------

------------------------------<< Table >>------------------------------

Decimal		Hex	Interpretation
49240	-16296	$C058*	Mask X0 and Y0 interrupts (off)
49241	-16295	$C059*	Enable X0 and Y0 interrupts (on)
49216	-16320	$C040	Read status of X0/Y0 interrupt mask
49242	-16294	$C05A*	Mask VBLINT interrupts (off)
49243	-16293	$C05B*	Enable VBLINT interrupts (on)
49217	-16319	$C041	Read status of VBLINT mask
49177	-16359	$C019	Reading resets VBLINT
49264	-16272	$C070	Reset VBLINT interrupt
49244	-16292	$C05C*	Select rising edge of X0 for interrupt (off)
49245	-16291	$C05D*	Select falling edge of X0 for interrupt (on)
49218	-16318	$C042	Read status of X0 edge selector
49246	-16290	$C05E*	Select rising edge of Y0 for interrupt (off)
49247	-16289	$C05F*	Select falling edge of Y0 for interrupt (on)
49219	-16317	$C043	Read status of Y0 edge selector
49251	-16285	$C063	Read mouse button on D7
49254	-16282	$C066	Read X1 mouse direction on D7
49255	-16281	$C067	Read Y1 mouse direction on D7

* These effects take place only if IODIS is clear; to clear it, write to $C07F then set it again by writing to $C07E. The firmware leaves IOUDIS set when it is not using it.

Table 9-2. Mouse Hardware Page Locations

<< Head 2 >>

9.1.4 I/O Firmware Support

The Lolly 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), set interrupts, and then do a JSR to the firmware routine called SETMOUSE (Table 9-4). 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 9-4), which stores the information in the port 4 screen holes (Table 9-5).

<< Head 3 >>

Pascal Support

Table 9-3 lists the locations and values of the I/O Firmware protocol table that Pascal 1.1 and 1.2 use. This standardized protocol is available to any application program.

------------------------------<< Table >>------------------------------

Address	Value	Description
$C405	$38	Pascal ID byte (opcode SEC)
$C407	$18	Pascal ID byte (opcode CLC)
$C40B	$01	Generic signature byte of firmware cards
$C40C	$20	2=XY pointing device; 0=identification code
$C40D	$ii	$C4ii is entry point of initialization routine (PINIT)
$C40E	$rr	$C4rr is entry point of read routine (PREAD)
$C40F	$ww	$C4ww is entry point of write routine (PWRITE)
$C410	$ss	$C4ss is entry point of the status routine (PSTATUS)
$C411	0	Optional routines follow

`Table 9-3.` Mouse Port I/O Firmware Protocol

<< Head 3 >>

BASIC and Assembly Language Support

In BASIC, before you can get input from the mouse, turn it on by printing PR#4 and then CHR$(1). The first input statement after that (use three labels) 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 abd +1023. These are called the clamping limits of the mouse.

The sign preceding the status digits is normally positive; it becomes negative when a key on the keyboard is pressed.

The first digit, s, of the status is normally 0; it becomes a 1 if either of the coordinates goes out of range. The second digit, t, of the status is 1 if the mouse button is not depressed, 2 if the button is depressed, and 0 if the button is still depressed since the last input.

To disable the mouse, type PR#4, then type 0, then press RETURN.

Table 9-4 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 except as noted in Table 9-4.

------------------------------<< Table >>------------------------------

Loc.	Offset for	Description
$C412	SETMOUSE	Sets the mouse mode to the value in the accumulator. Input: A register contains mode 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: don'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.
$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. Output: Carry bit = 0
$C415	CLEARMOUSE	Sets the mouse position screen holes to 0; 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 registers contain new X and Y positions Output: Carry bit = 0
$C417	CLAMPMOUSE	Sets new clamping boundaries (see Table 9-5). Does not update mouse position screen holes; use READMOUSE to do that. Input: A register = 0 means set new X limits A register = 1 means set new Y limits Output: Carry bit = 0
$C418	HOMEMOUSE	Sets the internal mouse position to the upper left corner of the clamping window. Does not update mouse position screen holes; use READMOUSE to do that.
$C419	INITMOUSE	Sets startup internal values; does not update mouse position screen holes. Output: Carry bit = 0

`Table 9-4.` Mouse BASIC and Assembly Language Firmware Routines

<< Head 2 >>

9.1.5 Screen Holes

Table 9-5 lists the screen holes that the mouse firmware uses. Note that some of the holes are "borrowed" from other port assignments. Also, the auxiliary-page counterparts of these addresses are reserved for startup values.

------------------------------<< Table >>------------------------------

Location	Description
Scratch area	
$478	Low byte of clamping minimum
$4F8	High byte of clamping minimum
$578	Low byte of clamping maximum
$5F8	High byte of clamping maximum
Port 4 screen holes	
$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	unused
3	VBLINT active
2	VBLINT interrupt on button
1	VBLINT interrupt on movement
0	mouse active

Port 5 screen holes: mouse bounds	
$47D	Low byte of minimum X value
$4FD	Low byte of minimum Y value
$57D	High byte of minimum X value

$5FD	High byte of minimum Y value
$67D	Low byte of maximum X value
$6FD	Low byte of maximum Y value
$77D	High byte of maximum X value
$7FD	High byte of maximum Y value

Table 9-5. Mouse Peripheral-card RAM locations

In transparent mode, the firmware updates the X and Y locations on an interrupt basis in a manner almost transparent to the currently executing program.

In movement interrupt mode, the firmware arms the video vertical blanking signal (with a period of 1/60th of a second) to interrupt whenever the mouse is moved at least a count of 1 in either coordinate. The next time the VBL signal occurs, program control branches to the location specified in $3F2 to acknowledge the interrupt and update the coordinates. (The interrupt handler must call the routine at $C410 to get status and current coordinates.) The handler may also reset the mode and position if desired.

In button-interrupt mode, the firmware arms the VBL interrupt when button status changes, the same way as in movement-interrupt mode.

In movement-button interrupt mode, the firmware arms the VBL interrupt whenever a change to coordinates or button status occurs. It allows for maximum responsiveness, without requiring the application program to do continual polling itself.

<< Head 2 >>

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 and use the PDL function to read from the mouse by doing this:

- Start up the system with the BASIC program that uses paddles.
- Type PR#4 RETURN to turn on the mouse.
- Type CONTROL-A RETURN to initialize the mouse.
- Type PR#0 RETURN to restore output to the screen.
- Type the name of the program you want to run.

Play the game using the mouse instead of the paddles.

<< Head 1 >>

9.2 Game Input

The Lolly supports game paddles, joysticks and other hand controls connected to the DB-9 connector on its back panel. Table 9-5 is a summary of game input characteristics.

------------------------------<< Table >>------------------------------

Port number:	None
Commands:	None
Initial characteristics:	Game inputs cannot be disabled.

Addresses

Hardware locations:

Location	Description
$C061	Switch input 0 and OPEN-APPLE key.
$C062	Switch input 1 and SOLID-APPLE key.
$C063	Switch input 0 (mouse button only; to distinguish it from paddle button)
$C064	Analog input (paddle) 0.
$C065	Analog input (paddle) 1.
$C070	Trigger paddle timer.

Monitor firmware routines:

Location	Name	Description
$FB1E	PREAD	Read a paddle position.

I/O FW entry points: None

Use of screen holes: None

'Table 9-6.' Game Input Characteristics

<< Head 2 >>

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 Lolly: two one-bit inputs, or switches, and two analog inputs. You can access all of these signals from your programs.

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. Complete electrical specifications of these inputs are given in Chapter 11; Table 11-18 shows the connector pin numbers.

<< Head 3 >>

Switch Inputs (SW0 and SW1)

The two one-bit inputs can be connected to the output of another electronic device that meet 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 the OPEN-APPLE and SOLID-APPLE keys 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 OPEN-APPLE keypress.

<< Head 3 >>

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 Chapter 7 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.

<< Head 2 >>

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.

-------------<< Gloss >>------------

These input signals cannot be reset and read in less than 3 milliseconds.

<< Head 3 >>

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.

-----------------------<< Gray Box >>----------------------

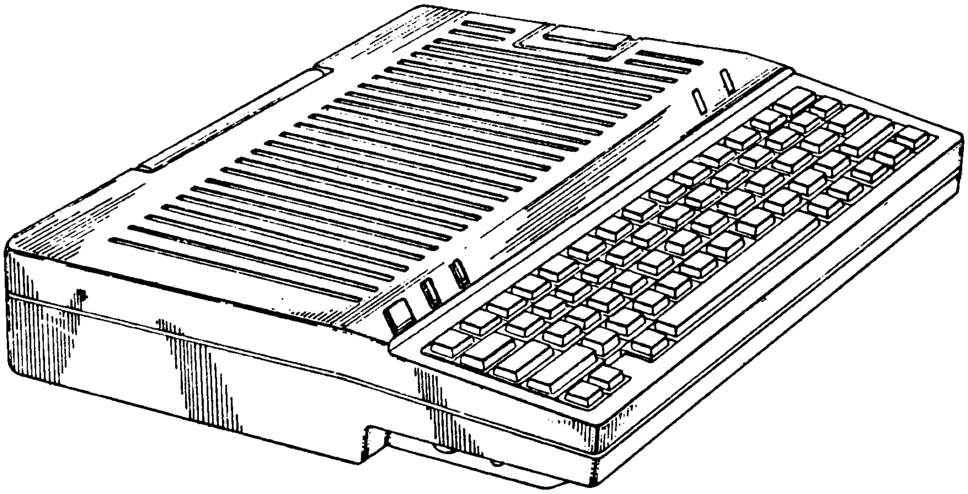
Note: if the hand control number you furnish in the X register does not equal 0 or 1, strange things may happen.

------------------------<< End Box >>-----------------------

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 10 • USING THE MONITOR



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 10

Using The Monitor

Chapter 10

Using The Monitor

The System Monitor is a set of subroutines in the Lolly 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.

The disk operating systems (DOS and ProDOS, Appendix D) 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
- change the contents of any location
- write programs in machine language to be executed directly by the Lolly's microprocessor
- move and compare blocks of memory
- invoke other programs from the Monitor

<< Head 1 >>

10.1 Invoking the Monitor

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 blinking cursor.

-------------<< Gloss >>------------
The positive and negative decimal equivalents of Monitor locations are listed in Appendix C. In addition, Appendix H contains conversion tables from one numbering system to another. Appendix E gives further details on how to use Lolly firmware from BASIC programs.

To use the Monitor, you type commands at the keyboard. When you have finished using the Monitor, you return to the BASIC language you were previously using by pressing CONTROL-RESET, by typing CONTROL-C and pressing RETURN, or by typing 3D0G, which executes the resident program--usually Applesoft--whose address is stored in a jump instruction at location $3D0.

-----------------------<< Gray Box >>----------------------

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 (such as using a discontinued cassette command like SAVE, LOAD or SHLOAD) will return control to BASIC rather than to the Monitor.

-----------------------<< End Box >>----------------------

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).

<< Head 1 >>

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. (See the "Summary of Monitor Commands" at the end of the chapter.)

This chapter contains many examples of the use of Monitor commands. In the examples, the commands and values you type are shown in a normal typeface and the responses of the Monitor are in a computer typeface. Of course, when you perform the examples, all of the characters that appear on the display screen will be in the same typeface. Some of the data values displayed by your Lolly may differ from the values printed in these examples, because they are variables stored in programmable memory.

<< Head 1 >>

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 Box >>---------------------

`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.

------------------------<< End Box >>-----------------------

<< Head 2 >>

103.1 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- 20\
```

```
\*\33

\0033- AA\
\*\
```

Each time the Monitor displays the value stored at a location, it saves the address of that location as the last opened location and as the next changeable location.

<< Head 2 >>

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 than the last opened location the address after the period is.

```
\*\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\
\*\
```

A memory dump includes several different items of information. The first line in the dump begins with the address of the location following the last opened location; all other lines begin with addresses that end alternately in zeros and eights, and there are never more than eight data values displayed on a single line in a memory dump.

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 04 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 the RETURN key 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 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\
\*\
```

<< Head 2 >>

10.3.3 Changing Memory Contents

The previous section showed you how to display the values stored in the Lolly's memory; this section shows you how to change those values. You can change any location in RAM (programmable memory); you can change the characteristics and treatment of output devices by changing the contents in locations assigned to them; and finally, you can change soft switch settings by referencing the switches' set and reset addresses.

----------------------<< Warning Box >>---------------------

`Warning`
Use these commands carefully. If you change the zero-page locations used by the interpreter or operating system in use (Appendix B), you may lose programs or data stored in memory.

------------------------<< End Box >>-----------------------

<< Head 3 >>

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- 00\
\*\: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 disappears, never to be seen again. The new value will remain until you replace it with another value.

<< Head 3 >>

Changing Consecutive Locations

You don't have to type a separate command with an address, a colon, a value, and 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, and ending with RETURN. The Monitor will duly 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\
\*\
```

<< Head 2 >>
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 typing the letter M. The format of the complete MOVE command looks like this:

```
{destination} < {start} . {end} M
```

When you type the actual command, the words in curly braces should be replaced by hexadecimal addresses, and the braces and spaces should be omitted. 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 the letter 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<300.30CM

\*\0.C

\0000- A9 8D 20 ED FD A9 45 20\
\0008- DA FD 4C 00 03\
\*\310<8.AM

\*\310.312
```

```
\0310- DA FD 4C\
\*\2<7.9M

\*\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.

If the destination address of the MOVE command is inside the source range of addresses, then strange (and sometimes wonderful) 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. See the section "Special Tricks with the Monitor" for an interesting application of this feature.

<< Head 2 >>

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.DM
```

```
\*\300<0.DV

\*\6:E4

\*\300<0.DV

\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 the section "Special Tricks with the Monitor".

<< Head 1 >>

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.

<< Head 2 >>

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.

<< Head 2 >>

10.4.2 Examining Registers

Typing CONTROL-E and pressing 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\
\*\
```

<< Head 1 >>

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.

<< Head 2 >>

10.5.1 Display Inverse and Normal

You can control the setting of the inverse-normal mask location used by the COUT subroutine (described in Chapter 3) from the Monitor so that all of 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\
\*\
```

<< Head 2 >>

10.5.2 Back to BASIC

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

```
3D0G
```

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

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.

----------------------<< Gray Box >>----------------------

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 the section "Machine-language Programs".

-----------------------<< End Box >>----------------------

<< Head 2 >>

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. Chapter 3 lists the Lolly port numbers available. The format of the command is

```
{port number} CONTROL-P
```

A CONTROL-P command to port number 0 will switch the stream of output characters back to the Lolly'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 Lolly's normal input device, the keyboard. The format for the command is:

```
{port number} CONTROL-K
```

Issuing 0 CONTROL-K directs the Monitor to accept input from the Lolly's built-in keyboard.

The CONTROL-P and CONTROL-K commands are the exact equivalents of the BASIC commands PR# and IN#. For more information on the way those commands work, refer to the section "The Standard I/O Links" in Chapter 3.

<< Head 2 >>

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, of course):

```
{value} + {value} @RETURN@
{value} - {value} @RETURN@
```

The Lolly performs the arithmetic and displays the result, as shown in these examples:

```
\*\20+13
\=33\
\*\4A-C
\=3E\
\*\FF+4
\=03\
\*\3-4
\=FF\
\*\
```

<< Head 1 >>

10.6 Special Tricks with the Monitor

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

<< Head 2 >>

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.

<< Head 2 >>

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<300.32DM
```

```
\*\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, you first fill the memory range from $300 to $320 with zeros and verify it, then change one location and verify again, to see the VERIFY command detect the discrepancy:

```
\*\300:0

\*\301<300.31FM

\*\301<300.31FV

\*\304:02

\*\301<300.31FV

\0303-00 (02)\
\0304-02 (00)\
\*\
```

<< Head 2 >>

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\
\*\
```

<< Head 2 >>

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\

\*\
```

<< Head 1 >>

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.

-----------------------<< Gray Box >>-----------------------

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.

-----------------------<< End Box >>-----------------------

-------------<< Gloss >>------------
A mini-assembler is available with DOS, and an assembler is available with ProDOS. See Appendix D.

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.

<< Head 2 >>

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 the letter G, the Apple IIc starts executing machine language instructions starting at the specified location. If you just type the 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\
```

```
\*\300G
\ABCDEFGHIJKLMNOPQRSTUVWXYZ\
\*\
```

<< Head 2 >>

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.

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.

--------------<< Gloss >>------------

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`.

-------------<< Gloss >>------------

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 screenfull (20 lines) of instructions, as shown in the following example:

```
\*\300L

\0300-   A9 C1      LDA   #$C1\
\0302-   20 ED FD   JSR   $FDED\
\0305-   18         CLC\
\0306-   69 01      ADC   #$01\
```

```
\0308-   C9 DB       CMP   #$DB\
\030A-   D0 F6       BNE   $0302\
\030C-   60          RTS\
\030D-   00          BRK\
\030E-   00          BRK\
\030F-   00          BRK\
\0310-   00          BRK\
\0311-   00          BRK\
\0312-   00          BRK\
\0313-   00          BRK\
\0314-   00          BRK\
\0315-   00          BRK\
\0316-   00          BRK\
\0317-   00          BRK\
\0318-   00          BRK\
\0319-   00          BRK\
\*\
```

The first seven lines of this example are the assembly-language form of the program you typed in the previous example. The rest of the lines are BRK instructions only if this part of memory has 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.

<< Head 1 >>

10.8 Summary of Monitor Commands

Here is a summary of the Monitor commands, showing the syntax diagram for each one.

<< Head 2 >>

Examining Memory.

{adrs}	Examines the value contained in one location.
{adrs1}.{adrs2}	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.

<< Head 2 >>

Changing the Contents of Memory.

{adrs}:{val} {val}...	Stores the values in consecutive memory locations starting at {adrs}.
:{val}{val}...	Stores values in memory starting at the next changeable location.

<< Head 2 >>

Moving and Comparing

{dest}<{start}.{end}M	Copies the values in the range {start}.{end} into the range beginning at {dest}.
{dest}<{start}.{end}V	Compares the values in the range {start}.{end} to those in the range beginning at {dest}.

<< Head 2 >>

The Register Command

CTRL E	Displays the locations where the contents of the 65C02's registers are stored and opens them for changing.

<< Head 2 >>

Miscellaneous Monitor Commands

I	Sets Inverse display mode.
N	Sets Normal display mode.
CTRL B	Enters the language currently active (normally Applesoft).

CTRL C	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} CTRL P	Diverts output to the device connected to port number {port}. If {port}=0, sends output to the video display.
{port} CTRL-K	Takes input from the device connected to port number {port}. If {port}=0, accepts input from the keyboard.
CTRL Y	Jumps to the machine language subroutine at location $3F8.

<< Head 2 >>

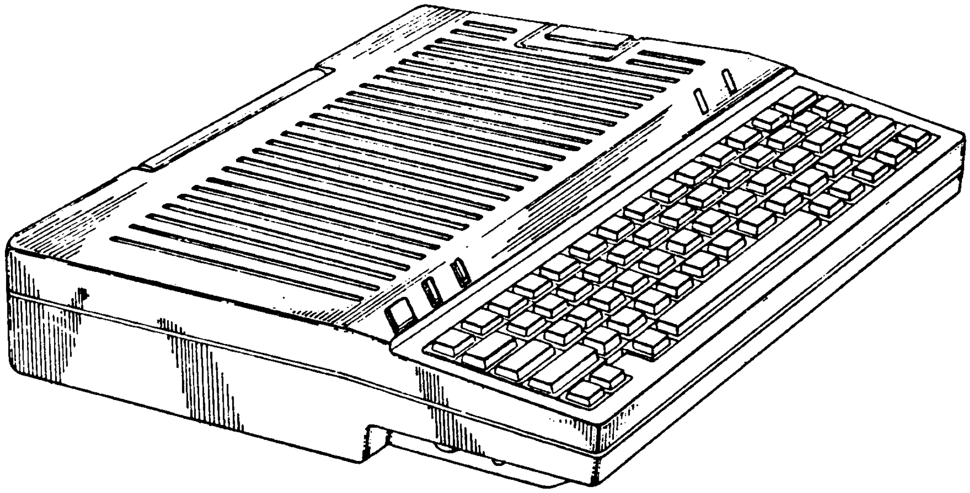
Running and Listing Programs.

{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.

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

CHAPTER 11 • HARDWARE



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Chapter 11

Hardware Implementation

Chapter 11

Hardware Implementation

[Draft reader's note: @0, @1 and @2 mean phi 0, 1 and 2; 0* C means zero degrees Centigrade]

Most of this manual describes functions--what the Lolly does. This chapter, on the other hand, describes objects: the pieces of hardware the Lolly uses to carry out its functions. If you are designing a device to connect to the Lolly back panel, or if you just want to know more about how the Lolly is built, you should study this chapter.

<< Head 1 >>

11.1 Environmental Specifications

The Lolly 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 Lolly is designed to function properly.

<< Table >>

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)

`Table 11-1.` Summary of Environmental Specifications

You should treat the Lolly with the same kind of care as any other electrical appliance. You should protect it from physical violence, such as hard bumps against furniture while you move it around. You should 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 Lolly, by blocking the ventilation openings in the top and bottom for example, the first symptom will be erratic operation. The memory devices in the Lolly are sensitive to heat: when they get too hot, they occasionally change a bit of data.

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.

<< Head 1 >>

11.2 Electrical Specifications

The electrical tolerances for the Lolly are defined by those of its power supply and internal voltage converter. This section describes those 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.

<< Head 2 >>

11.2.1 The External Power Supply

If you purchased your Lolly 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 Lolly internal converter. The basic specifications of the external power supply are listed in Table 11-2. The Lolly external power supply's cord should be plugged into a three-wire 115-volt (nominal) outlet. The line voltage must be in the range given in Table 11-2.

----------------------<< Warning Box >>---------------------

`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.

-----------------------<< End Box >>----------------------

------------------------------<< Table >>-----------------------------

Line voltage:	105 to 129 VAC
Maximum power consumption:	25 W continuous
Supply voltage:	+15 V DC (nominal)
Maximum supply current:	1.2 A continuous

`Table 11-2.` Power Supply Specifications

<< Head 2 >>

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 >>-----------------------------

[Figure 11-1]

`Figure 11-1.` External Power
Connector

-----------------------------<< Table >>-----------------------------

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

Table 11-3. External Power
Connector Signals

<< Head 2 >>

11.2.3 The Internal Converter

The internal converter in the Lolly operates on 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 Lolly itself. Minus 5 volts is derived from the -12 volts provided by the voltage converter.

<< Table >>

Input voltage:	+9 to 20 VDC	
Maximum power consumption:	31.5 W continuous	
Supply voltages:	+5V	+/-5%
	+12V	+/-6%
	-12V	+/-10%
Maximum supply currents:	+5V:	1.5 A
	+12V:	0.9 A continuous,
		1.5 A intermittent*
	-12V:	100 mA
	(-5V:	0.05 A)
Maximum case temperature:	60* C (140 F)	

`Table 11-4.` Internal Converter Specifications. *Intermittent load defined as 1.5 A for 100 milliseconds.

The Lolly 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 Lolly'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 Lolly 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.

<< Head 1 >>

11.3 Lolly Overall Block Diagram

Figure 11-2 is an overall block diagram of the Lolly. The following sections contain more detailed diagrams of the major parts of the machine. A full set of schematic diagrams of the Lolly appears in Section 11-14.

----------------------------<< Figure >>-----------------------------

[Figure 11-2]

`Figure 11-2.` Lolly Block Diagram

<< Head 1 >>

11.4 The CMOS 65C02 Microprocessor

The Lolly uses a CMOS 6502 (designated as 65C02) microprocessor as its central processing unit (CPU). The 65C02 in the Lolly runs at a clock rate of 1.023 MHz and performs up to 500,000 eight-bit operations per second.

-------------<< Gloss >>------------

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 1MHz clock is equivalent to many other types of microprocessors with clock rates up to 5MHz.

In addition to requiring lower power than earlier NMOS 6502 processors, the 65C02 in Lolly provides the programmer with 27 new instructions. (These instructions are described in Appendix A.) 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.

<< Head 2 >>

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.

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

<< Table >>

Type:	65C02
Register complement:	Accumulator (A) Index Registers (X,Y) Stack Pointer (S) Processor Status (P)
Register size:	Eight bits
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:	5mW (at 1MHz)

`Table 11-5.` 65C02 Microprocessor Specifications

<< Head 2 >>

11.4.2 Timing

The operation of the Lolly is controlled by a set of synchronous timing signals, sometimes called clock signals. In electronics, the word clock is used to identify signals that control the timing of circuit operations. The Lolly doesn't contain the kind of clock you tell time by, although its internal timing is accurate enough that a program running on the Lolly can simulate such a clock.

The frequency of the oscillator that generates the master timing signal is 14.31818 MHz. Circuitry in the Lolly 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 @1 and @0, complementary signals at a frequency of 1.02273 MHz. The Lolly signal named @0 is equivalent to the signal called @2 in the hardware manual (it isn't identical: it's a tiny bit early).

-------------<< Gloss >>------------

If you need more information about
the 65C02 itself, refer to the
Synertek Hardware Manual.

-----------------------------<< Figure >>-----------------------------

[Figure 11-4]

`Figure 11-4.` 65C02 Timing Signals

------------------------------<< Table >>------------------------------

Signal Name	Description
14M	Master oscillator, 14.31818 MHz; also 80-column dot clock.
7M	Intermediate timing signal and 40-column dot clock.
Q3	Intermediate timing signal, 2.04545 MHz with asymmetrical duty cycle
@0	Phase 0 of 65C02 clock, 1.022727 MHz; Complement of @1.
@1	Phase 1 of 65C02 clock, 1.022727 MHz; Complement of @0.

`Table 11-6.` 65C02 Timing Signal Descriptions

The operations of the 65C02 are related to the clock signals in a simple way: address during @1, data during @0. The 65C02 puts an address on the address bus during @1. This address is valid not later than 110 nanoseconds after @1 goes high and remains valid through all of @0. The 65C02 reads or writes data during @0. If the 65C02 is writing, the read/write signal is low during @0 and the 65C02 puts data on the data bus. The data is valid not later than 75 nanoseconds after @0 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 @0.

<< Head 1 >>

11.5 The Custom Integrated Circuits

Most of the circuitry that controls memory and I/O addressing in the Lolly is in five custom integrated circuits

- the Memory Management Unit (MMU)
- the Input-Output Unit (IOU)
- the Timing Generator (TMG) unit
- 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 Lolly 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 I/O.

<< Head 2 >>

11.5.1 The Memory Management Unit

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

Page 2 display (PAGE2): Chapter 5
Hi-res mode (HIRES): Chapter 5
Store to 80-column display (80STORE): Chapter 5
Select bank 2: Chapter 2
Enable bank-switched RAM: 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 (YINT): Chapter 9
Reset mouse X interrupt (XINT): 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.

The 64K dynamic RAMs used in the Lolly 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 >>------------------------------

[Figure 11-5]

`Figure 11-5.` The MMU Pinouts

------------------------------<< Table >>------------------------------

Pin Number	Name	Description
1	GND	Power and signal common
2	A0	65C02 address input
40-26	A1-A15	65C02 address input
3	@0	Clock phase 0
4	Q3	Timing signal
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
16	C06X'	Causes C06x outputs to go to 0 during @0.
17	EN80'	Enables auxiliary RAM
18	KBD'	Enable keyboard data bits 0-6
19	ROMEN2'	Enables built-in firmware ROM #2
20	ROMEN1'	Enables built-in firmware ROM #1
21	MD7	State of MMU flags
22	C07X	Causes C07x outputs to go to 0 during @0.
23	CASEN'	Enables main RAM
24	SELIO'	Goes to 0 during @0 for any access to $C0 page except $C08x, Bx, Cx or Fx.
25	+5V	Power
26-40	A15-A0	Sixteen-bit address bus

`Table 11-7.` The MMU Signal Descriptions

<< Head 2 >>

11.5.2 The Input/Output Unit

The circuitry inside the Input/Output Unit (IOU) implements the following soft switches, all described in Chapters 2 and 3:

Page 2 display
Hi-res mode
Text mode
Mixed mode
80-column display
Character-set select
Any-key-down
Mouse coordinates (X0, Y0)
Vertical blanking

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 Lolly require a multiplexed address, as described below 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 below in Section 11.9.1.

------------------------------<< Figure >>------------------------------

[Figure 11-6]

`Figure 11-6.` The IOU Pinouts

------------------------------<< Table >>------------------------------

Pin Number	Name	Description
1	GND	Power and signal common
2	GR	Graphics mode enable
3,4	SEGA,SEGB	High/low res character ROM signals
5	VC	Display vertical counter bit
6	80VID'	80-column video enable
7	CASSO	Reserved
8	SPKR	Speaker output signal
9	MD7	Internal IOU flags for data bus (bit 7)
10	Y0	Low when leading edge of Y0 selected; high when trailing edge of Y0 selected for IRQ'
11	MTRON	Not used.
12	MTR'	Not used
13	X0	Low when leading edge of X0 selected; high when trailing edge of X0 selected for IRQ'
14	R/W'	65C02 read-write control signal
15	RESET'	Power on and reset output
16	IRQ'	Mouse interface maskable interrupt line to 65C02
17-24	RA0-RA7	Multiplexed RAM address (phase 0)
25	PRAS'	Row-address strobe (phase 0)
26	@0	Master clock phase 0
27	Q3	Intermediate timing signal
28	+5V	Power
29	A6	Address bit 6 from 65C02
30	SELIO'	The SELIO' output for MMU pin 24
31	AKD	Any-key-down signal

32	KSTRB	Keyboard strobe signal
33,34	VID7,VID6	Video display control 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)

`Table 11-8.` The IOU Signal Descriptions

<< Head 2 >>

11.5.3 The TMG Unit

A custom timing generator chip (TMG) generates several timing and control signals in the Lolly. The TMG pinouts are shown in Figure 11-7; the signals are listed in Table 11-9.

------------------------------<< Figure >>------------------------------

[Figure 11-7]

`Figure 11-7.` The TMG Pinouts

------------------------------<< Table >>------------------------------

Pin Number	Name	Description
1	14M	14.31818 MHz master timing signal
2	7M	7.159009 MHz timing signal
3	CREF	3.579545 MHz color reference timing signal
4	H0	Horizontal video timing signal
5	VID7	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	ENTMG	Enable master timing
12	LDPS'	Video shift-register load enable
13	VID7M	Video dot clock enable, 7MHz or continuous 0
14	@1	Phase 1 system clock
15	@0	Phase 0 system clock
16	Q3	Intermediate timing and strobe signal
17	PCAS'	RAM Column-address strobe
18	-	Reserved for testing
19	PRAS'	RAM Row-address strobe
20	+5V	Power

`Table 11-9.` The TMG Signal Descriptions

<< Head 2 >>

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 read/write timing for all RAM used, double-high-resolution enable/disable, read data line 7 status registers and write command registers. It also provides IOU control for the areas of memory occupied by mouse interrupts and double-high-res soft-switch controls. Its pin assignments are shown in Figure 11-8 and its signals are listed in Table 11-10.

------------------------------<< Figure >>------------------------------

[Figure 11-8]

`Figure 11-8.` The GLU Pinouts

------------------------------<< Table >>------------------------------

Pin Number	Name	Description
1	D7	Indicates status of graphics, double-high-resolution and IOU status, depending on the address selected.
2,20,3-7	A0,A2-A7	Address lines to select least significant byte of addresses on C0 page.
8	PH0	Phase 0 of 1.0227 MHz processor synch 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	BRCLK	Baud rate clock output (1.7898 MHz) for serial interfaces.
14	EN80'	Selects the alternate RAM for 80-column display.
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	IOUSELIO'	Device select output to the IOU.
18	RR/W'	Main RAM read/write.
19	R/W80	Auxiliary RAM read/write.
21	DISK'	Disk controller device select output.
22	14M	Master clock (14.31313 MHz) for system.
23	SER'	Serial controller device select output.
24	Vcc	+5 volt supply.

`Table 11-10.` The GLU Signal Descriptions

<< Head 2 >>

11.5.5 The Disk Controller Unit (IWM)

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.

-------------<< Gloss >>------------
For further information on group
code recording, refer to
Section 11.10.

-----------------------------<< Figure >>-----------------------------

[Figure 11-9]

`Figure 11-9.` The IWM Pinouts

------------------------------<< Table >>------------------------------

Pin Number	Name	Description
1	PH0	Stepper motor control phase 0. One of four programmable disk drive motor phase outputs.
2	PH2	Stepper motor control phase 2.
3	A0	The data input to the state bit selected by A1 to A3.
	READ'	A low on this input enables the IWM to to send the register selected by the state onto the data bus.
4-6	A1-A3	These three inputs select one of the 8 bits in the state register to be updated.
7	DEV'	Device enable. The falling edge of DEV' latches information on A1-A3. The rising edge of either Q3 or DEV' 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	ENBL2'	Programmable buffered output lines, only one of which can be low at any one time. If one of them is low, then MTRON (motor on) is true.
20	ENBL1'	The second programmable buffered output line.
21	SENSE	An input to the IWM that can be polled via bit 7 of the status register.
22	RDDATA	Serial data input line. The IWM synchronizes the falling transition of each pulse.
23	RESET'	IWM reset: places all IWM outputs in their inactive state and sets all state and mode register bits to zero.
24	FCLK	Clock input (either 7 or 8 MHz) for internal

		synchronous logic.
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	PHASE3	Stepper motor control phase 3.
28	PHASE1	Stepper motor control phase 1.

`Table 11-11. The IWM Signal Descriptions

<< Head 1 >>

11.6 Memory Addressing

The 65C02 microprocessor can address 65,536 locations. The Lolly 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 Lolly and the way they are addressed. Input and output also use portions of the memory address space; refer to Chapter 2 for information.

<< Head 2 >>

11.6.1 ROM Addressing

In the Lolly, the following programs are permanently stored in a type 23128 16K by 8-bit ROMs (read-only memory):

- Applesoft editor and interpreter
- Monitor
- enhanced video firmware

The ROM is enabled by two signals called ROMEN1 and ROMEN2. (In the Lolly, ROMEN1 and ROMEN2 are directly electrically connected.) The segment of the ROM enabled by ROMEN1 occupies the memory address space from $C100 to $CFFF. The address space from $C300 to $C3FF and from $C800 to $CFFF contains the enhanced video firmware.

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

- ROM addresses $C100 to $C1FF and $C200 to $C2FF contain firmware for serial ports 1 and 2, respectively.

- ROM addresses $C400 to $C4FF contain mouse interface support.

- ROM addresses $C500 to $C5FF are reserved.

- ROM addresses $C600 to $C6FF contain 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 (for ProDOS or RAM-based programs) the external drive as if it were slot 7 drive 1, for external-drive startup only.

The remainder of this ROM, addressed from $D000 to $DFFF, contains part of the Applesoft BASIC interpreter.

The ROM segment enabled by ROMEN2, sometimes called the Monitor ROM, occupies the memory address space from $E000 to $FFFF. This ROM contains the rest of the Applesoft interpreter, in the address space from $E000 to $EFFF, and the Monitor subroutines, from $F000 to $FFFF.

The other ROMs in the Lolly are a type 2316 ROM (Figure 11-11) used for the keyboard character decoder, and a type 2332 ROM (Figure 11-12) used for character sets for the video display. This 2332 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.

-----------------------------<< Figure >>-----------------------------

[Figure 11-10]

`Figure 11-10.` The 2364 ROM Pinouts

----------------------------<< Figure >>----------------------------

[Figure 11-11]

`Figure 11-11.` The 2316 ROM Pinouts

----------------------------<< Figure >>----------------------------

[Figure 11-12]

`Figure 11-12.` The 2332 ROM Pinouts

<< Head 2 >>

11.6.2 RAM Addressing

The RAM (programmable) memory in the Lolly 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 Lolly, thanks to the way the microprocessor and the video circuits share the memory.

The memory circuits in the Lolly 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 @0, and the display circuits read data only during @1.

<< Head 3 >>

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 Lolly 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 Lolly refreshes the display sixty times per second.

The dynamic RAM devices used in the Lolly also need a kind of refreshent, because the data is stored in the form of electric charges which diminish with time and must be replenished every so often. The Lolly 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 <u>multiplexing</u>. 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 11-13).

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 half-millisecond (see Table 11-12). This more than satisfies the refresh requirements of the dynamic RAMs.

------------------------------<< Figure >>------------------------------

[Figure 11-13]

`Figure 11-13.` The 64K RAM Pinouts

------------------------------<< Table >>------------------------------

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

`Table 11-12.` RAM Address Multiplexing

<< Head 3 >>
Dynamic-RAM Timing

The Lolly'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 @0, and display data is strobed by the falling edge of @1, as shown in Figure 11-14.

----------------------------<< Figure >>----------------------------

[Figure 11-14]

`Figure 11-14.` 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 labelled 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 >>-----------------------------

Signal Name	Description
@0	Clock phase 0 (CPU phase)
@1	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

`Table 11-13.` RAM Timing Signals

<< Head 2 >>

11.7 The Keyboard

Figure 11-15 is an overall block diagram of the elements that make up keyboard input, as described in what follows.

The Lolly'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 11-15). 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 C70 and R32. The debounce time is also set externally, by C71.

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 >>-----------------------------

[Figure 11-15]

`Figure 11-15.` Keyboard Block Diagram

`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 >>-----------------------------

[Figure 11-17]

`Figure 11-17.` Keyboard Signals

<< Head 2 >>

11.8 The Speaker

The Lolly's built-in loudspeaker is controlled by a single bit of output from the IOU (Input Output Unit), amplified by a hybrid circuit (Figure 11-18).

-----------------------------<< Figure >>-----------------------------

[Figure 11-18]

`Figure 11-18.` Speaker Circuit
Diagram

<< Head 1 >>

1.8.1 3Volume Control

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.

<< Head 3 >>

11.8.2 Output Jack

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

<< Head 1 >>

11.9 The Video Display

The Lolly 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.

-----------------------<< Gray Box >>-----------------------

Lollys manufactured for sale in the U.S. generate a video signal that is compatible with the standards set by the NTSC (National Television Standards Committee). Lollys 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.

------------------------<< End Box >>-----------------------

The display portion of the video signal is a time-varying voltage generated from a stream of data bits, where a one corresponds to a voltage that generates a bright dot, and a zero 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.

<< Head 2 >>

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 below in the section "Display Address Mapping". 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 Lolly's video display is not interlaced.)

<< Head 2 >>

11.9.2 Display Memory Addressing

As described in Chapter 2 in the section "Addressing Display Pages Directly", 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. They will not [alas!] eliminate that necessity.

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 the section "Video Display Modes", below.

<< Head 2 >>

11.9.3 Display Address Mapping

Consider the simplest display on the Lolly, the 40-column text mode. To address forty columns requires six bits, and to address twenty-four rcws 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 above, in the section "Dynamic-RAM Refreshment".

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-x19is 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 >>------------------------------

[Figure 11-10]

`Figure 11-19.` Display Address Transformation

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 11-11). 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 11-11, 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 >>-----------------------------

[Figure 11-20]

`Figure 11-20.` 40-column Text Display Memory. Memory locations marked with an asterisk (*) are screen holes, described in Chapter 2.

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 2-5. Notice that the three rows in each block of 120 bytes are not adjacent on the display.

------------------------------<< Table >>------------------------------

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

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

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. Address bits marked with asterisks (*) are different for different modes: see Table 11-15, below, and the next three sections.

------------------------------<< Table >>------------------------------

Address Bit	Display Mode: Text and Lo-Res	Hi-Res
A10	80VID+PG2'	VA
A11	80VID'.PG2	VB
A12	0	VC
A13	0	80VID+PG2'
A14	0	80VID'.PG2

`Table 11-15.` Memory Address Bits for Display Modes

<< Head 2 >>

11.9.4 Video Display Modes

The different display modes all use the address-mapping scheme described in the previous section, but they use different-sized memory areas in different locations. The next three sections describe the addressing schemes and the methods of generating the actual video signals for the different display modes.

<< Head 3 >>

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 PG2 and 80VID, the display-page and 80-column-video soft switches. Address bits A12, A13, and A14 are set to zero. Notice that 80VID active inhibits PG2: 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.

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 @0), but their outputs are sent to the character generator alternately by @0 and @1. In 80-column mode, LDPS' loads data from the character generator into the shift register twice during each microsecond, once during @0 and once during @1, and VID7M remains low, enabling the clock continuously at 14M.

------------------------------<< Figure >>------------------------------

[Figure 11-21]

`Figure 11-21.` Video Timing Signals

<< Head 3 >>

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 11-17.

-----------------------------<< Table >>-----------------------------

Display mode	SEGA	SEGB	SEGC
Text	VA	VB	VC
Graphics	H0	HIRES'	VC

`Table 11-17.` Character-generator
Control Signals

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.

The video signal generated by the Lolly 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 Lolly'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.

<< Head 3 >>

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 PG2 and 80VID, the signals controlled by the display-page (PAGE2) and 80-column-video (80COL) soft switches. As in text mode, 80VID 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 Table 11-14 and Table 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 therefor 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 Lolly 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.

-----------------------<< Gray Box >>-----------------------

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.

-----------------------<< End Box >>-----------------------

<< Head 3 >>

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 pages 1 and 1X.

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.

-------------<< Gloss >>------------
For further information about double-high-resolution graphics display, refer to the bibliography.

<< Head 2 >>

11.9.5 Video Output Signals

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 Lolly (Figure 11-22):

- At the phono jack on the back of the Lolly.

- At the video expansion connector (pin 12) on the back panel (Table 11-16).

-----------------------------<< Figure >>----------------------------

[Figure 11-22]

`Figure 11-22.` Video Output Back Panel Connectors

<< Head 3 >>

Monitor Output

The sleeve of the phono jack at the center of the Lolly 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.

<< Head 3 >>
Video Expansion Output

The back panel of the Lolly has a DB-15 connector for sophisticated video interfaces external to the computer. Figure 11-23 shows the pin assignments for this connector; Table 11-16 describes the signals.

----------------------<< Warning Box >>---------------------

`Warning`
Several of these signals, such as 14 MHz, must be buffered within about 4 inches (100 cm) of the back panel connector--preferably inside a container directly connected to the back panel. For technical information, contact Apple Technical Support.

------------------------<< End Box >>-----------------------

------------------------------<< Figure >>------------------------------

[Figure 11-23]

`Figure 11-23.` The Video Expansion
Connector Pinouts

-----------------------------<< Table >>-----------------------------

Pin Number	Name	Description
1	TEXT	Video text signal from GLU
2	14M	14 MHz master timing signal from the system oscillator
3	SYNC'	Display synchronization signal from IOU pin 39.
4	SEGB	Display vertical counter bit from IOU pin 4.
5	1VSOUND	One-volt sound signal from pin 5 of the audio hybrid circuit (AUD).
6	LDPS'	Video shift-register load enable from pin 12 of TMG.
7	WNDW'	Active area display blanking.
8	+12 V	Regulated +12 volts.
9	PRAS'	RAM row-address strobe from TMG pin 19.
10	GR	Graphics mode enable from IOU pin 2.
11	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	VIDD7	From 74LS374 video latch; causes half-dot shift if high
15	CREF	Color reference signal from TMG pin 3.

`Table 11-16.` The Video Expansion Connector Signals

---------------------<< Warning Box >>---------------------

`Warning`
Caution: The maximum allowable current drain of +12 V regulated power at this 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.

-----------------------<< End Box >>-----------------------

<< Head 2 >>

11.10 Disk I/O

Disk I/O--for both the built-in and external drive--is supported by tThe IWM disk controller unit. The external drive is attached via a DB-19 connector. Figure 11-23 shows this connector; Table 11-18 describes their pin assignments. Supply voltages come from the power supply; all other signals come from the IWM, described in Section 11.5.5.

-----------------------------<< Figure >>-----------------------------

[Figure 11-23]

`Figure 11-23.` Disk Drive Connectors

------------------------------<< Table >>------------------------------

Connector Pin Number	Name	Description
1,2,3,4	GND	Ground reference and supply
6,16	+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

`Table 11-18.` Disk Drive Connector Signals

<< Head 1 >>

11.11 Serial I/O

Lolly has built into it two 6551 Asynchronous Communication Interface Adapters and supporting input and output buffers for full-duplex serial communication. Figure 11-24 is a block diagram of the Lolly serial ports. ACIA outputs are buffered by a 1448 quad line driver. Similarly, ACIA inputs are buffered by a 1489 quad line receiver.

Figure 11-25 is a detail block diagram of the 6551 ACIA. The registers are described in Sections 11.11.1 through 11.11.4.

-------------<< Gloss >>------------

The RS-232-C signals are defined in the Glossary.

-----------------------------<< Figure >>-----------------------------

[Figure 11-24]

`Figure 11-24.` Serial Port Block
Diagram

-----------------------------<< Figure >>-----------------------------

[Figure 11-25]

`Figure 11-25.` 6551 ACIA Block
Diagram

The 6551 pin assignments are shown in Figure 11-26 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 +5V by a 3.3Kohm pullup resistor.

-----------------------------<< Figure >>-----------------------------

[Figure 11-26]

Figure 11-26.` The 6551 Pinouts

------------------------------<< Table >>------------------------------

Pin Number	Name	Description
1	GND	Power and signal common ground
2	A4 A5	Address line 4 for serial port 1 Address line 5 for 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	EXTINT'	External interrupt (port 1 only)
17	KSTRB DSR	Keyboard strobe input (port 1) Data set ready input (port 2)
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

`Table 11-19.` The 6551 Signal Descriptions

The back panel connectors for both serial ports are 5-pin DIN jacks. The pin assignments are shown in Figure 11-27 and described in Table 11-20.

-----------------------------<< Figure >>------------------------------

[Figure 11-27]

`Figure 11-27.` Serial Port Back
Panel Connectors

-----------------------------<< Table >>-----------------------------

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

`Table 11-20.` Serial Port Connector Signals

<< Head 2 >>

11.11.1 ACIA Control Register

Figure 11-28 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 Lolly's TMG chip; the resulting baud rates are compared to the nominal baud rates in Table 11-21. Notice that many of these baud rates are somewhat below the nominal rate. (The EIA standard allows plus or minus 2% from the nominal rate.) If a Lolly serial port is used with a modem that is 2% above the nominal rate, then framing errors can occur, especially at 1200 baud and above, using 8 data bits. It may be necessary to select a lower baud rate for 8-bit binary data transfers.

------------------------------<< Figure >>------------------------------

[Figure 11-28]

`Figure 11-28.` ACIA Control Register

------------------------------<< Table >>------------------------------

Nominal	Actual	Nominal	Actual
50		1800	
75		2400	
110		3600	
135		4800	
150		7200	
300		9600	
600		19200	
1200			

`Table 11-21.` Serial Port Baud Rates

<< Head 2 >>

11.11.2 ACIA Command Register

Figure 11-29 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 >>----------------------------

[Figure 11-29]

`Figure 11-29.` ACIA Command
Register

<< Head 2 >>

11.11.3 ACIA Status Register

Figure 11-30 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 >>----------------------------

[Figure 11-30]

`Figure 11-30.` ACIA Status Register

<< Head 2 >>

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.

<< Head 2 >>

11.12 Mouse Input

The mouse is a hand-held X-Y pointing device with a button that someone can roll along a flat surface. This section describes a typical mouse to help you understand how the Lolly knows what it is doing in concrete terms.

Figure 11-32 gives you a look inside a mechanical mouse. It 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, make movements of the ball cause two disks to rotate. The disks have rectangular holes arranged near their edges, making them resemble slide mounts used with stereoscopic slide viewers.

The light from a tiny bulb reaches a photoreceptor whenever one of the holes on the disk lies between them. An internal circuit in the mouse makes 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 11-33), that varies directly with the speed of mouse movement. One of these indicates the X component (X0) of mouse movement, one 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 Lolly needs to know whether to add or to subtract 1 from a counter.

-----------------------------<< Figure >>------------------------------

[Figure 11-33]

`Figure 11-33.` Sample Mouse
Waveform

There is a second bulb/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 half-wave offset from their respective movement waves (Figure 11-34). 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 >>------------------------------

[Figure 11-34]

`Figure 11-34.` Mouse Movement and
Direction Waveforms

Figure 11-35 shows the pin assignments for the mouse DB09 connector on the back panel. Table 11-22 gives the signal names and descriptions.

------------------------------<< Figure >>------------------------------

[Figure 11-35]

`Figure 11-35.` Mouse Connector

------------------------------<< Table >>------------------------------

DB-9 Pin Number	Signal Name	Description
1	MOUSEID'	Mouse identifier: when active, disables NE556 hand controller chip.
2	+5V	Total current drain from this pin must not exceed 100mA. 100mA.
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.

`Table 11-35.` Mouse Connector

<< Head 2 >>

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 Lolly via the same DB-9 connector as the one used for the mouse (Section 11.12), but the Lolly interprets these signals differently.

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.

The switch inputs are multiplexed by a 74LS251 high-speed 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.

The nature and timing of the switch inputs are unusual (Figure 11-36). To use them, connect each one (as shown in Figure 11-37) to a single-pole, momentary-contact pushbutton switch.

-----------------------------<< Figure >>-----------------------------

[Figure 11-36]

`Figure 11-36.` Hand Controller Signal Diagram

----------------------------<< Figure >>----------------------------

[Figure 11-37]

`Figure 11-37.` How to Connect
Switch Inputs

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 one (high). A variable resistance of up to 150K ohms connected between one of these inputs and the +5V supply controls the charging time of one of the two 0.022-microfarad capacitors. When the voltage on the capacitor passes a certain threshold, the output of the NE556 changes back to zero (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.

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

----------------------------<< Figure >>----------------------------

[Figure 11-38]

`Figure 11-38.` Hand Control
Connector

--------------------------------<< Table >>------------------------------

Connector Pin Number	Signal Name	Description
7,1,6	SW1, SW0	Switch inputs (sometimes called paddle buttons).
2	+5V	+5V power supply. Total current drain from this pin must not exceed 100mA.
3	GND	System ground.
5,8,4,9	PDL0-PDL3	Hand control inputs. Each of these should be connected to a 150K-ohm variable resistor connected to +5V.

`Table 11-23.` Hand Control Connector Signals

<< Head 1 >>

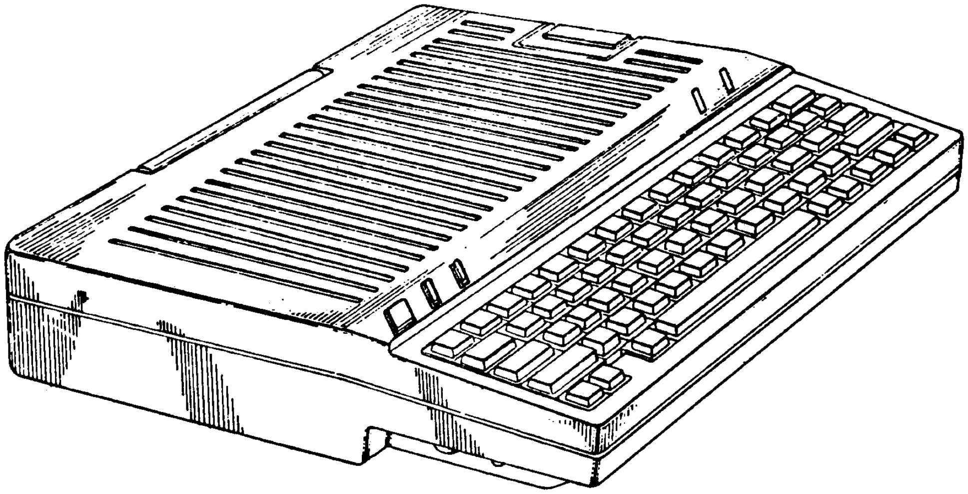
11.14 Schematic Diagrams

The following 5 pages contain schematic diagrams for the Lolly.

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

APPENDIX A • 65C02 MICRO-PROCESSOR



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Appendix A

The 65C02

Appendix A

The 65C02

This appendix summarizes the features of the 65C02 microprocessor, including its overall structure, instruction set, and differences from the 6502 microprocessor. For further information, refer to the bibliography.

<< Head 1 >>

A.1 Programming Model

Figure A-1 shows a model of the 65C02 processor. The lower portion of the figure depicts the parts of the processor that the programmer deals with directly: the accumulator (A), index registers (X and Y), the program counter (PC), stack pointer (S), and processor status register (P).

-----------------------------<< Figure >>-----------------------------

[Figure A-1]

`Figure A-1.` Model of 65C02 Microprocessor

<< Head 1 >>

A.2 Notation

Table A-1 shows the notation that applies to the instruction code and timing table in section A.3.

------------------------------<< Table >>------------------------------

----

`Table A-1.` Notation for
Instruction Code Tables

Table A-2 shows the notation to use for the various addressing modes when you specify them in your assembly language source program.

------------------------------<< Table >>------------------------------

Addressing Mode	Symbols	Example
Immediate		
Absolute		
Zero Page		
Accumulator		
Implied		
(Indirect, X)		
(Indirect, Y)		
Zero Page, X		
Zero Page, Y		
Absolute, X		
Absolute, Y		
Relative		
(Absolute)		
Absolute (Indirect, X)		
(Zero Page)		

`Table A-2.` Addressing Mode
Notation in Source Programs

<< Head 1 >>

A.3 Instruction Code Specifications

Table A-3 lists the operation codes, execution times and memory requirements of the 65C02 instruction set, as well as the resulting changes in the processor status register. New instructions, new addressing modes, and changed execution times are listed in section A.6.

Execution times are specified in number of cycles. One cycle time for the Apple IIc equals 1.9556 microseconds.

------------------------------<< Table >>------------------------------

----

`Table A-3.` 65C02 Instruction Code Specifications

<< Head 1 >>

A.4 Instruction Set by Alphabet

Table A-4 lists the 65C02 instructions in alphabetical order by their assembly-language names.

------------------------------<< Table >>------------------------------

----

`Table A-4.` 65C02 Instructions Sorted Alphabetically

<< Head 1 >>

A.5 Instruction Set by Operation Code

Table A-5 lists the 65C02 instructions in numerical order by their hexadecimal machine-language codes.

------------------------------<< Table >>------------------------------

----
`Table A-5.` 65C02 Instructions
Sorted by Number

<< Head 1 >>
A.6 Differences Between 6502 and 65C02

This section describes the new instructions and addressing modes of the 65C02, as well as the changed execution times of some of the instructions. 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.

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.

<< Head 2 >>
A.6.1 New Instructions

Table A-6 is a list of those 65C02 instructions that are not present in the 6502 instruction set.

------------------------------<< Table >>------------------------------

----
`Table A-6.` Instructions New to the
65C02

<< Head 2 >>
A.6.2 New Addressing Modes

Table A-7 lists the combinations of instructions and addressing modes available on the 65C02 and not available on the 6502.

------------------------------<< Table >>------------------------------

---

`Table A-7.` New
Instruction/Addressing Combinations

<< Head 2 >>

A.6.3 Differing Cycle Times

Table A-8 lists the instructions whose execution time is different on the 65C02 than on the 6502.

------------------------------<< Table >>------------------------------

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

`Table A-8.` Cycle Time Differences

<< Head 2 >>

A.6.4 Differing Instruction Results

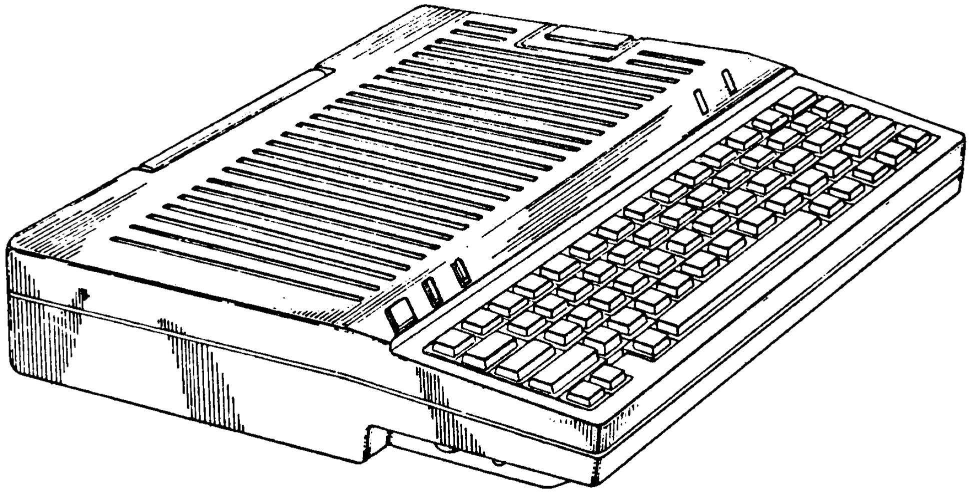
It is important to note that the BIT instruction, when used in immediate mode (code $89) leaves Processor Status Register bits 7 (N) and 6 (V) unchanged on the 65C02. On the 6502, all modes of the BIT instruction have the same effect on the Status Reegister: the value of memory bit 7 is placed in status 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.

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

APPENDIX B • MEMORY MAP



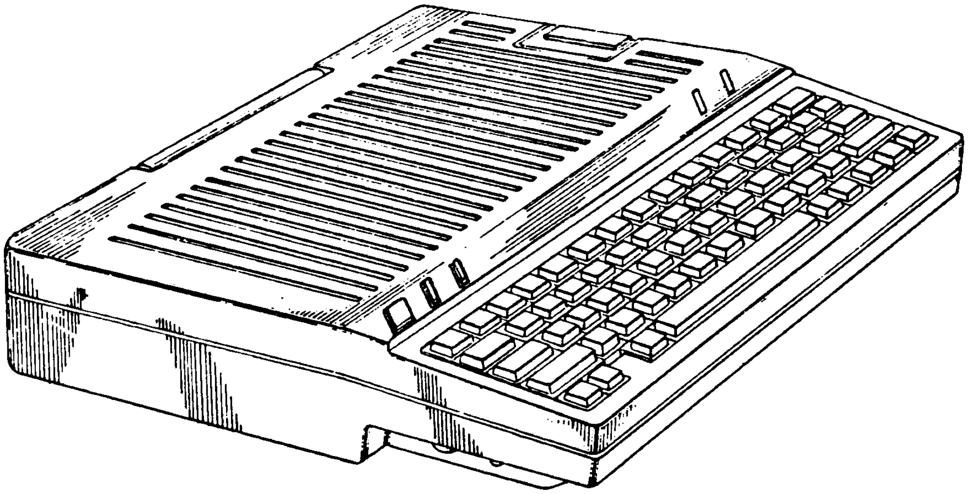
Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

APPENDIX C • FIRMWARE LOCATIONS



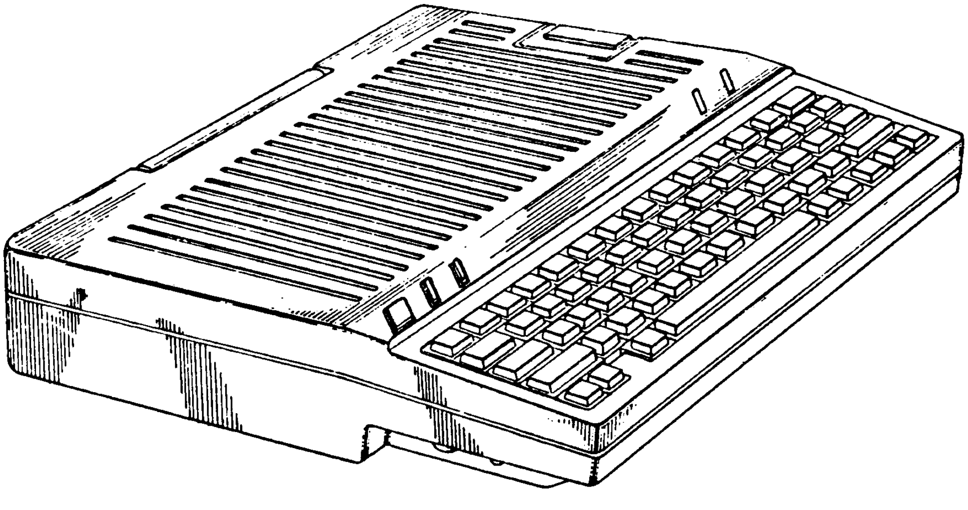
Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

APPENDIX D • OPERATING SYSTEMS



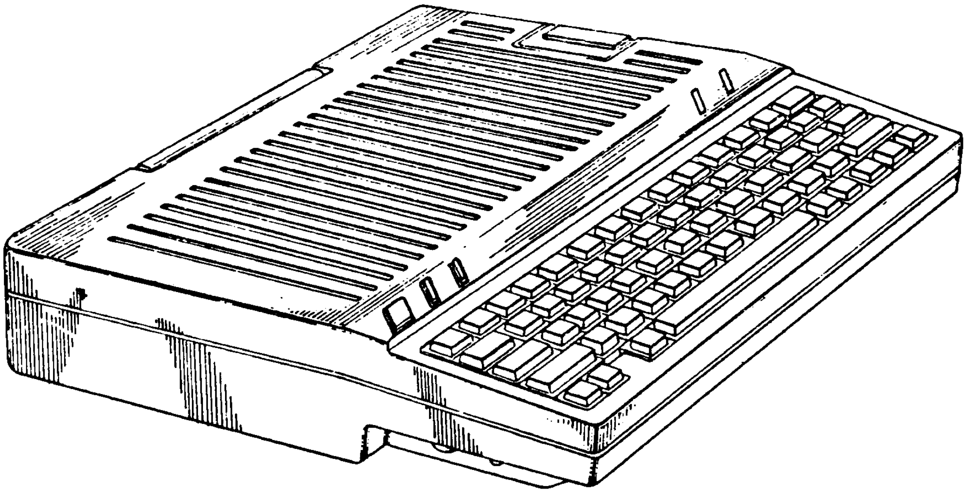
Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

APPENDIX E • LANGUAGES



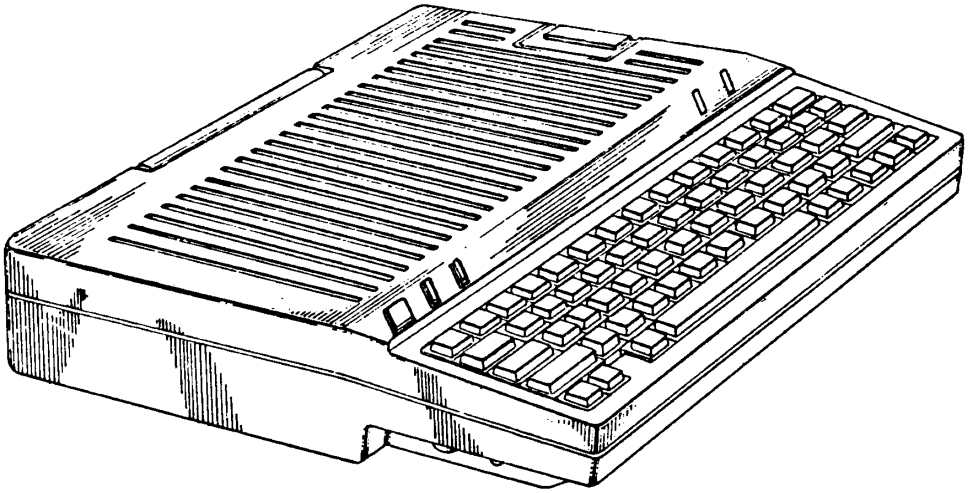
Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

APPENDIX F • //e & LOLLY DIFFERENCES



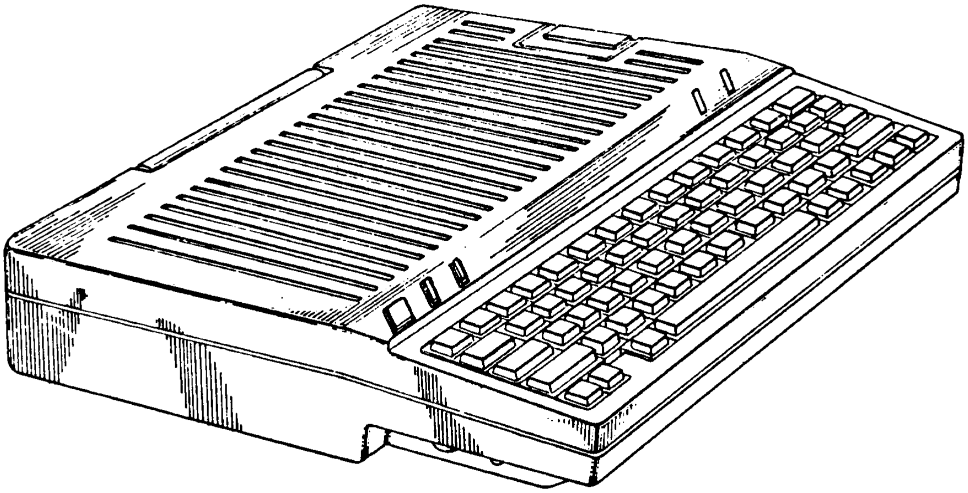
Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

APPENDIX G • USA & FOREIGN KEYBOARDS



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Appendix G

USA and International Models

Appendix G

USA and International Models

This appendix repeats some of the keyboard information given in Chapter 4 for the two USA keyboard layouts, for easy comparison with the other layouts available. Following these there is a composite table of the ASCII codes and the characters associated with them on all the models discussed.

<< Head 1 >>

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, by noting 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.

----------------------<< Gray Box >>----------------------

Note: CAPS LOCK affects only keys that can produce a lowercase letter (with or without an accent) and their uppercase equivalents. With these keys, CAPS LOCK down is equivalent to holding down the SHIFT key, 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 also shifts the top row of keys, selecting the numbers.

------------------------<< End Box >>-----------------------

-----------------------<< Gray Box >>-----------------------

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 are the use of symbols instead of words on 5 keys (TAB, two SHIFT keys, CAPS LOCK and RETURN), the shapes of three keys (the left SHIFT key, CAPS LOCK
and RETURN), and the resulting repositioning of two keys (\| and `~ in Figures~G-1 and~G-3).

~~~~~~------------------------<< End Box >>-----------------------~~~~~~

<< Head 2 >>

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 Lolly with the Keyboard switch up. Table G-1 lists the ASCII codes resulting from all simple and combination keystrokes on this keyboard.
~~~~~~

-----------------------------<< Figure >>-----------------------------

[Figure G-1]

`Figure G-1.` USA Standard or 'Sholes' Keyboard (Keyboard Switch Up)

----------------------------<< Table >>----------------------------

Key	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
Key	Hex	Char	Hex	Char	Hex	Char	Hex	Char
DELETE	7F	DEL	7F	DEL	7F	DEL	7F	DEL
L-ARROW	08	BS	08	BS	08	BS	08	BS
TAB	09	HT	09	HT	09	HT	09	HT
D-ARROW	0A	LF	0A	LF	0A	LF	0A	LF
U-ARROW	0B	VT	0B	VT	0B	VT	0B	VT
RETURN	0D	CR	0D	CR	0D	CR	0D	CR
R-ARROW	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	36	6	5E	^	5E	^
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	|	7F	DEL

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

-----------------------------<< Table >>-----------------------------

Key	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
Key	Hex	Char	Hex	Char	Hex	Char	Hex	Char
]}	5D	]	1D	GS	7D	}	1D	GS
`~	60	`	60	`	7E	~	7E	~
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB

X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

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

<< Head 2 >>

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 >>-----------------------------

[Figure G-2]

`Figure G-2.` USA Simplified or 'Dvorak' Keyboard (Keyboard Switch Down)

<< Head 2 >>

G.1.3 ISO Layout of USA Keyboard

Figure G-3 shows the layout of the keyboard of all European (ISO) keyboards when the Keyboard switch is up. All keystrokes produce the same ASCII codes as those shown in Table G-1.

----------------------------<< Figure >>----------------------------

[Figure G-3]

`Figure G-3.` ISO Version of USA Standard Keyboard (Keyboard Switch Up)

<< Head 2 >>

G.1.4 English Keyboard

With the Keyboard switch up, the English model of Lolly 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 what is 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 >>----------------------------

[Figure G-4]

`Figure G-4.` English Keyboard
(Keyboard Switch Down)

----------------------------<< Table >>----------------------------

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

`Table G-2.` English Keyboard Code
Differences from Table G-1

<< Head 2 >>

G.1.5 French and Canadian Keyboards

With the Keyboard switch up, the French model of Lolly 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.

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 Figure G-6. The changes in ASCII code production (from what is in Table G-1) are shown in Table G-3.

-----------------------------<< Figure >>------------------------------

[Figure G-5]

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

-----------------------------<< Figure >>------------------------------

[Figure G-6]

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

-----------------------------<< Table >>-----------------------------

Key	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
Key	Hex	Char	Hex	Char	Hex	Char	Hex	Char
&1	26	&	26	&	31	1	31	1
{2	7B	{	7B	{	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
}7	7D	}	7D	}	37	7	37	7
!8	21	!	21	!	38	8	38	8
\9	5C	\	1C	FS	39	9	1C	FS
@0	40	@	00	NUL	30	0	00	NUL
)[	29	)	1B	ESC	5B	[	1B	ESC
^~	5E	^	1E	RS	7E	~	1E	RS
$*	24	$	24	$	2A	*	2A	*
|%	7C	|	7C	|	25	%	25	%
`#	60	`	60	`	23	#	23	#
<>	3C	<	3C	<	3E	>	3E	>
,?	2C	,	2C	,	3F	?	3F	?
;.	3B	;	3B	;	2E	.	2E	.
:/	3A	:	3A	:	2F	/	2F	/

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

<< Head 2 >>

G.1.6 German Keyboard

With the Keyboard switch up, the German model of Lolly 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 what is in Table G-1) is shown in Table G-4.

-----------------------------<< Figure >>-----------------------------

[Figure G-7]

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

-----------------------------<< Table >>-----------------------------

Key	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
Key	Hex	Char	Hex	Char	Hex	Char	Hex	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	=
~?	7E	~	7E	~	3F	?	3F	?
}]	7D	}	1D	GS	5D	]	1D	GS

+*	2B	+	2B	+	2A	*	2A	*
|\	7C	|	1C	FS	5C	\	1C	FS
{[	7B	{	1B	ESC	5B	[	1B	ESC
#^	23	#	1E	RS	5E	^	1E	RS
<>	3C	<	3C	<	3E	>	3E	>
,;	2C	,	2C	,	3B	;	3B	;
.:	2E	.	2E	.	3A	:	3A	:

`Table G-4.` German Keyboard Code
Differences from Table G-1

<< Head 2 >>

G.1.7 Italian Keyboard

With the Keyboard switch up, the German model of Lolly 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-8. The change in ASCII code production (from what is in Table G-1) is shown in Table G-5.

---------------------------<< Figure >>----------------------------

[Figure G-8]

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

<< Table >>

Key	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
Key	Hex	Char	Hex	Char	Hex	Char	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
\5	5C	\	1C	FS	35	5	1C	FS
}6	7D	}	7D	}	36	6	36	6
)7	29	)	29	)	37	7	37	7
#8	23	#	23	#	38	7	38	8
{9	7B	{	7B	{	39	9	39	9
]0	5D	]	1D	GS	30	0	1D	GS
~^	7E	~	1E	RS	5E	^	1E	RS
$*	24	$	24	$	2A	*	2A	*
`%	60	`	60	`	25	%	25	%
@[	40	@	00	NUL	5B	[	1B	ESC
<>	3C	<	3C	<	3E	>	3E	>
,?	2C	,	2C	,	3F	?	3F	?
;.	3B	;	3B	;	2E	.	2E	.
:/	3A	:	3A	:	2F	/	2F	/
|!	7C	|	7C	|	21	!	21	!

`Table G-5.` Italian Keyboard Code
Differences from Table G-1

<< Head 2 >>

G.1.8 Western Spanish Keyboard

With the Keyboard switch up, the Western (that is, American) Spanish model of Lolly 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 what is in Table G-1) is shown in Table G-6.

----------------------------<< Figure >>----------------------------

[Figure G-9]

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

------------------------------<< Table >>------------------------------

Key	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
Key	Hex	Char	Hex	Char	Hex	Char	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
|\	7C	|	1C	FS	5C	\	1C	FS
}[	7D	}	7D	}	5B	[	5B	[
{@	7B	{	00	NUL	40	@	00	NUL
<>	3C	<	1E	RS	3E	>	1E	RS
,;	2C	,	2C	,	3B	;	3B	:
.:	2E	.	2E	.	3A	:	3A	:

`Table G-6.` Spanish Keyboard Code
Differences from Table G-1

<< Head 1 >>

G.2 ASCII Character Sets

Table G-7 lists the ASCII (American National Standard Code for Information Interchange) codes that the Lolly 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.

------------------------------<< Figure >>--------------------------------

[Figure G-7]

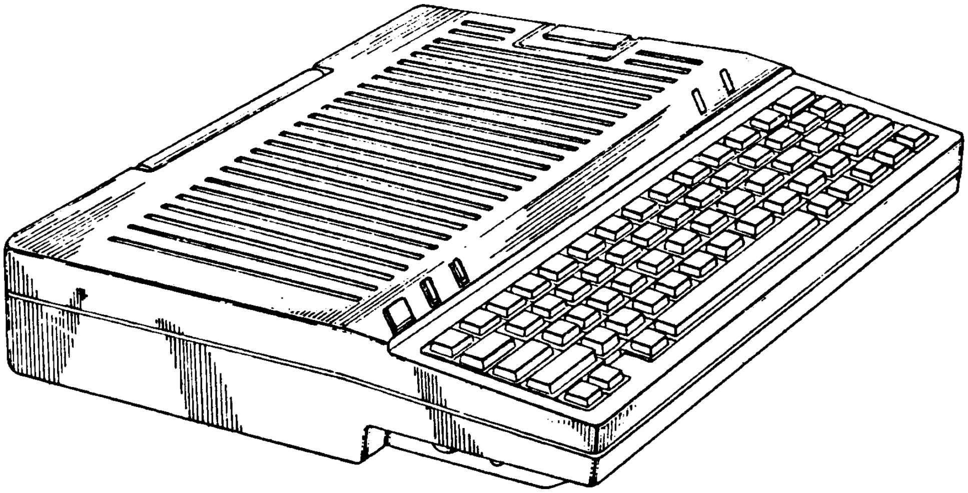
[use Table U-8 from the original International Supplement, page 32]

`Figure G-7.` ASCII Code Equivalents

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

APPENDIX H • CONVERSION TABLES



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Appendix H

Conversion Tables

Appendix H

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.

<< Head 1 >>

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 Lolly are listed in Table H-1.

------------------------------<< Table >>-----------------------------

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 info yet)
serial transfer	data	0 value	1 value
serial transfer	parity	SPACE	MARK
serial transfer	end		stop bit(s)
serial transfer	communic. 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"

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

- 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).

- 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 Lolly 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. Table H-8 lists some of the ways bytes are commonly interpreted.

-----------------------------<< Figure >>-----------------------------

[Figure H-1]

	high nibble				low nibble			
	MSB 7	6	5	4	3	2	1	LSB 0
hexadecimal	$80	$40	$20	$10	$08	$04	$02	$01
decimal	128	64	32	16	8	4	2	1

Nibbles and Hexadecimal Digits

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

Figure H-1.` Bits, Nibbles and Bytes

- Two bytes make a word (Figure H-2). 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.

-----------------------------<< Figure >>-----------------------------

[Figure H-2]

`Figure H-2.` Bytes and Words

<< Head 1 >>

H.2 Hexadecimal and Decimal

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

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

Examples: $3C = ? $FD47 = ?

$30 = 48 $F000 = 61440
$0C = 12 $ D00 = 3328
________ $ 40 = 64

$3C = 60

$ 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

------------------------------<< Table >>------------------------------

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

`Table H2.` Hexadecimal/Decimal Conversion

<< Head 1 >>

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.

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.

------------------------------<< Table >>------------------------------

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

^cp
`Table H-3.` Decimal to Negative Decimal Conversion

<< Head 1 >>

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. This table 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.

------------------------------<< Table >>------------------------------

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
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
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

`Table H-4.` Hexadecimal Values for
High-res Dot Patterns

<< Head 1 >>

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 Universal Utilities Disk presents a menu from which to select the characteristics of, say, a printer or modem. From the selections made, it generates a PIN for the user. Other products have a ready-made PIN that the user can simply type in.

Table H-7 is a definition of the PIN number digits. Notice that the PIN has a format similar to a telephone number, so it is easy to use.

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^ne 42

```
-----------------------------<< Table >>-----------------------------
                                    x  x  x  /  x  x  x  x


 1 = printer mode
 2 = communication mode *

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

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

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

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

 1 = do not echo output on screen
 2 = echo output on screen

 1 = do not generate LF after CR
 2 = generate LF after CR

 1 = do not generate CR *
 2 = generate CR after 40 characters
 3 = generate CR after 72 characters
 4 = generate CR after 80 characters
 5 = generate CR after 132 characters
```

---

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

`Table H-7.` PIN Numbers

<< Head 1 >>

H.6 Eight-bit Code Conversions

The table below shows the entire ASCII character set, and how to generate each character. Here is how to interpret this table:

- 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. The numbers between columns refer to footnotes.
- Angle brackets enclose the names of single keys (like ESC for the ESC key), or enclose keystrokes involving more than one key (like CONTROL-M, which means "hold down CONTROL while pressing M.")

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type
0000000	0	00	NUL	Blank (null)	CONTROL-@
0000001	1	01	SOH	Start of Header	CONTROL-A
0000010	2	02	STX	Start of Text	CONTROL-B
0000011	3	03	ETX	End of Text	CONTROL-C
0000100	4	04	EOT	End of Transm.	CONTROL-D
0000101	5	05	ENQ	Enquiry	CONTROL-E
0000110	6	06	ACK	Acknowledge	CONTROL-F
0000111	7	07	BEL	Bell	CONTROL-G
0001000	8	08	BS	Backspace	CONTROL-H or LEFT-ARROW
0001001	9	09	HT	Horizontal Tab	CONTROL-I or TAB
0001010	10	0A	LF	Linefeed	CONTROL-J or DOWN-ARROW
0001011	11	0B	VT	Vertical Tab	CONTROL-K or UP-ARROW
0001100	12	0C	FF	Form Feed	CONTROL-L
0001101	13	0D	CR	Carriage Return	CONTROL-M or RETURN
0001110	14	0E	SO	Shift Out	CONTROL-N
0001111	15	0F	SI	Shift In	CONTROL-O
0010000	16	10	DLE	Data Link Escape	CONTROL-P
0010001	17	11	DC1	Device Control 1	CONTROL-Q
0010010	18	12	DC2	Device Control 2	CONTROL-R
0010011	19	13	DC3	Device Control 3	CONTROL-S
0010100	20	14	DC4	Device Control 4	CONTROL-T
0010101	21	15	NAK	Neg. Acknowledge	CONTROL-U or RIGHT-ARROW
0010110	22	16	SYN	Synchronization	CONTROL-V
0010111	23	17	ETB	End of Text Blk.	CONTROL-W
0011000	24	18	CAN	Cancel	CONTROL-X
0011001	25	19	EM	End of Medium	CONTROL-Y
0011010	26	1A	SUB	Substitute	CONTROL-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-_

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type
0100000	32	20	SP	Space	spacebar
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		
0110001	49	31	1		
0110010	50	32	2		
0110011	51	33	3		
0110100	52	34	4		
0110101	53	35	5		
0110110	54	36	6		
0110111	55	37	7		
0111000	56	38	8		
0111001	57	39	9		
0111010	58	3A	:		
0111011	59	3B	;		
0111100	60	3C	<		
0111101	61	3D	=		
0111110	62	3E	>		
0111111	63	3F	?		

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type
1000000	64	40	@		
1000001	65	41	A		
1000010	66	42	B		
1000011	67	43	C		
1000100	68	44	D		
1000101	69	45	E		
1000110	70	46	F		
1000111	71	47	G		
1001000	72	48	H		
1001001	73	49	I		
1001010	74	4A	J		
1001011	75	4B	K		
1001100	76	4C	L		
1001101	77	4D	M		
1001110	78	4E	N		
1001111	79	4F	O		
1010000	80	50	P		
1010001	81	51	Q		
1010010	82	52	R		
1010011	83	53	S		
1010100	84	54	T		
1010101	85	55	U		
1010110	86	56	V		
1010111	87	57	W		
1011000	88	58	X		
1011001	89	59	Y		
1011010	90	5A	Z		
1011011	91	5B	[	Opening Bracket	
1011100	92	5C	\	Reverse Slant	
1011101	93	5D	]	Closing Bracket	
1011110	94	5E	^	Circumflex	
1011111	95	5F	_	Underline	

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type
1100000	96	60	'	Opening Quote	
1100001	97	61	a		
1100010	98	62	b		
1100011	99	63	c		
1100100	100	64	d		
1100101	101	65	e		
1100110	102	66	f		
1100111	103	67	g		
1101000	104	68	h		
1101001	105	69	i		
1101010	106	6A	j		
1101011	107	6B	k		
1101100	108	6C	l		
1101101	109	6D	m		
1101110	110	6E	n		
1101111	111	6F	o		
1110000	112	70	p		
1110001	113	71	q		
1110010	114	72	r		
1110011	115	73	s		
1110100	116	74	t		
1110101	117	75	u		
1110110	118	76	v		
1110111	119	77	w		
1111000	120	78	x		
1111001	121	79	y		
1111010	122	7A	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	

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type
10000000	128	80	NUL	Blank (null)	<CONTROL-@>
10000001	129	81	SOH	Start of Header	<CONTROL-A>
10000010	130	82	STX	Start of Text	<CONTROL-B>
10000011	131	83	ETX	End of Text	<CONTROL-C>
10000100	132	84	EOT	End of Transm.	<CONTROL-D>
10000101	133	85	ENQ	Enquiry	<CONTROL-E>
10000110	134	86	ACK	Acknowledge	<CONTROL-F>
10000111	135	87	BEL	Bell	<CONTROL-G>
10001000	136	88	BS	Backspace	<CONTROL-H> or LEFT-ARROW
10001001	137	89	HT	Horizontal Tab	<CONTROL-I> or TAB
10001010	138	8A	LF	Linefeed	<CONTROL-J> or DOWN-ARROW
10001011	139	8B	VT	Vertical Tab	<CONTROL-K> or UP-ARROW
10001100	140	8C	FF	Form Feed	<CONTROL-L>
10001101	141	8D	CR	Carriage Return	<CONTROL-M> or RETURN
10001110	142	8E	SO	Shift Out	<CONTROL-N>
10001111	143	8F	SI	Shift In	<CONTROL-O>
10010000	144	90	DLE	Data Link Escape	<CONTROL-P>
10010001	145	91	DC1	Device Control 1	<CONTROL-Q>
10010010	146	92	DC2	Device Control 2	<CONTROL-R>
10010011	147	93	DC3	Device Control 3	<CONTROL-S>
10010100	148	94	DC4	Device Control 4	<CONTROL-T>
10010101	149	95	NAK	Neg. Acknowledge	<CONTROL-U> or RIGHT-ARROW
10010110	150	96	SYN	Synchronization	<CONTROL-V>
10010111	151	97	ETB	End of Text Blk.	<CONTROL-W>
10011000	152	98	CAN	Cancel	<CONTROL-X>
10011001	153	99	EM	End of Medium	<CONTROL-Y>
10011010	154	9A	SUB	Substitute	<CONTROL-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-_>

Binary	Dec	Hex	Char	ASCII Interpretation	What to Type
10100000	160	A0	SP	Space	spacebar
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		
10110001	177	B1	1		
10110010	178	B2	2		
10110011	179	B3	3		
10110100	180	B4	4		
10110101	181	B5	5		
10110110	182	B6	6		
10110111	183	B7	7		
10111000	184	B8	8		
10111001	185	B9	9		
10111010	186	BA	:		
10111011	187	BB	;		
10111100	188	BC	<		
10111101	189	BD	=		
10111110	190	BE	>		
10111111	191	BF	?		

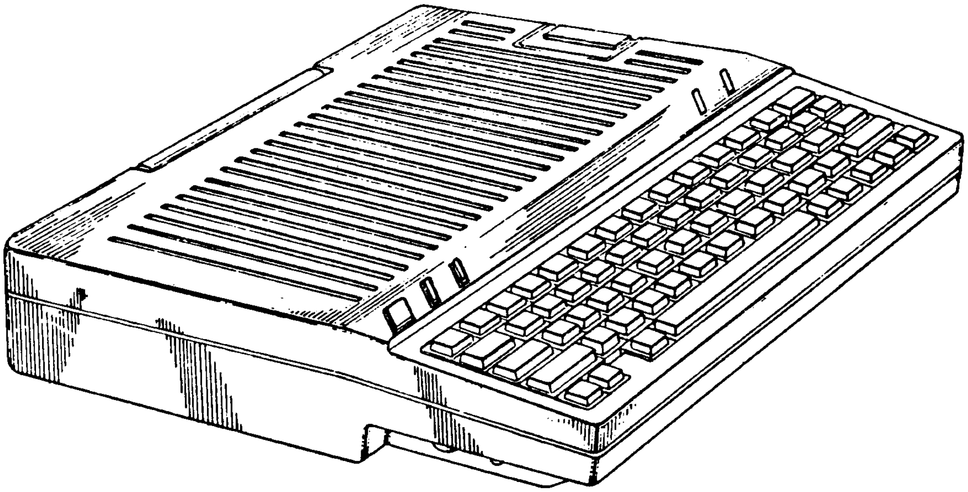
Binary	Dec	Hex	Char	ASCII Interpretation	What to Type
11000000	192	C0	@		
11000001	193	C1	A		
11000010	194	C2	B		
11000011	195	C3	C		
11000100	196	C4	D		
11000101	197	C5	E		
11000110	198	C6	F		
11000111	199	C7	G		
11001000	200	C8	H		
11001001	201	C9	I		
11001010	202	CA	J		
11001011	203	CB	K		
11001100	204	CC	L		
11001101	205	CD	M		
11001110	206	CE	N		
11001111	207	CF	O		
11010000	208	D0	P		
11010001	209	D1	Q		
11010010	210	D2	R		
11010011	211	D3	S		
11010100	212	D4	T		
11010101	213	D5	U		
11010110	214	D6	V		
11010111	215	D7	W		
11011000	216	D8	X		
11011001	217	D9	Y		
11011010	218	DA	Z		
11011011	219	DB	[	Opening Bracket	
11011100	220	DC	\	Reverse Slant	
11011101	221	DD	]	Closing Bracket	
11011110	222	DE	^	Circumflex	
11011111	223	DF	_	Underline	

Binary	Dec	Hex	Char	ASCII Interpretation	What to Type
11100000	224	E0	'	Open Quote (grave accent)	
11100001	225	E1	a		
11100010	226	E2	b		
11100011	227	E3	c		
11100100	228	E4	d		
11100101	229	E5	e		
11100110	230	E6	f		
11100111	231	E7	g		
11101000	232	E8	h		
11101001	233	E9	i		
11101010	234	EA	j		
11101011	235	EB	k		
11101100	236	EC	l		
11101101	237	ED	m		
11101110	238	EE	n		
11101111	239	EF	o		
11110000	240	F0	p		
11110001	241	F1	q		
11110010	242	F2	r		
11110011	243	F3	s		
11110100	244	F4	t		
11110101	245	F5	u		
11110110	246	F6	v		
11110111	247	F7	w		
11111000	248	F8	x		
11111001	249	F9	y		
11111010	250	FA	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

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

GLOSSARY



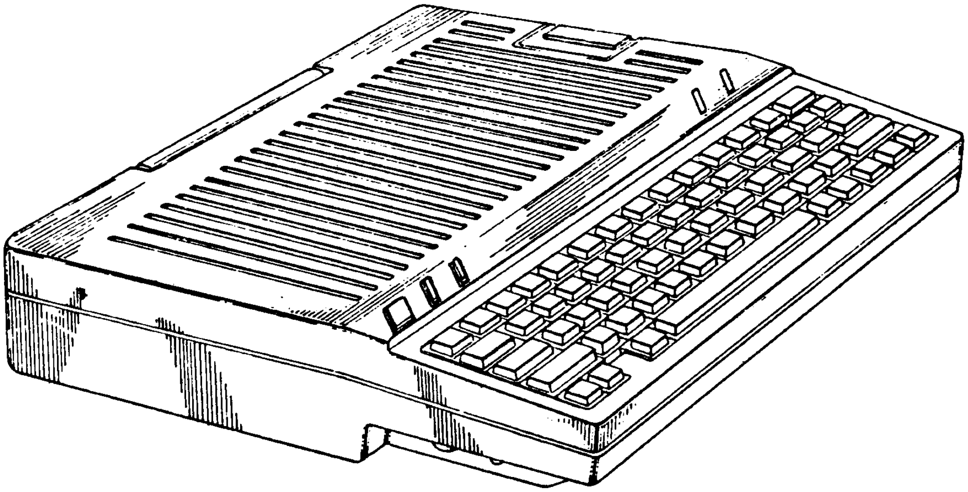
Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

BIBLIOGRAPHY



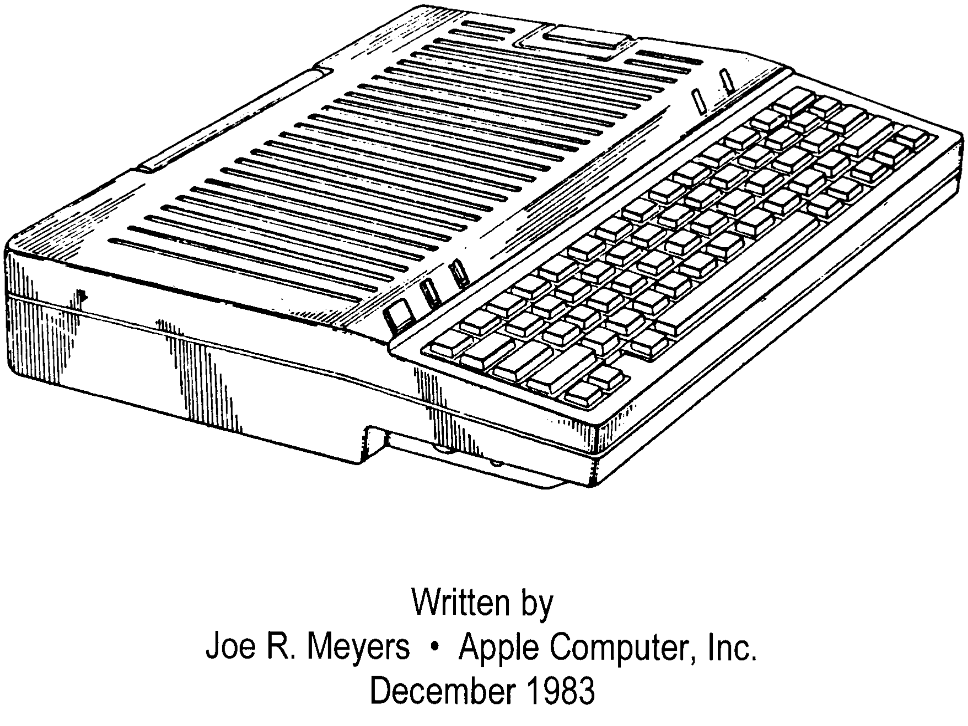
Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

FIRMWARE LISTINGS



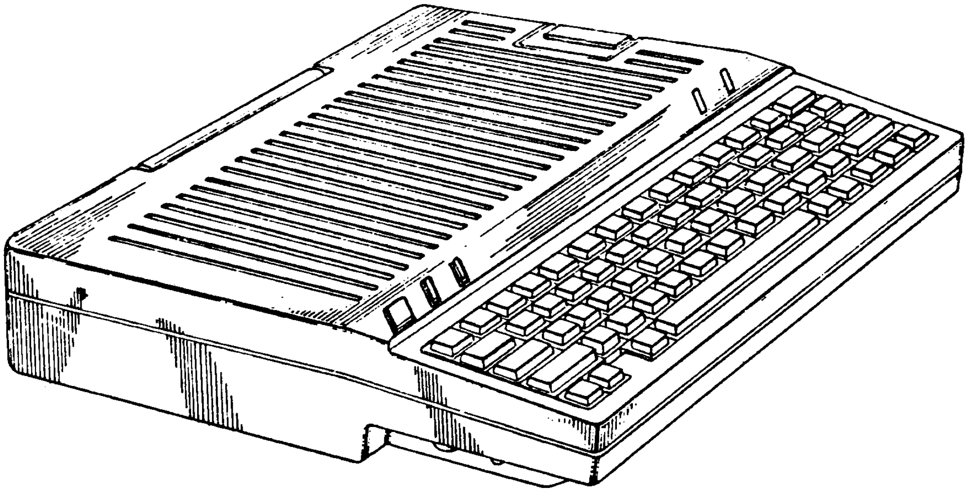
Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

INDEXES



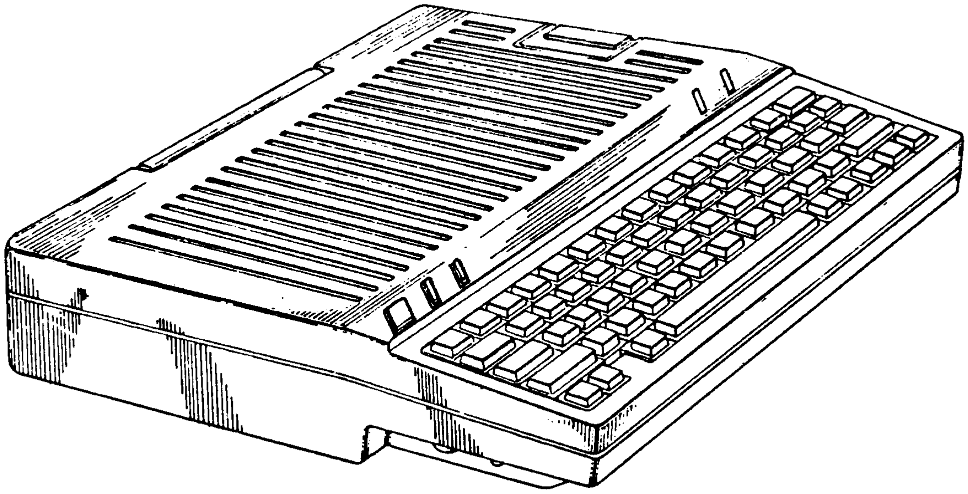
Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)

Apple //c Computer Information

Apple //c Technical Reference Manual
Pre-Release Draft Copy

THE END



Written by
Joe R. Meyers • Apple Computer, Inc.
December 1983

(This page is not part of the original document)