

Noy

APPLE II[®]

REFERENCE MANUAL



 apple computer inc.

NOTICE

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

This manual is copyrighted and contains proprietary information. All rights are reserved. This document may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from Apple Computer Inc.

©1979 by **Apple Computer Inc.**
10260 Bandley Drive
Cupertino, CA 95014
(408) 996-1010

Reorder Apple product number A2L0001A (030-0004-01)

Written by Christopher Espinosa

"Apple" is a trademark of Apple Computer Inc.



Apple II Reference Manual

A REFERENCE MANUAL
FOR THE APPLE II
AND THE APPLE II PLUS
PERSONAL COMPUTERS

TABLE OF CONTENTS

CHAPTER 1 APPROACHING YOUR APPLE

2	THE POWER SUPPLY
3	THE MAIN BOARD
4	TALKING TO YOUR APPLE
5	THE KEYBOARD
6	READING THE KEYBOARD
9	THE APPLE VIDEO DISPLAY
9	THE VIDEO CONNECTOR
10	EURAPPLE (50 HZ) MODIFICATION
10	SCREEN FORMAT
12	SCREEN MEMORY
12	SCREEN PAGES
12	SCREEN SWITCHES
14	THE TEXT MODE
17	THE LOW-RESOLUTION GRAPHICS (LO-RES) MODE
19	THE HIGH-RESOLUTION GRAPHICS (HI-RES) MODE
20	OTHER INPUT/OUTPUT FEATURES
20	THE SPEAKER
22	THE CASSETTE INTERFACE
23	THE GAME I/O CONNECTOR
23	ANNUNCIATOR OUTPUTS
24	ONE-BIT INPUTS
24	ANALOG INPUTS
25	STROBE OUTPUT
25	VARIETIES OF APPLES
25	AUTOSTART ROM / MONITOR ROM
26	REVISION 0 / REVISION 1 BOARD
27	POWER SUPPLY CHANGES
27	THE APPLE II PLUS

CHAPTER 2 CONVERSATION WITH APPLES

- 30 STANDARD OUTPUT
- 30 THE STOP-LIST FEATURE
- 31 BUT SOFT, WHAT LIGHT THROUGH YONDER WINDOW BREAKS!
(OR, THE TEXT WINDOW)
- 32 SEEING IT ALL IN BLACK AND WHITE
- 32 STANDARD INPUT
- 32 RDKEY
- 33 GETLN
- 34 ESCAPE CODES
- 36 THE RESET CYCLE
- 36 AUTOSTART ROM RESET
- 37 AUTOSTART ROM SPECIAL LOCATIONS
- 38 "OLD MONITOR" ROM RESET

CHAPTER 3 THE SYSTEM MONITOR

- 40 ENTERING THE MONITOR
- 40 ADDRESSES AND DATA
- 41 EXAMINING THE CONTENTS OF MEMORY
- 41 EXAMINING SOME MORE MEMORY
- 43 EXAMINING STILL MORE MEMORY
- 43 CHANGING THE CONTENTS OF A LOCATION
- 44 CHANGING THE CONTENTS OF CONSECUTIVE LOCATIONS
- 44 MOVING A RANGE OF MEMORY
- 46 COMPARING TWO RANGES OF MEMORY
- 46 SAVING A RANGE OF MEMORY ON TAPE
- 47 READING A RANGE FROM TAPE
- 48 CREATING AND RUNNING MACHINE LANGUAGE PROGRAMS
- 49 THE MINI-ASSEMBLER

- 51 DEBUGGING PROGRAMS
- 53 EXAMINING AND CHANGING REGISTERS
- 54 MISCELLANEOUS MONITOR COMMANDS
- 55 SPECIAL TRICKS WITH THE MONITOR
- 57 CREATING YOUR OWN COMMANDS
- 59 SUMMARY OF MONITOR COMMANDS
- 61 SOME USEFUL MONITOR SUBROUTINES
- 65 MONITOR SPECIAL LOCATIONS
- 66 MINI-ASSEMBLER INSTRUCTION FORMATS

CHAPTER 4 MEMORY ORGANIZATION

- 68 RAM STORAGE
- 70 RAM CONFIGURATION BLOCKS
- 72 ROM STORAGE
- 73 I/O LOCATIONS
- 74 ZERO PAGE MEMORY MAPS

CHAPTER 5 INPUT/OUTPUT STRUCTURE

- 78 BUILT-IN I/O
- 79 PERIPHERAL BOARD I/O
- 80 PERIPHERAL CARD I/O SPACE
- 80 PERIPHERAL CARD ROM SPACE
- 81 I/O PROGRAMMING SUGGESTIONS
- 82 PERIPHERAL SLOT SCRATCHPAD RAM
- 83 THE CSW/KSW SWITCHES
- 84 EXPANSION ROM

CHAPTER 6 HARDWARE CONFIGURATION

- 88 THE MICROPROCESSOR
- 90 SYSTEM TIMING
- 92 POWER SUPPLY
- 94 ROM MEMORY
- 95 RAM MEMORY
- 96 THE VIDEO GENERATOR
- 97 VIDEO OUTPUT JACKS
- 98 BUILT-IN I/O
- 99 "USER 1" JUMPER
- 100 THE GAME I/O CONNECTOR
- 100 THE KEYBOARD
- 102 KEYBOARD CONNECTOR
- 103 CASSETTE INTERFACE JACKS
- 104 POWER CONNECTOR
- 105 SPEAKER
- 105 PERIPHERAL CONNECTORS



- ¹¹⁷ APPENDIX A
THE 6502 INSTRUCTION SET
- ¹²⁹ APPENDIX B
SPECIAL LOCATIONS
- ¹³⁵ APPENDIX C
ROM LISTINGS
- ¹⁷⁷ GLOSSARY
- ¹⁸⁵ BIBLIOGRAPHY

INDEX

- 190 GENERAL INDEX
- 194 INDEX OF FIGURES
- 195 INDEX OF PHOTOS
- 195 INDEX OF TABLES
- 195 CAST OF CHARACTERS



INTRODUCTION

This is the User Reference Manual for the Apple II and Apple II Plus personal computers. Like the Apple itself, this book is a tool. As with all tools, you should know a little about it before you start to use it.

This book will not teach you how to program. It is a book of facts, not methods. If you have just unpacked your Apple, or you do not know how to program in any of the languages available for it, then before you continue with this book, read one of the other manuals accompanying your Apple. Depending upon which variety of Apple you have purchased, you should have received one of the following:

Apple II BASIC Programming Manual
(part number A2L0005)

The Applesoft Tutorial
(part number A2L0018)

These are tutorial manuals for versions of the BASIC language available on the Apple. They also include complete instructions on setting up your Apple. The Bibliography at the end of this manual lists other books which may interest you.

There are a few different varieties of Apples, and this manual applies to all of them. It is possible that some of the features noted in this manual will not be available on your particular Apple. In places where this manual mentions features which are not universal to all Apples, it will use a footnote to warn you of these differences.

This manual describes the Apple II computer and its parts and procedures. There are sections on the System Monitor, the input/output devices and their operation, the internal organization of memory and input/output devices, and the actual electronic design of the Apple itself. For information on any other Apple hardware or software product, please refer to the manual accompanying that product.

Franklin, Wm. Wm. Franklin, Franklin, Franklin, Franklin, Franklin,

CHAPTER 1

APPROACHING YOUR APPLE

- 2 THE POWER SUPPLY
- 3 THE MAIN BOARD
- 4 TALKING TO YOUR APPLE
- 5 THE KEYBOARD
- 6 READING THE KEYBOARD
- 9 THE APPLE VIDEO DISPLAY
- 9 THE VIDEO CONNECTOR
- 10 EURAPPLE (50 HZ) MODIFICATION
- 10 SCREEN FORMAT
- 12 SCREEN MEMORY
- 12 SCREEN PAGES
- 12 SCREEN SWITCHES
- 14 THE TEXT MODE
- 17 THE LOW-RESOLUTION GRAPHICS (LO-RES) MODE
- 19 THE HIGH-RESOLUTION GRAPHICS (HI-RES) MODE
- 20 OTHER INPUT/OUTPUT FEATURES
- 20 THE SPEAKER
- 22 THE CASSETTE INTERFACE
- 23 THE GAME I/O CONNECTOR
- 23 ANNUNCIATOR OUTPUTS
- 24 ONE-BIT INPUTS
- 24 ANALOG INPUTS
- 25 STROBE OUTPUT
- 25 VARIETIES OF APPLES
- 25 AUTOSTART ROM / MONITOR ROM
- 26 REVISION 0 / REVISION 1 BOARD
- 27 POWER SUPPLY CHANGES
- 27 THE APPLE II PLUS

For detailed information on setting up your Apple, refer to Chapter 1 of either the **Apple BASIC Programming Manual** or **The Applesoft Tutorial**.

In this manual, all directional instructions will refer to this orientation: with the Apple's typewriter-like keyboard facing you, "front" and "down" are towards the keyboard, "back" and "up" are away. Remove the lid of the Apple by prying up the back edge until it "pops", then pull straight back on the lid and lift it off.

This is what you will see:

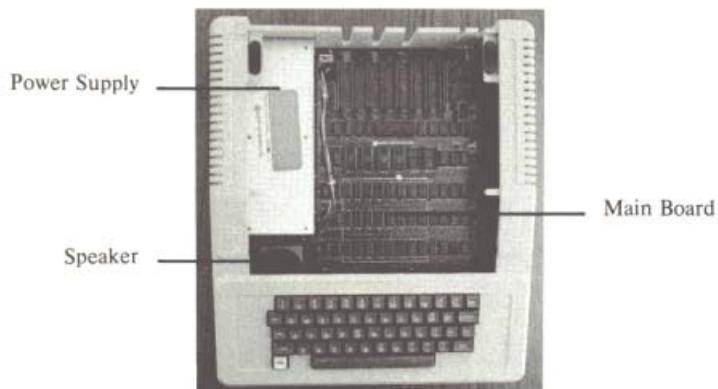


Photo 1. The Apple II.

THE POWER SUPPLY

The metal box on the left side of the interior is the Power Supply. It supplies four voltages: +5v, -5.2v, +11.8v, and -12.0v. It is a high-frequency "switching"-type power supply, with many protective features to ensure that there can be no imbalances between the different supplies. The main power cord for the computer plugs directly into the back of the power supply. The power-on switch is also on the power supply itself, to protect you and your fingers from accidentally becoming part of the high-voltage power supply circuit.

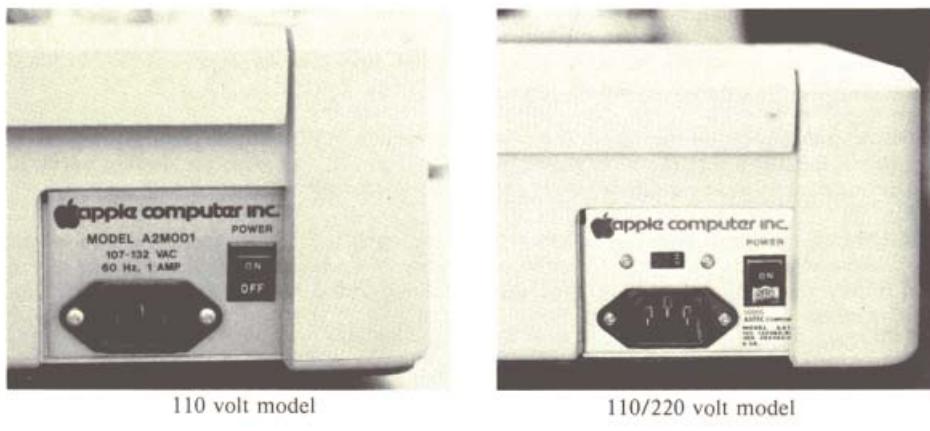


Photo 2. The back of the Apple Power Supply.

THE MAIN BOARD

The large green printed circuit board which takes up most of the bottom of the case is the computer itself. There are two slightly different models of the Apple II main board: the original (Revision 0) and the Revision 1 board. The slight differences between the two lie in the electronics on the board. These differences are discussed throughout this book. A summary of the differences appears in the section "Varieties of Apples" on page 25.

On this board there are about eighty integrated circuits and a handful of other components. In the center of the board, just in front of the eight gold-toothed edge connectors ("slots") at the rear of the board, is an integrated circuit larger than all others. This is the brain of your Apple. It is a Synertek/MOS Technology 6502 microprocessor. In the Apple, it runs at a rate of 1,023,000 machine cycles per second and can do over five hundred thousand addition or subtraction operations in one second. It has an addressing range of 65,536 eight-bit bytes. Its repertory includes 56 instructions with 13 addressing modes. This microprocessor and other versions of it are used in many computers systems, as well as other types of electronic equipment.

Just below the microprocessor are six sockets which may be filled with from one to six slightly smaller integrated circuits. These ICs are the Read-Only Memory (ROM) "chips" for the Apple. They contain programs for the Apple which are available the moment you turn on the power. Many programs are available in ROM, including the Apple System Monitor, the Apple Autostart Monitor, Apple Integer BASIC and Applesoft II BASIC, and the Apple *Programmer's Aid #1* utility subroutine package. The number and contents of your Apple's ROMs depend upon which type of Apple you have, and the accessories you have purchased.

Right below the ROMs and the central mounting nut is an area marked by a white square on the board which encloses twenty-four sockets for integrated circuits. Some or all of these may be filled with ICs. These are the main Random Access Memory (RAM) "chips" for your Apple. An Apple can hold 4,096 to 49,152 bytes of RAM memory in these three rows of components.* Each row can hold eight ICs of either the 4K or 16K variety. A row must hold eight of the same

* You can extend your RAM memory to 64K by purchasing the Apple Language Card, part of the Apple Language System (part number A2B0006).

type of memory components, but the two types can both be used in various combinations on different rows to give nine different memory sizes.* The RAM memory is used to hold all of the programs and data which you are using at any particular time. The information stored in RAM disappears when the power is turned off.

The other components on the Apple II board have various functions: they control the flow of information from one part of the computer to another, gather data from the outside world, or send information to you by displaying it on a television screen or making a noise on a speaker.

The eight long peripheral slots on the back edge of the Apple's board can each hold a peripheral card to allow you to extend your RAM or ROM memory, or to connect your Apple to a printer or other input/output device. These slots are sometimes called the Apple's "backplane" or "mother board".

TALKING TO YOUR APPLE

Your link to your Apple is at your fingertips. Most programs and languages that are used with the Apple expect you to talk to them through the Apple's keyboard. It looks like a normal typewriter keyboard, except for some minor rearrangement and a few special keys. For a quick review on the keyboard, see pages 6 through 12 in the **Apple II BASIC Programming Manual** or pages 5 through 11 in **The Applesoft Tutorial**.

Since you're talking with your fingers, you might as well be hearing with your eyes. The Apple will tell you what it is doing by displaying letters, numbers, symbols, and sometimes colored blocks and lines on a black-and-white or color television set.

* The Apple II is designed to use both the 16K and the less expensive 4K RAMs. However, due to the greater availability and reduced cost of the 16K chips, Apple now supplies only the 16K RAMs.

THE KEYBOARD

The Apple Keyboard

Number of Keys: 52

Coding: Upper Case ASCII

Number of codes: 91

Output: Seven bits, plus strobe

Power requirements: +5v at 120mA
-12v at 50mA

Rollover: 2 key

Special keys: CTRL
ESC
RESET
REPT
— →

Memory mapped locations:	Hex	Decimal	
Data	\$C000	49152	-16384
Clear	\$C010	49168	-16368

The Apple II has a built-in 52-key typewriter-like keyboard which communicates using the American Standard Code for Information Interchange (ASCII)*. Ninety-one of the 96 upper-case ASCII characters can be generated directly by the keyboard. Table 2 shows the keys on the keyboard and their associated ASCII codes. "Photo" 3 is a diagram of the keyboard.

The keyboard is electrically connected to the main circuit board by a 16-conductor cable with plugs at each end that plug into standard integrated circuit sockets. One end of this cable is connected to the keyboard; the other end plugs into the Apple board's keyboard connector, near the very front edge of the board, under the keyboard itself. The electrical specifications for this connector are given on page 102.

Most languages on the Apple have commands or statements which allow your program to accept input from the keyboard quickly and easily (for example, the INPUT and GET statements in BASIC). However, your programs can also read the keyboard directly.

* All ASCII codes used by the Apple normally have their high bit set. This is the same as standard mark-parity ASCII.



"Photo" 3. The Apple Keyboard.

READING THE KEYBOARD

The keyboard sends seven bits of information which together form one character. These seven bits, along with another signal which indicates when a key has been pressed, are available to most programs as the contents of a memory location. Programs can read the current state of the keyboard by reading the contents of this location. When you press a key on the keyboard, the value in this location becomes 128 or greater, and the particular value it assumes is the numeric code for the character which was typed. Table 3 on page 8 shows the ASCII characters and their associated numeric codes. The location will hold this one value until you press another key, or until your program tells the memory location to forget the character it's holding.

Once your program has accepted and understood a keypress, it should tell the keyboard's memory location to "release" the character it is holding and prepare to receive a new one. Your program can do this by referencing another memory location. When you reference this other location, the value contained in the first location will drop below 128. This value will stay low until you press another key. This action is called "clearing the keyboard strobe". Your program can either read or write to the special memory location; the data which are written to or read from that location are irrelevant. It is the mere *reference* to the location which clears the keyboard strobe. Once you have cleared the keyboard strobe, you can still recover the code for the key which was last pressed by adding 128 (hexadecimal \$80) to the value in the keyboard location.

These are the special memory locations used by the keyboard:

Table 1: Keyboard Special Locations			
Location:			Description
Hex	Decimal		
\$C000	49152	-16384	Keyboard Data
\$C010	49168	-16368	Clear Keyboard Strobe

The [RESET] key at the upper right-hand corner does not generate an ASCII code, but instead is directly connected to the microprocessor. When this key is pressed, all processing stops. When the key is released, the computer starts a reset cycle. See page 36 for a description of the RESET

function.

The **CTRL** and **SHIFT** keys generate no codes by themselves, but only alter the codes produced by other keys.

The **REPT** key, if pressed alone, produces a duplicate of the last code that was generated. If you press and hold down the **REPT** key while you are holding down a character key, it will act as if you were pressing that key repeatedly at a rate of 10 presses each second. This repetition will cease when you release either the character key or **REPT**.

The POWER light at the lower left-hand corner is an indicator lamp to show when the power to the Apple is on.

Table 2: Keys and Their Associated ASCII Codes

Key	Alone	CTRL	SHIFT	Both	Key	Alone	CTRL	SHIFT	Both
space	\$A0	\$A0	\$A0	\$A0	RETURN	\$8D	\$8D	\$8D	\$8D
Ø	\$B0	\$B0	\$B0	\$B0	G	\$C7	\$87	\$C7	\$87
1!	\$B1	\$B1	\$A1	\$A1	H	\$C8	\$88	\$C8	\$88
2"	\$B2	\$B2	\$A2	\$A2	I	\$C9	\$89	\$C9	\$89
3#	\$B3	\$B3	\$A3	\$A3	J	\$CA	\$8A	\$CA	\$8A
4\$	\$B4	\$B4	\$A4	\$A4	K	\$CB	\$8B	\$CB	\$8B
5%	\$B5	\$B5	\$A5	\$A5	L	\$CC	\$8C	\$CC	\$8C
6&	\$B6	\$B6	\$A6	\$A6	M	\$CD	\$8D	\$DD	\$9D
7'	\$B7	\$B7	\$A7	\$A7	N~	\$CE	\$8E	\$DE	\$9E
8(\$B8	\$B8	\$A8	\$A8	O	\$CF	\$8F	\$CF	\$8F
9)	\$B9	\$B9	\$A9	\$A9	P@	\$D0	\$90	\$C0	\$80
:*	\$BA	\$BA	\$AA	\$AA	Q	\$D1	\$91	\$D1	\$91
:+	\$BB	\$BB	\$AB	\$AB	R	\$D2	\$92	\$D2	\$92
,<	\$AC	\$AC	\$BC	\$BC	S	\$D3	\$93	\$D3	\$93
-=	\$AD	\$AD	\$BD	\$BD	T	\$D4	\$94	\$D4	\$94
.>	\$AE	\$AE	\$BE	\$BE	U	\$D5	\$95	\$D5	\$95
/?	\$AF	\$AF	\$BF	\$BF	V	\$D6	\$96	\$D6	\$96
A	\$C1	\$81	\$C1	\$81	W	\$D7	\$97	\$D7	\$97
B	\$C2	\$82	\$C2	\$82	X	\$D8	\$98	\$D8	\$98
C	\$C3	\$83	\$C3	\$83	Y	\$D9	\$99	\$D9	\$99
D	\$C4	\$84	\$C4	\$84	Z	\$DA	\$9A	\$DA	\$9A
E	\$C5	\$85	\$C5	\$85	→	\$88	\$88	\$88	\$88
F	\$C6	\$86	\$C6	\$86	→	\$95	\$95	\$95	\$95
					ESC	\$9B	\$9B	\$9B	\$9B

All codes are given in hexadecimal. To find the decimal equivalents, use Table 3.

Table 3: The ASCII Character Set

Decimal:	128	144	160	176	192	208	224	240
Hex:	\$80	\$90	\$A0	\$B0	\$C0	\$D0	\$E0	\$F0
0 \$0	nul	dle		0	@	P	p	
1 \$1	soh	dcl	!	1	A	Q	a	q
2 \$2	stx	dc2	"	2	B	R	b	r
3 \$3	etx	dc3	#	3	C	S	c	s
4 \$4	eot	dc4	\$	4	D	T	d	t
5 \$5	enq	nak	%	5	E	U	e	u
6 \$6	ack	syn	&	6	F	V	f	v
7 \$7	bel	etb	'	7	G	W	g	w
8 \$8	bs	can	(8	H	X	h	x
9 \$9	ht	em)	9	I	Y	i	y
10 \$A	lf	sub	*	:	J	Z	j	z
11 \$B	vt	esc	+	;	K	[k	{
12 \$C	ff	fs	,	<	L	\	l	
13 \$D	cr	gs	-	=	M]	m	-
14 \$E	so	rs	.	>	N	^	n	
15 \$F	si	us	/	?	O	_	o	rub

Groups of two and three lower case letters are abbreviations for standard ASCII control characters.

Not all the characters listed in this table can be generated by the keyboard. Specifically, the characters in the two rightmost columns (the lower case letters), the symbols [(left square bracket), \ (backslash), _ (underscore), and the control characters "fs", "us", and "rub", are not available on the Apple keyboard.

The decimal or hexadecimal value for any character in the above table is the sum of the decimal or hexadecimal numbers appearing at the top of the column and the left side of the row in which the character appears.

THE APPLE VIDEO DISPLAY

The Apple Video Display

Display type:	Memory mapped into system RAM
Display modes:	Text, Low-Resolution Graphics, High-Resolution Graphics
Text capacity:	960 characters (24 lines, 40 columns)
Character type:	5 × 7 dot matrix
Character set:	Upper case ASCII, 64 characters
Character modes:	Normal, Inverse, Flashing
Graphics capacity:	1,920 blocks (Low-Resolution) in a 40 by 48 array 53,760 dots (High-Resolution) in a 280 by 192 array
Number of colors:	16 (Low-Resolution Graphics) 6 (High-Resolution Graphics)

THE VIDEO CONNECTOR

In the right rear corner of the Apple II board, there is a metal connector marked "VIDEO". This connector allows you to attach a cable between the Apple and a closed-circuit video monitor. One end of the connecting cable should have a male RCA phono jack to plug into the Apple, and the other end should have a connector compatible with the particular device you are using. The signal that comes out of this connector on the Apple is similar to an Electronic Industries Association (EIA)-standard, National Television Standards Committee (NTSC)-compatible, positive composite color video signal. The level of this signal can be adjusted from zero to 1 volt peak by the small round potentiometer on the right edge of the board about three inches from the back of the board.

A non-adjustable, 2 volts peak version of the same video signal is available in two other places: on a single wire-wrap pin* on the left side of the board about two inches from the back of the board, and on one pin of a group of four similar pins also on the left edge near the back of the board. The other three pins in this group are connected to -5 volts, +12 volts, and ground. See page 97 for a full description of this auxiliary video connector.

* This pin is not present in Apple II systems with the Revision 0 board.

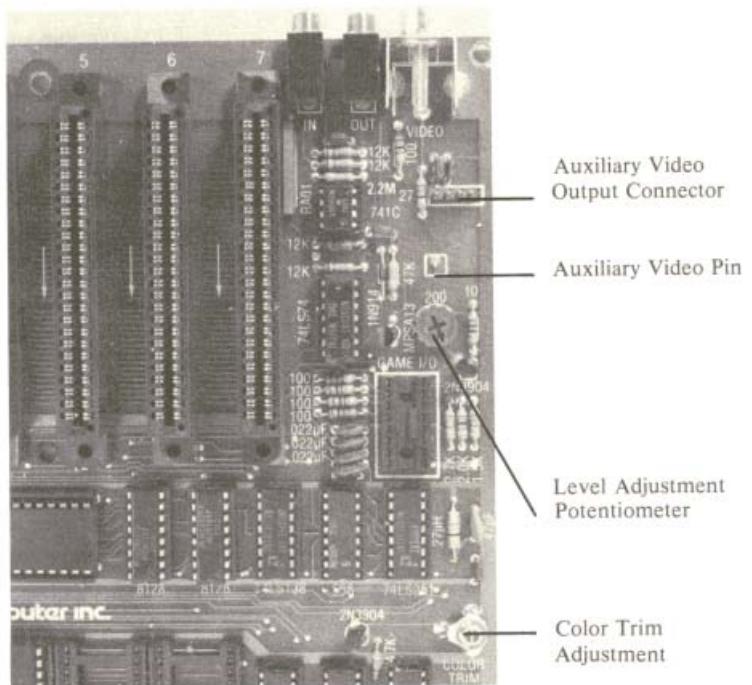


Photo 4. The Video Connectors and Potentiometer.

EURAPPLE (50 HZ) MODIFICATION

Your Apple can be modified to generate a video signal compatible with the CCIR standard used in many European countries. To make this modification, just cut the two X-shaped pads on the right edge of the board about nine inches from the back of the board, and solder together the three O-shaped pads in the same locations (see photo 5). You can then connect the video connector of your Apple to a European standard closed-circuit black-and-white or color video monitor. If you wish, you can obtain a "Eurocolor" encoder to convert the video signal into a PAL or SECAM standard color television signal suitable for use with any European television receiver. The encoder is a small printed circuit board which plugs into the rightmost peripheral slot (slot 7) in your Apple and connects to the single auxiliary video output pin.

WARNING: This modification will void the warranty on your Apple and requires the installation of a different main crystal. This modification is not for beginners.

SCREEN FORMAT

Three different kinds of information can be shown on the video display to which your Apple is connected:

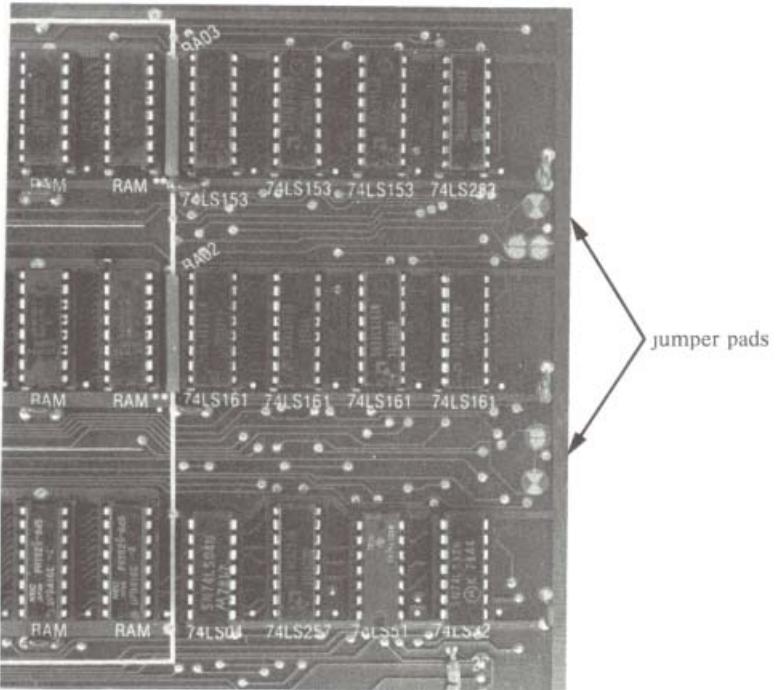


Photo 5. Eurapple (50 hz) Jumper Pads.

- 1) **Text.** The Apple can display 24 lines of numbers, special symbols, and upper-case letters with 40 of these characters on each line. These characters are formed in a dot matrix 7 dots high and 5 dots wide. There is a one-dot wide space on either side of the character and a one-dot high space above each line.
- 2) **Low-Resolution Graphics.** The Apple can present 1,920 colored squares in an array 40 blocks wide and 48 blocks high. The color of each block can be selected from a set of sixteen different colors. There is no space between blocks, so that any two adjacent blocks of the same color look like a single, larger block.
- 3) **High-Resolution Graphics.** The Apple can also display colored dots on a matrix 280 dots wide and 192 dots high. The dots are the same size as the dots which make up the Text characters. There are six colors available in the High-Resolution Graphics mode: black, white, red, blue, green, and violet.* Each dot on the screen can be either black, white, or a color, although not all colors are available for every dot.

When the Apple is displaying a particular type of information on the screen, it is said to be in that particular "mode". Thus, if you see words and numbers on the screen, you can reasonably be assured that your Apple is in Text mode. Similarly, if you see a screen full of multicolored blocks, your computer is probably in Low-Resolution Graphics mode. You can also have a four-line "caption" of text at the bottom of either type of graphics screen. These four lines replace

* For Apples with Revision 0 boards, there are four colors: black, white, green, and violet.

the lower 8 rows of blocks in Low-Resolution Graphics, leaving a 40 by 40 array. In High-Resolution Graphics, they replace the bottom 32 rows of dots, leaving a 280 by 160 matrix. You can use these "mixed modes" to display text and graphics simultaneously, but there is no way to display both graphics modes at the same time.

SCREEN MEMORY

The video display uses information in the system's RAM memory to generate its display. The value of a single memory location controls the appearance of a certain, fixed object on the screen. This object can be a character, two stacked colored blocks, or a line of seven dots. In Text and Low-Resolution Graphics mode, an area of memory containing 1,024 locations is used as the source of the screen information. Text and Low-Resolution Graphics share this memory area. In High-Resolution Graphics mode, a separate, larger area (8,192 locations) is needed because of the greater amount of information which is being displayed. These areas of memory are usually called "pages". The area reserved for High-Resolution Graphics is sometimes called the "picture buffer" because it is commonly used to store a picture or drawing.

SCREEN PAGES

There are actually *two* areas from which each mode can draw its information. The first area is called the "primary page" or "Page 1". The second area is called the "secondary page" or "Page 2" and is an area of the same size immediately following the first area. The secondary page is useful for storing pictures or text which you want to be able to display instantly. A program can use the two pages to perform animation by drawing on one page while displaying the other and suddenly flipping pages.

Text and Low-Resolution Graphics share the same memory range for the secondary page, just as they share the same range for the primary page. Both mixed modes which were described above are also available on the secondary page, but there is no way to mix the two pages on the same screen.

Table 4: Video Display Memory Ranges					
Screen	Page	Begins at:		Ends at:	
		Hex	Decimal		
Text/Lo-Res	Primary	\$400	1024	\$7FF	2047
	Secondary	\$800	2048	\$BFF	3071
Hi-Res	Primary	\$2000	8192	\$3FFF	16383
	Secondary	\$4000	16384	\$5FFF	24575

SCREEN SWITCHES

The devices which decide between the various modes, pages, and mixes are called "soft switches". They are switches because they have two positions (for example: on or off, text or graphics) and they are called "soft" because they are controlled by the software of the computer.

A program can "throw" a switch by referencing the special memory location for that switch. The data which are read from or written to the location are irrelevant; it is the *reference to the address* of the location which throws the switch.

There are eight special memory locations which control the setting of the soft switches for the screen. They are set up in pairs; when you reference one location of the pair you turn its corresponding mode "on" and its companion mode "off". The pairs are:

Table 5: Screen Soft Switches			
Location:	Hex	Decimal	Description:
\$C050	49232	-16304	Display a GRAPHICS mode.
\$C051	49233	-16303	Display TEXT mode.
\$C052	49234	-16302	Display all TEXT or GRAPHICS.
\$C053	49235	-16301	Mix TEXT and a GRAPHICS mode.*
\$C054	49236	-16300	Display the Primary page (Page 1).
\$C055	49237	-16299	Display the Secondary page (Page 2).
\$C056	49238	-16298	Display LO-RES GRAPHICS mode.*
\$C057	49239	-16297	Display HI-RES GRAPHICS mode.*

There are ten distinct combinations of these switches:

Table 6: Screen Mode Combinations							
Primary Page		Secondary Page		Screen		Switches	
All Text	\$C054	\$C051	All Text	\$C055	\$C051		
All Lo-Res	\$C054	\$C056	All Lo-Res	\$C055	\$C056		
Graphics	\$C052	\$C050	Graphics	\$C052	\$C050		
All Hi-Res	\$C054	\$C057	All Hi-Res	\$C055	\$C057		
Graphics	\$C052	\$C050	Graphics	\$C052	\$C050		
Mixed Text	\$C054	\$C056	Mixed Text	\$C055	\$C056		
and Lo-Res	\$C053	\$C050	and Lo-Res	\$C053	\$C050		
Mixed Text	\$C054	\$C057	Mixed Text	\$C055	\$C057		
and Hi-Res	\$C053	\$C050	and Hi-Res	\$C053	\$C050		

(Those of you who are learned in the ways of binary will immediately cry out, "Where's the other six?!", knowing full well that with 4 two-way switches there are indeed *sixteen* possible combinations. The answer to the mystery of the six missing modes lies in the TEXT/GRAFICS switch. When the computer is in Text mode, it can also be in one of six combinations of the Lo-Res/Hi-Res graphics mode, "mix" mode, or page selection. But since the Apple is displaying text, these different graphics modes are invisible.)

To set the Apple into one of these modes, a program needs only to refer to the addresses of the memory locations which correspond to the switches that set that mode. Machine language programs should use the hexadecimal addresses given above; BASIC programs should PEEK or POKE their decimal equivalents (given in Table 5, "Screen Soft Switches", above). The switches may be thrown in any order; however, when switching into one of the Graphics modes, it is helpful to throw the TEXT/GRAFICS switch last. All the other changes in mode will then take place invisibly behind the text, so that when the Graphics mode is set, the finished graphics

* These modes are only visible if the "Display GRAPHICS" switch is "on".



screen appears all at once.

THE TEXT MODE

In the Text mode, the Apple can display 24 lines of characters with up to 40 characters on each line. Each character on the screen represents the contents of one memory location from the memory range of the page being displayed. The character set includes the 26 upper-case letters, the 10 digits, and 28 special characters for a total of 64 characters. The characters are formed in a dot matrix 5 dots wide and 7 dots high. There is a one-dot wide space on both sides of each character to separate adjacent characters and a one-dot high space above each line of characters to separate adjacent lines. The characters are normally formed with white dots on a dark background; however, each character on the screen can also be displayed using dark dots on a white background or alternating between the two to produce a flashing character. When the Video Display is in Text mode, the video circuitry in the Apple turns off the color burst signal to the television monitor, giving you a clearer black-and-white display.*

The area of memory which is used for the primary text page starts at location number 1024 and extends to location number 2047. The secondary screen begins at location number 2048 and extends up to location 3071. In machine language, the primary page is from hexadecimal address \$400 to address \$7FF; the secondary page is from \$800 to \$BFF. Each of these pages is 1,024 bytes long. Those of you intrepid enough to do the multiplication will realize that there are only 960 characters displayed on the screen. The remaining 64 bytes in each page which are not displayed on the screen are used as temporary storage locations by programs stored in PROM on Apple Intelligent Interface® peripheral boards (see page 82).

Photo 6 shows the sixty-four characters available on the Apple's screen.



Photo 6. The Apple Character Set.

Table 7 gives the decimal and hexadecimal codes for the 64 characters in normal, inverse, and flashing display modes.

* This feature is not present on the Revision 0 board.

Table 7: ASCII Screen Characters

Decimal	Hex	Inverse			Flashing			(Control)			Normal			(Lowercase)		
		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224
0 \$00	@	P	Q	!	0	@	P	!	0	@	P	!	0	@	P	!
1 \$10	A	R	S	#	3	C	S	#	3	C	S	#	3	C	S	!
2 \$20	B	T	U	%	5	E	U	%	5	E	U	%	5	E	U	%
3 \$30	C	V	&	6	F	V	&	6	F	V	&	6	F	V	&	6
4 \$40	D	W	*	7	G	W	*	7	G	W	*	7	G	W	*	7
5 \$50	E	X	(8	H	X	(8	H	X	(8	H	X	(8
6 \$60	F	Y)	9	I	Y)	9	I	Y)	9	I	Y)	9
7 \$70	G	Z	*	:	J	Z	*	:	J	Z	*	:	J	Z	*	:
8 \$80	H	K	+	:	K	L	+	:	K	L	+	:	K	L	+	:
9 \$90	I	L	\	:	<	M	\	-	<	L	\	-	<	L	\	-
10 \$A0	J	M	-	=	N	-	>	N	=	M	-	>	N	-	>	=
11 \$B0	K	O	/	?	O	-	/	?	O	-	/	?	O	-	/	?
12 \$C0	L	-	>	O	-	/	?	O	-	/	?	O	-	/	?	O
13 \$D0	M	-	>	O	-	/	?	O	-	/	?	O	-	/	?	O
14 \$E0	N	-	>	O	-	/	?	O	-	/	?	O	-	/	?	O
15 \$F0	O	-	>	O	-	/	?	O	-	/	?	O	-	/	?	O

Table 7. ASCII Screen Character Set

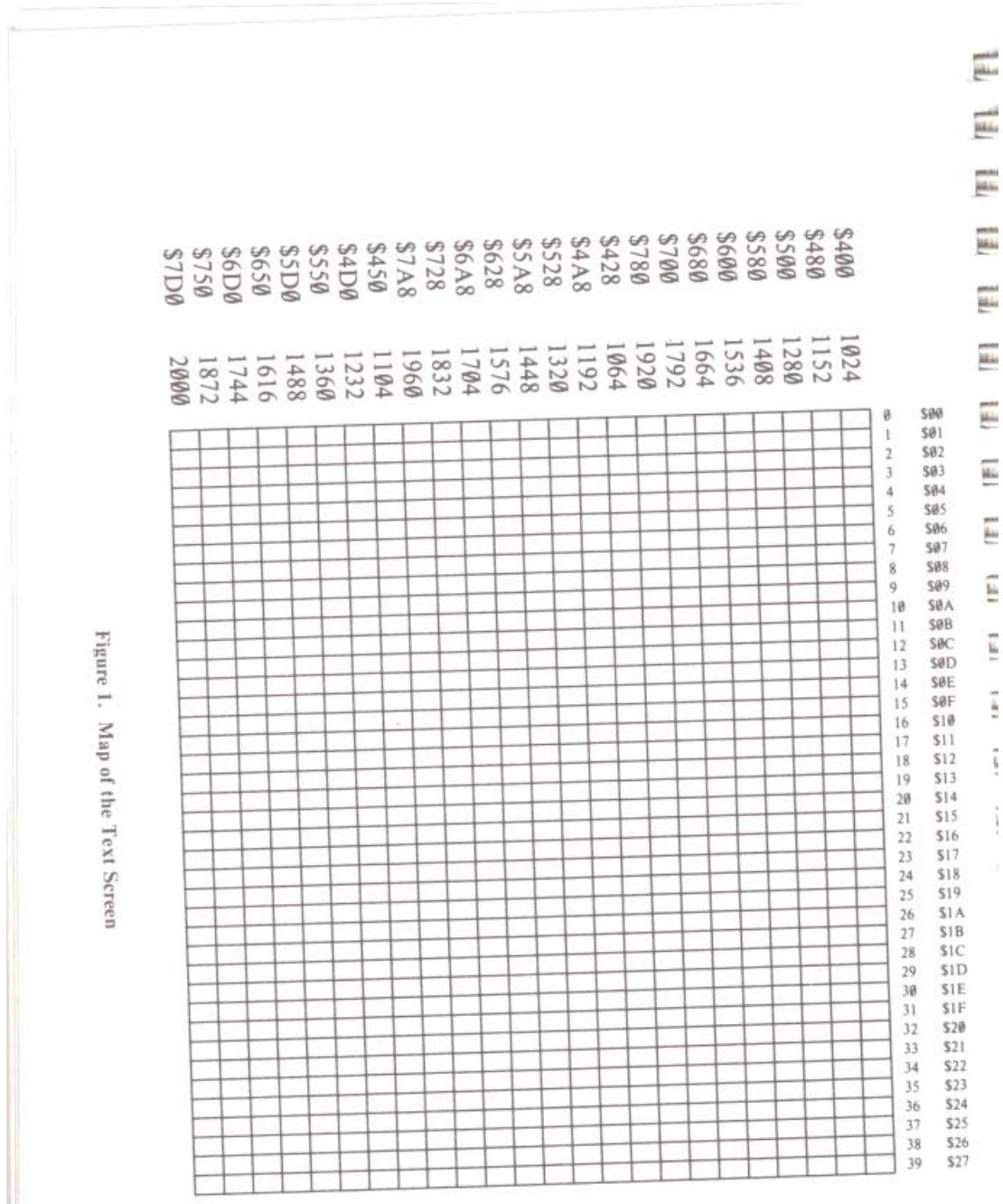


Figure 1. Map of the Text Screen

Figure 1 is a map of the Apple's display in Text mode, with the memory location addresses for each character position on the screen.

THE LOW-RESOLUTION GRAPHICS (LO-RES) MODE

In the Low-Resolution Graphics mode, the Apple presents the contents of the same 1,024 locations of memory as is in the Text mode, but in a different format. In this mode, each byte of memory is displayed not as an ASCII character, but as two colored blocks, stacked one atop the other. The screen can show an array of blocks 40 wide and 48 high. Each block can be any of sixteen colors. On a black-and-white television set, the colors appear as patterns of grey and white dots.

Since each byte in the page of memory for Low-Resolution Graphics represents two blocks on the screen, stacked vertically, each byte is divided into two equal sections, called (appropriately enough) "nybbles". Each nibble can hold a value from zero to 15. The value which is in the lower nibble of the byte determines the color for the upper block of that byte on the screen, and the value which is in the upper nibble determines the color for the lower block on the screen. The colors are numbered zero to 15, thus:

Table 8: Low-Resolution Graphics Colors					
Decimal	Hex	Color	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

(Colors may vary from television to television, particularly on those without hue controls. You can adjust the tint of the colors by adjusting the COLOR TRIM control on the right edge of the Apple board.)

So, a byte containing the hexadecimal value \$D8 would appear on the screen as a brown block on top of a yellow block. Using decimal arithmetic, the color of the lower block is determined by the quotient of the value of the byte divided by 16; the color of the upper block is determined by the remainder.

Figure 2 is a map of the Apple's display in Low-Resolution Graphics mode, with the memory location addresses for each block on the screen.

Since the Low-Resolution Graphics screen displays the same area in memory as is used for the Text screen, interesting things happen if you switch between the Text and Low-Resolution Graphics modes. For example, if the screen is in the Low-Resolution Graphics mode and is full of colored blocks, and then the TEXT/GRAFICS screen switch is thrown to the Text mode, the screen will be filled with seemingly random text characters, sometimes inverse or flashing. Similarly, a screen full of text when viewed in Low-Resolution Graphics mode appears as long horizontal grey, pink, green or yellow bars separated by randomly colored blocks.

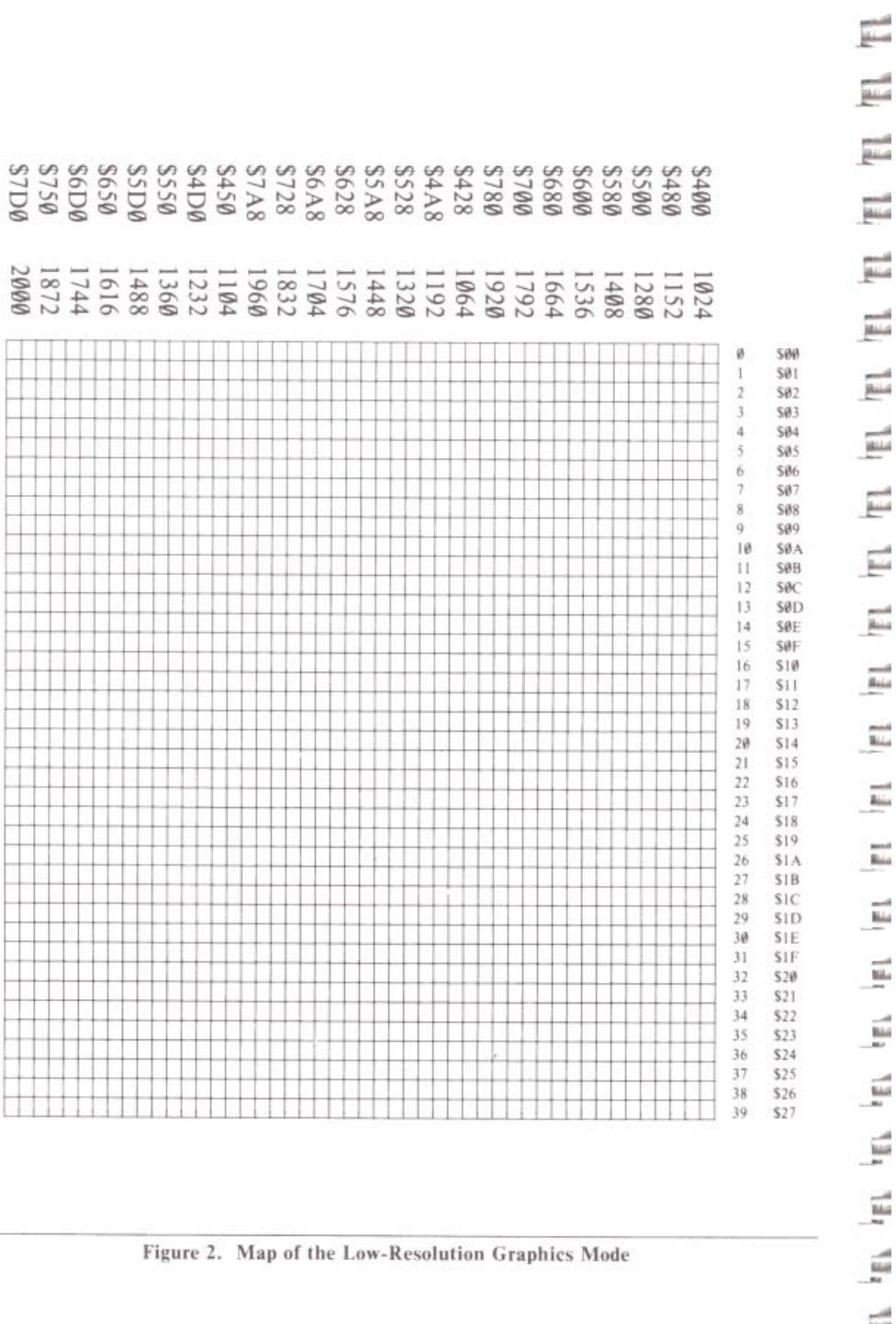


Figure 2. Map of the Low-Resolution Graphics Mode

THE HIGH-RESOLUTION GRAPHICS (HI-RES) MODE

The Apple has a second type of graphic display, called High-Resolution Graphics (or sometimes "Hi-res"). When your Apple is in the High-Resolution Graphics mode, it can display 53,760 dots in a matrix 280 dots wide and 192 dots high. The screen can display black, white, violet, green, red, and blue dots, although there are some limitations concerning the color of individual dots.

The High-Resolution Graphics mode takes its data from an 8,192-byte area of memory, usually called a "picture buffer". There are two separate picture buffers: one for the primary page and one for the secondary page. Both of these buffers are independent of and separate from the memory areas used for Text and Low-Resolution Graphics. The primary page picture buffer for the High-Resolution Graphics mode begins at memory location number 8192 and extends up to location number 16383; the secondary page picture buffer follows on the heels of the first at memory location number 16384, extending up to location number 24575. For those of you with sixteen fingers, the primary page resides from \$2000 to \$3FFF and the secondary page follows in succession at \$4000 to \$5FFF. If your Apple is equipped with 16K (16,384 bytes) or less of memory, then the secondary page is inaccessible to you; if its memory size is less than 16K, then the entire High-Resolution Graphics mode is unavailable to you.

Each dot on the screen represents one bit from the picture buffer. Seven of the eight bits in each byte are displayed on the screen, with the remaining bit used to select the colors of the dots in that byte. Forty bytes are displayed on each line of the screen. The least significant bit (first bit) of the first byte in the line is displayed on the left edge of the screen, followed by the second bit, then the third, etc. The most significant (eighth) bit is not displayed. Then follows the first bit of the next byte, and so on. A total of 280 dots are displayed on each of the 192 lines of the screen.

On a black-and-white monitor or TV set, the dots whose corresponding bits are "on" (or equal to 1) appear white; the dots whose corresponding bits are "off" (or equal to 0) appear black. On a color monitor or TV, it is not so simple. If a bit is "off", its corresponding dot will always be black. If a bit is "on", however, its color will depend upon the *position* of that dot on the screen. If the dot is in the leftmost column on the screen, called "column 0", or in any even-numbered column, then it will appear violet. If the dot is in the rightmost column (column 279) or any odd-numbered column, then it will appear green. If two dots are placed side-by-side, they will both appear white. If the undisplayed bit of a byte is turned on, then the colors blue and red are substituted for violet and green, respectively.* Thus, there are six colors available in the High-Resolution Graphics mode, subject to the following limitations:

- 1) Dots in even columns must be black, violet, or blue.
- 2) Dots in odd columns must be black, green, or red.
- 3) Each byte must be either a violet/green byte or a blue/red byte. It is not possible to mix green and blue, green and red, violet and blue, or violet and red in the same byte.

* On Revision 0 Apple boards, the colors red and blue are unavailable and the setting of the eighth bit is irrelevant.

- 4) Two colored dots side by side always appear white, even if they are in different bytes.
- 5) On European-modified Apples, these rules apply but the colors generated in the High-Resolution Graphics mode may differ.

Figure 3 shows the Apple's display screen in High-Resolution Graphics mode with the memory addresses of each line on the screen.

OTHER INPUT/OUTPUT FEATURES

Apple Input/Output Features

Inputs:	Cassette Input Three One-bit Digital Inputs Four Analog Inputs
Outputs:	Cassette Output Built-In Speaker Four "Annunciator" Outputs Utility Strobe Output

THE SPEAKER

Inside the Apple's case, on the left side under the keyboard, is a small 8 ohm speaker. It is connected to the internal electronics of the Apple so that a program can cause it to make various sounds.

The speaker is controlled by a soft switch. The switch can put the paper cone of the speaker in two positions: "in" and "out". This soft switch is not like the soft switches controlling the various video modes, but is instead a *toggle* switch. Each time a program references the memory address associated with the speaker switch, the speaker will change state: change from "in" to "out" or vice-versa. Each time the state is changed, the speaker produces a tiny "click". By referencing the address of the speaker switch frequently and continuously, a program can generate a steady tone from the speaker.

The soft switch for the speaker is associated with memory location number 49200. Any reference to this address (or the equivalent addresses -16336 or hexadecimal \$C030) will cause the speaker to emit a click.

A program can "reference" the address of the special location for the speaker by performing a "read" or "write" operation to that address. The data which are read or written are irrelevant, as it is the *address* which throws the switch. Note that a "write" operation on the Apple's 6502 microprocessor actually performs a "read" before the "write", so that if you use a "write" operation to flip any soft switch, you will actually throw that switch *twice*. For toggle-type soft switches, such as the speaker switch, this means that a "write" operation to the special location

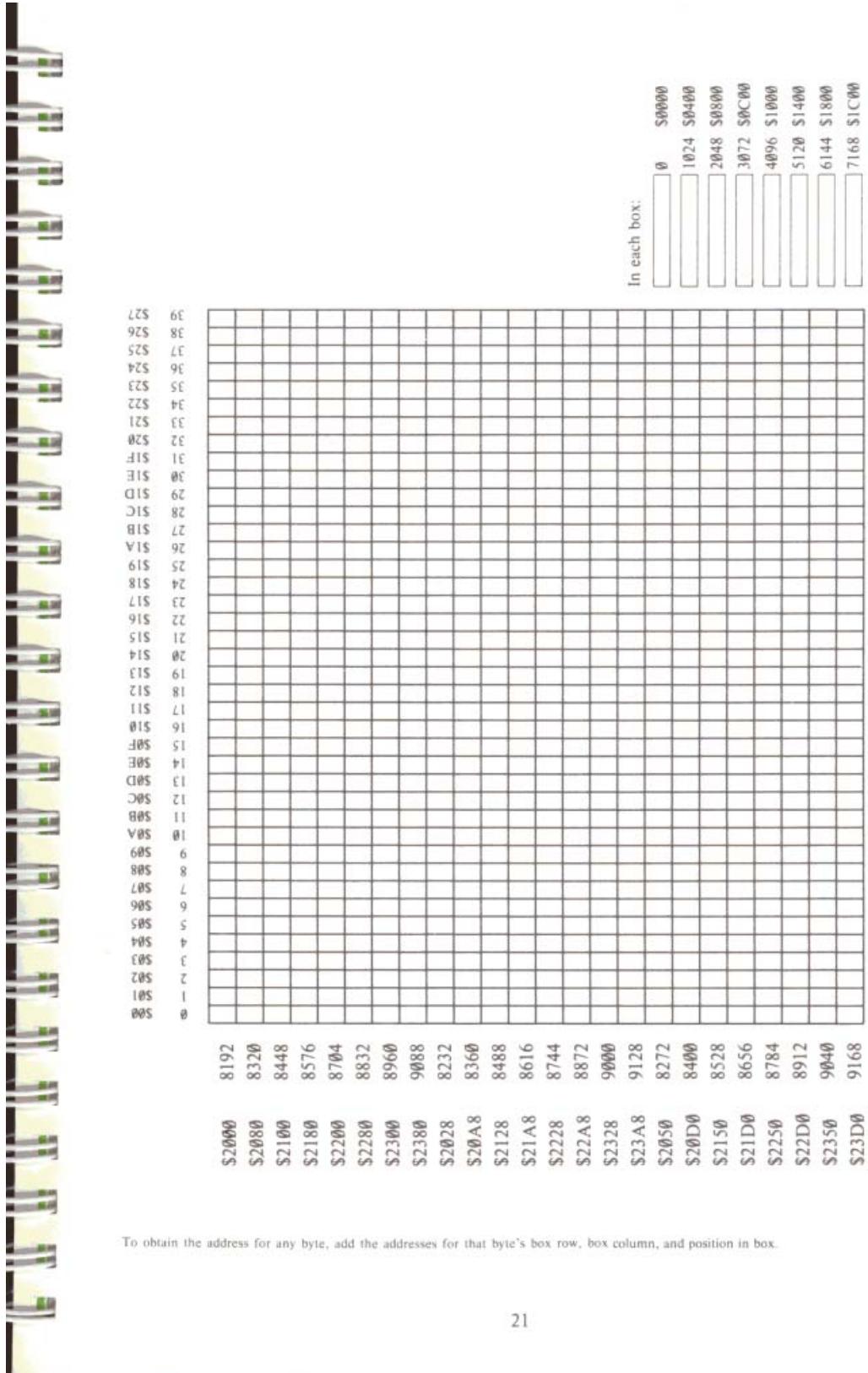


Figure 3. Map of the High-Resolution Graphics Screen

controlling the switch will leave the switch in the same state it was in before the operation was performed.

THE CASSETTE INTERFACE

On the back edge of the Apple's main board, on the right side next to the VIDEO connector, are two small black packages labelled "IN" and "OUT". These are miniature phone jacks into which you can plug a cable which has a pair of miniature phono plugs on each end. The other end of this cable can be connected to a standard cassette tape recorder so that your Apple can save information on audio cassette tape and read it back again.

The connector marked "OUT" is wired to yet another soft switch on the Apple board. This is another toggle switch, like the speaker switch (see above). The soft switch for the cassette output plug can be toggled by referencing memory location number 49184 (or the equivalent -16352 or hexadecimal SC020). Referencing this location will make the voltage on the OUT connector swing from zero to 25 millivolts (one fortieth of a volt), or return from 25 millivolts back to zero. If the other end of the cable is plugged into the MICROPHONE input of a cassette tape recorder which is recording onto a tape, this will produce a tiny "click" on the recording. By referencing the memory location associated with the cassette output soft switch repeatedly and frequently, a program can produce a tone on the recording. By varying the pitch and duration of this tone, information may be encoded on a tape and saved for later use. Such a program to encode data on a tape is included in the System Monitor and is described on page 46.

Be forewarned that if you attempt to flip the soft switch for the cassette output by writing to its special location, you will actually generate *two* "clicks" on the recording. The reason for this is mentioned in the description of the speaker (above). You should only use "read" operations when toggling the cassette output soft switch.

The other connector, marked "IN", can be used to "listen" to a cassette tape recording. Its main purpose is to provide a means of listening to tones on the tape, decoding them into data, and storing them in memory. Thus, a program or data set which was stored on cassette tape may be read back in and used again.

The input circuit takes a 1 volt (peak-to-peak) signal from the cassette recorder's EARPHONE jack and converts it into a string of ones and zeroes. Each time the signal applied to the input circuit swings from positive to negative, or vice-versa, the input circuit changes state: if it was sending ones, it will start sending zeroes, and vice versa. A program can inspect the state of the cassette input circuit by looking at memory location number 49248 or the equivalents -16288 or hexadecimal SC060. If the value which is read from this location is greater than or equal to 128, then the state is a "one"; if the value in the memory location is less than 128, then the state is a "zero". Although BASIC programs can read the state of the cassette input circuit, the speed of a BASIC program is usually much too slow to be able to make any sense out of what it reads. There is, however, a program in the System Monitor which will read the tones on a cassette tape and decode them. This is described on page 47.

THE GAME I/O CONNECTOR

The purpose of the Game I/O connector is to allow you to connect special input and output devices to heighten the effect of programs in general, and specifically, game programs. This connector allows you to connect three one-bit inputs, four one-bit outputs, a data strobe, and four analog inputs to the Apple, all of which can be controlled by your programs. Supplied with your Apple is a pair of Game Controllers which are connected to cables which plug into the Game I/O connector. The two rotary dials on the Controllers are connected to two analog inputs on the Connector; the two pushbuttons are connected to two of the one-bit inputs.

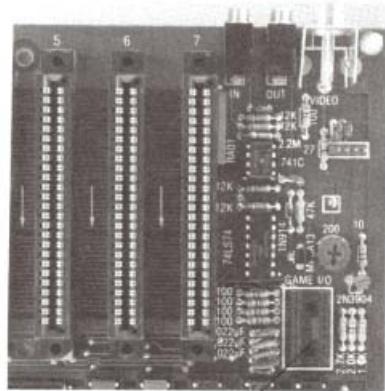


Photo 7. The Game I/O Connector.

ANNUNCIATOR OUTPUTS

The four one-bit outputs are called "annunciators". Each annunciator output can be used as an input to some other electronic device, or the annunciator outputs can be connected to circuits to drive lamps, relays, speakers, etc.

Each annunciator is controlled by a soft switch. The addresses of the soft switches for the annunciators are arranged into four pairs, one pair for each annunciator. If you reference the first address in a pair, you turn the output of its corresponding annunciator "off"; if you reference the second address in the pair, you turn the annunciator's output "on". When an annunciator is

"off", the voltage on its pin on the Game I/O Connector is near 0 volts; when an annunciator is "on", the voltage is near 5 volts. There are no inherent means to determine the current setting of an annunciator bit. The annunciator soft switches are:

Table 9: Annunciator Special Locations				
Ann.	State	Address:		
		Decimal	Hex	
0	off	49240	-16296	\$C058
	on	49241	-16295	\$C059
1	off	49242	-16294	\$C05A
	on	49243	-16293	\$C05B
2	off	49244	-16292	\$C05C
	on	49245	-16291	\$C05D
3	off	49246	-16290	\$C05E
	on	49247	-16289	\$C05F

ONE-BIT INPUTS

The three one-bit inputs can each be connected to either another electronic device or to a push-button. You can read the state of any of the one-bit inputs from a machine language or BASIC program in the same manner as you read the Cassette Input, above. The locations for the three one-bit inputs have the addresses 49249 through 49251 (-16287 through -16285 or hexadecimal \$C061 through \$C063).

ANALOG INPUTS

The four analog inputs can be connected to 150K Ohm variable resistors or potentiometers. The variable resistance between each input and the +5 volt supply is used in a one-shot timing circuit. As the resistance on an input varies, the timing characteristics of its corresponding timing circuit change accordingly. Machine language programs can sense the changes in the timing loops and obtain a numerical value corresponding to the position of the potentiometer.

Before a program can start to read the setting of a potentiometer, it must first reset the timing circuits. Location number 49264 (-16272 or hexadecimal \$C070) does just this. When you reset the timing circuits, the values contained in the four locations 49252 through 49255 (-16284 through -16281 or \$C064 through \$C067) become greater than 128 (their high bits are set). Within 3.060 milliseconds, the values contained in these four locations should drop below 128. The exact time it takes for each location to drop in value is directly proportional to the setting of the game paddle associated with that location. If the potentiometers connected to the analog inputs have a greater resistance than 150K Ohms, or there are no potentiometers connected, then the values in the game controller locations may never drop to zero.

STROBE OUTPUT

There is an additional output, called **C040 STROBE**, which is normally +5 volts but will drop to zero volts for a duration of one-half microsecond under the control of a machine language or BASIC program. You can trigger this "strobe" by referring to location number 49216 (-16320 or SC04F). Be aware that if you perform a "write" operation to this location, you will trigger the strobe *twice* (see a description of this phenomenon in the section on the Speaker).

Table 10: Input/Output Special Locations			
Function:	Address: Decimal	Hex	Read/Write
Speaker	49200	-16336	SC030
Cassette Out	49184	-16352	SC020
Cassette In	49256	-16288	SC060
Annunciators*	49240 through 49247	-16296 through -16289	SC058 through SC05F
Flag inputs	49249 49250 49251	-16287 -16286 -16285	SC061 SC062 SC063
Analog Inputs	49252 49253 49254 49255	-16284 -16283 -16282 -16281	SC064 SC065 SC066 SC067
Analog Clear	49264	-16272	SC070
Utility Strobe	49216	-16320	SC040

VARIETIES OF APPLES

There are a few variations on the basic Apple II computer. Some of the variations are revisions or modifications of the computer itself; others are changes to its operating software. These are the basic variations:

AUTOSTART ROM / MONITOR ROM

All Apple II Plus Systems include the Autostart Monitor ROM. All other Apple systems do not contain the Autostart ROM, but instead have the Apple System Monitor ROM. This version of the ROM lacks some of the features present in the Autostart ROM, but also has some features which are not present in that ROM. The main differences in the two ROMs are listed on the following pages.

* See the previous table.

- **Editing Controls.** The ESC-I, J, K, and M sequences, which move the cursor up, left, right, and down, respectively, are not available in the Old Monitor ROM.
- **Stop-List.** The Stop-List feature (invoked by a **CTRL S**), which allows you to introduce a pause into the output of most BASIC or machine language programs or listings, is not available in the Old Monitor ROM.
- **The RESET cycle.** When you first turn on your Apple or press **RESET**, the Old Monitor ROM will send you directly into the Apple System Monitor, instead of initiating a warm or cold start as described in "The RESET Cycle" on page 36.

The Old Monitor ROM does, however, support the STEP and TRACE debugging features of the System Monitor, described on page 51. The Autostart ROM does not recognize these Monitor commands.

REVISION 0 / REVISION 1 BOARD

The Revision 0 Apple II board lacks a few features found on the current Revision 1 version of the Apple II main board. To determine which version of the main board is in your Apple, open the case and look at the upper right-hand corner of the board. Compare what you see to Photo 4 on page 10. If your Apple does not have the single metal video connector pin between the four-pin video connector and the video adjustment potentiometer, then you have a Revision 0 Apple.

The differences between the Revision 0 and Revision 1 Apples are summarized below.

- **Color Killer.** When the Apple's Video Display is in Text mode, the Revision 0 Apple board leaves the color burst signal active on the video output circuit. This causes text characters to appear tinted or with colored fringes.
- **Power-on RESET.** Revision 0 Apple boards have no circuit to automatically initiate a RESET cycle when you turn the power on. Instead, you must press **RESET** once to start using your Apple.

Also, when you turn on the power to an Apple with a Revision 0 board, the keyboard will become active, as if you had typed a random character. When the Apple starts looking for input, it will accept this random character as if you had typed it. In order to erase this character, you should press **CTRL X** after you **RESET** your Apple when you turn on its power.

- **Colors in High-Resolution Graphics.** Apples with Revision 0 boards can generate only four colors in the High-Resolution Graphics mode: black, white, violet, and green. The high bit of each byte displayed on the Hi-Res screen (see page 19) is ignored.
- **24K Memory Map problem.** Systems with a Revision 0 Apple II board which contain 20K or 24K bytes of RAM memory appear to BASIC to have more memory than they actually do. See "Memory Organization", page 72, for a description of this problem.
- **50 Hz Apples.** The Revision 0 Apple II board does not have the pads and jumpers which you can cut and solder to convert the VIDEO OUT signal of your Apple to conform to the European PAL/SECAM television standard. It also lacks the third VIDEO connector, the single metal pin in front of the four-pin video connector.

- **Speaker and Cassette Interference.** On Apples with Revision Ø boards, any sound generated by the internal speaker will also appear as a signal on the Cassette Interface's OUT connector. If you leave the tape recorder in RECORD mode, then any sound generated by the internal speaker will also appear on the tape recording.
- **Cassette Input.** The input circuit for the Cassette Interface has been modified so that it will respond with more accuracy to a weaker input signal.

POWER SUPPLY CHANGES

In addition, some Apples have a version of the Apple Power Supply which will accept only a 110 volt power line input. These are not equipped with the voltage selector switch on the back of the supply.

THE APPLE II PLUS

The **Apple II Plus** is a standard Apple II computer with a Revision 1 board, an Autostart Monitor ROM, and the Applesoft II BASIC language in ROM in lieu of Apple Integer BASIC. European models of the Apple II Plus are equipped with a 110/220 volt power supply. The Apple Mini-Assembler, the Floating-Point Package, and the SWEET-16 interpreter, stored in the Integer BASIC ROMs, are not available on the Apple II Plus.



CHAPTER 2

CONVERSATION WITH APPLES

- 30 STANDARD OUTPUT
- 30 THE STOP-LIST FEATURE
- 31 BUT SOFT, WHAT LIGHT THROUGH YONDER WINDOW BREAKS!
(OR, THE TEXT WINDOW)
- 32 SEEING IT ALL IN BLACK AND WHITE
- 32 STANDARD INPUT
- 32 RDKEY
- 33 GETLN
- 34 ESCAPE CODES
- 36 THE RESET CYCLE
- 36 AUTOSTART ROM RESET
- 37 AUTOSTART ROM SPECIAL LOCATIONS
- 38 "OLD MONITOR" ROM RESET

Almost every program and language on the Apple needs some sort of input from the keyboard, and some way to print information on the screen. There is a set of subroutines stored in the Apple's ROM memory which handle most of the standard input and output from all programs and languages on the Apple.

The subroutines in the Apple's ROM which perform these input and output functions are called by various names. These names were given to the subroutines by their authors when they were written. The Apple itself does not recognize or remember the names of its own machine language subroutines, but it's convenient for us to call these subroutines by their given names.

STANDARD OUTPUT

The standard output subroutine is called COUT. COUT will display upper-case letters, numbers, and symbols on the screen in either Normal or Inverse mode. It will ignore control characters except RETURN, the bell character, the line feed character, and the backspace character.

The COUT subroutine maintains its own invisible "output cursor"** (the position at which the next character is to be placed). Each time COUT is called, it places one character on the screen at the current cursor position, replacing whatever character was there, and moves the cursor one space to the right. If the cursor is bumped off the right edge of the screen, then COUT shifts the cursor down to the first position on the next line. If the cursor passes the bottom line of the screen, the screen "scrolls" up one line and the cursor is set to the first position on the newly blank bottom line.

When a RETURN character is sent to COUT, it moves the cursor to the first position of the next line. If the cursor falls off the bottom of the screen, the screen scrolls as described above.

THE STOP-LIST FEATURE

When any program or language sends a RETURN code to COUT, COUT will take a quick peek at the keyboard. If you have typed a **CTRL S** since the last time COUT looked at the keyboard, then it will stop and wait for you to press another key. This is called the *Stop-List* feature.** When you press another key, COUT will then output the RETURN code and proceed with normal output. The code of the key which you press to end the Stop-List mode is ignored unless it is a **CTRL C**. If it is, then COUT passes this character code back to the program or language which is sending output. This allows you to terminate a BASIC program or listing by typing **CTRL C** while you are in Stop-List mode.

A line feed character causes COUT to move its mythical output cursor down one line without any horizontal motion at all. As always, moving beyond the bottom of the screen causes the screen to scroll and the cursor remains at its same position on a fresh bottom line.

A backspace character moves the imaginary cursor one space to the left. If the cursor is bumped off the left edge, it is reset to the rightmost position on the previous line. If there is no previous line (if the cursor was at the top of the screen), the screen does *not* scroll downwards, but instead

* From latin *cursus*, "runner"

** The Stop-list feature is not present on Apples without the Autostart ROM.

the cursor is placed again at the rightmost position on the top line of the screen.

When COUT is sent a “bell” character (CTRL G), it does not change the screen at all, but instead produces a tone from the speaker. The tone has a frequency of 100Hz and lasts for 1/10th of a second. The output cursor does not move for a bell character.

BUT SOFT, WHAT LIGHT THROUGH YONDER WINDOW BREAKS!

(OR, THE TEXT WINDOW)

In the above discussions of the various motions of the output cursor, the words “right”, “left”, “top”, and “bottom” mean the physical right, left, top, and bottom of the standard 40-character wide by 24-line tall screen. There is, however, a way to tell the COUT subroutine that you want it to use only a section of the screen, and not the entire 960-character display. This segregated section of the text screen is called a “window”. A program or language can set the positions of the top, bottom, left side, and width of the text window by storing those positions in four locations in memory. When this is done, the COUT subroutine will use the new positions to calculate the size of the screen. It will never print any text outside of this window, and when it must scroll the screen, it will only scroll the text within the window. This gives programs the power to control the placement of text, and to protect areas of the screen from being overwritten with new text.

Location number 32 (hexadecimal \$20) in memory holds the column position of the leftmost column in the window. This position is normally position 0 for the extreme left side of the screen. This number should never exceed 39 (hexadecimal \$27), the leftmost column on the text screen. Location number 33 (hexadecimal \$21) holds the width, in columns, of the cursor window. This number is normally 40 (hexadecimal \$28) for a full 40-character screen. Be careful that the sum of the window width and the leftmost window position does not exceed 40! If it does, it is possible for COUT to place characters in memory locations not on the screen, endangering your programs and data.

Location number 34 (hexadecimal \$22) contains the number of the top line of the text window. This is also normally 0, indicating the topmost line of the display. Location 35 (hexadecimal \$23) holds the number of the bottom line of the screen (plus one), thus normally 24 (hexadecimal \$18) for the bottommost line of the screen. When you change the text window, you should take care that you know the whereabouts of the output cursor, and that it will be inside the new window.

Table 11: Text Window Special Locations

Function:	Location: Decimal Hex	Minimum/Normal/Maximum Value Decimal Hex
Left Edge	32 \$20	0/0/39 \$0/\$0/\$17
Width	33 \$21	0/40/40 \$0/\$28/\$28
Top Edge	34 \$22	0/0/24 \$0/\$0/\$18
Bottom Edge	35 \$23	0/24/24 \$0/\$18/\$18

SEEING IT ALL IN BLACK AND WHITE

The COUT subroutine has the power to print what's sent to it in either Normal or Inverse text modes (see page 14). The particular form of its output is determined by the contents of location number *50* (hexadecimal \$32). If this location contains the value 255 (hexadecimal \$FF), then COUT will print characters in Normal mode; if the value is 63 (hexadecimal \$3F), then COUT will present its display in Inverse mode. Note that this mode change only affects the characters printed after the change has been made. Other values, when stored in location *50*, do unusual things: the value 127 prints letters in Flashing mode, but all other characters in Inverse; any other value in location *50* will cause COUT to ignore some or all of its normal character set.

Table 12: Normal/Inverse Control Values		
Value:	Effect:	
Decimal	Hex	
255	\$FF	COUT will display characters in Normal mode.
63	\$3F	COUT will display characters in Inverse mode.
127	\$7F	COUT will display letters in Flashing mode, all other characters in Inverse mode.

The Normal/Inverse "mask" location, as it is called, works by performing a logical "AND" between the bits contained in location *50* and the bits in each outgoing character code. Every bit in location *50* which is a logical "zero" will force the corresponding bit in the character code to become "zero" also, regardless of its former setting. Thus, when location *50* contains 63 (hexadecimal \$3F or binary 00111111), the top two bits of every output character code will be turned "off". This will place characters on the screen whose codes are all between 0 and 63. As you can see from the ASCII Screen Character Code table (Table 7 on page 15), all of these characters are in Inverse mode.

STANDARD INPUT

There are actually two subroutines which are concerned with the gathering of standard input: RDKEY, which fetches a single keystroke from the keyboard, and GETLN, which accumulates a number of keystrokes into a chunk of information called an *input line*.

RDKEY

The primary function of the RDKEY subroutine is to wait for the user to press a key on the keyboard, and then report back to the program which called it with the code for the key which was pressed. But while it does this, RDKEY also performs two other helpful tasks:

- 1). *Input Prompting.* When RDKEY is activated, the first thing it does is make visible the hidden output cursor. This accomplishes two things: it reminds the user that the Apple is waiting for a key to be pressed, and it also associates the input it wants with a particular place on the screen. In most cases, the input prompt appears near a word or phrase describing what is being requested by the particular program or language currently in use. The input cursor itself is a flashing representation of whatever character was at the position of the output cursor. Usually this is the blank character, so the input cursor most often appears to be a flashing square.

When the user presses a key, RDKEY dutifully removes the input cursor and returns the value of the key which was pressed to the program which requested it. Remember that the output cursor is just a position on the screen, but the input cursor is a flashing character on the screen. They usually move in tandem and are rarely separated from each other, but when the input cursor disappears, the output cursor is still active.

- 2). *Random Number Seeding.* While it waits for the user to press a key, RDKEY is continually adding 1 to a pair of numbers in memory. When a key is finally pressed, these two locations together represent a number from 0 to 65,535, the exact value of which is quite unpredictable. Many programs and languages use this number as the base of a random number generator. The two locations which are randomized during RDKEY are numbers 78 and 79 (hexadecimal \$4E and \$4F).

GETLN

The vast majority of input to the Apple is gathered into chunks called *input lines*. The subroutine in the Apple's ROM called GETLN requests an input line from the keyboard, and after getting one, returns to the program which called it. GETLN has many features and nuances, and it is good to be familiar with the services it offers.

When called, GETLN first prints a *prompting character*, or "prompt". The prompt helps you to identify which program has called GETLN requesting input. A prompt character of an asterisk (*) represents the System Monitor, a right caret (>) indicates Apple Integer BASIC, a right bracket () is the prompt for Applesoft II BASIC, and an exclamation point (!) is the hallmark of the Apple Mini-Assembler. In addition, the question-mark prompt (?) is used by many programs and languages to indicate that a user program is requesting input. From your (the user's) point of view, the Apple simply prints a prompt and displays an input cursor. As you type, the characters you type are printed on the screen and the cursor moves accordingly. When you press [RETURN], the entire line is sent off to the program or language you are talking to, and you get another prompt.

Actually, what really happens is that after the prompt is printed, GETLN calls RDKEY, which displays an input cursor. When RDKEY returns with a keycode, GETLN stores that keycode in an *input buffer* and prints it on the screen where the input cursor was. It then calls RDKEY again. This continues until the user presses [RETURN]. When GETLN receives a RETURN code from the keyboard, it sticks the RETURN code at the end of the input buffer, clears the remainder of the screen line the input cursor was on, and sends the RETURN code to COUT (see above). GETLN then returns to the program which called it. The program or language which requested input may now look at the entire line, all at once, as saved in the input buffer.

At any time while you are typing a line, you can type a [CTRL X] and cancel that entire line. GETLN will simply forget everything you have typed, print a backslash (\), skip to a new line, and display another prompt, allowing you to retype the line. Also, GETLN can handle a maximum of 255 characters in a line. If you exceed this limit, GETLN will cancel the entire line and you must start over. To warn you that you are approaching the limit, GETLN will sound a tone every keypress starting with the 249th character.

GETLN also allows you to edit and modify the line you are typing in order to correct simple typographical errors. A quick introduction to the standard editing functions and the use of the two arrow keys, [←] and [→], appears on pages 28-29 and 53-55 of the **Apple II BASIC Programming Manual**, or on pages 27-28, 52-53 and Appendix C of **The Applesoft Tutorial**, at least one

of which you should have received. Here is a short description of GETLN's editing features:

THE BACKSPACE (**⌫**) KEY

Each press of the backspace key makes GETLN "forget" one previous character in the input line. It also sends a backspace character to COUT (see above), making the cursor move back to the character which was deleted. At this point, a character typed on the keyboard will replace the deleted character both on the screen and in the input line. Multiple backspaces will delete successive characters; however, if you backspace over more characters than you have typed, GETLN will forget the entire line and issue another prompt.

THE RETYPE (**⤓**) KEY

Pressing the retype key has exactly the same effect as typing the character which is under the cursor. This is extremely useful for re-entering the remainder of a line which you have backspaced over to correct a typographical error. In conjunction with *pure cursor moves* (which follow), it is also useful for recopying and editing data which is already on the screen.

ESCAPE CODES

When you press the key marked **ESC** on the keyboard, the Apple's input subroutines go into *escape mode*. In this mode, eleven keys have separate meanings, called "escape codes". When you press one of these eleven keys, the Apple will perform the function associated with that key. After it has performed the function, the Apple will either continue or terminate escape mode, depending upon which escape code was performed. If you press any key in escape mode which is not an escape code, then that keypress will be ignored and escape mode will be terminated.

The Apple recognizes eleven escape codes, eight of which are *pure cursor moves*, which simply move the cursor without altering the screen or the input line, and three of which are *screen clear codes*, which simply blank part or all of the screen. All of the screen clear codes and the first four pure cursor moves (escape codes @, A, B, C, D, E, and F) terminate the escape mode after operating. The final four escape codes (I, K, M, and J) complete their functions with escape mode active.*

ESC A A press of the **ESC** key followed by a press of the **A** key will move the cursor one space to the right without changing the input line. This is useful for skipping over unwanted characters in an input line: simply backspace back over the unwanted characters, press **ESC A** to skip each offending symbol, and use the retype key to re-enter the remainder of the line.

ESC B Pressing **ESC** followed by **B** moves the cursor back one space, also without disturbing the input line. This may be used to enter something twice on the same line without retyping it: just type it once, press **ESC B** repeatedly to get back to the beginning of the phrase, and use the retype key to enter it again.

* These four escape codes are not available on Apples without the Autostart Monitor ROM.

[ESC] C The key sequence **[ESC] C** moves the cursor one line directly down, with no horizontal movement. If the cursor reaches the bottom of the text window, then the cursor remains on the bottom line and the text in the window scrolls up one line. The input line is not modified by the **[ESC] C** sequence. This, and **[ESC] D** (below), are useful for positioning the cursor at the beginning of another line on the screen, so that it may be re-entered with the retype key.

[ESC] D The **[ESC] D** sequence moves the cursor directly up one line, again without any horizontal movement. If the cursor reaches the top of the window, it stays there. The input line remains unmodified. This sequence is useful for moving the cursor to a previous line on the screen so that it may be re-entered with the retype key.

[ESC] E The **[ESC] E** sequence is called “clear to end of line”. When COUT detects this sequence of keypresses, it clears the remainder of the screen line (*not* the input line!) from the cursor position to the right edge of the text window. The cursor remains where it is, and the input line is unmodified. **[ESC] E** always clears the rest of the line to blank spaces, regardless of the setting of the Normal/Inverse mode location (see above).

[ESC] F This sequence is called “clear to end of screen”. It does just that: it clears everything in the window below or to the right of the cursor. As before, the cursor does not move and the input line is not modified. This is useful for erasing random garbage on a cluttered screen after a lot of cursor moves and editing.

[ESC] @ The **[ESC] @** sequence is called “home and clear”. It clears the entire window and places the cursor in the upper left-hand corner. The screen is cleared to blank spaces, regardless of the setting of the Normal/Inverse location, and the input line is not changed (note that “[@]” is **SHIFT P**).

[ESC] K These four escape codes are synonyms for the four pure cursor moves given above.
[ESC] J When these four escape codes finish their respective functions, they do *not* turn off the **[ESC] M** escape mode; you can continue typing these escape codes and moving the cursor around
[ESC] I the screen until you press any key other than another escape code. These four keys are placed in a “directional keypad” arrangement, so that the direction of each key from the center of the keypad corresponds to the direction which that escape code moves the cursor.

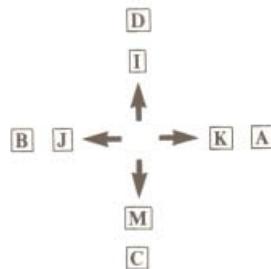


Figure 4. Cursor-moving Escape Codes.

THE RESET CYCLE

When you turn your Apple's power switch on* or press and release the [RESET] key, the Apple's 6502 microprocessor initiates a RESET cycle. It begins by jumping into a subroutine in the Apple's Monitor ROM. In the two different versions of this ROM, the Monitor ROM and the Autostart ROM, the RESET cycle does very different things.

AUTOSTART ROM RESET

Apples with the Autostart ROM begin their RESET cycles by flipping the soft switches which control the video screen to display the full primary page of Text mode, with Low-Resolution Graphics mixed mode lurking behind the veil of text. It then opens the text window to its full size, drops the output cursor to the bottom of the screen, and sets Normal video mode. Then it sets the COUT and KEYIN switches to use the Apple's internal keyboard and video display as the standard input and output devices. It flips annunciators 0 and 1 ON and annunciators 2 and 3 OFF on the Game I/O connector, clears the keyboard strobe, turns off any active I/O Expansion ROM (see page 84), and sounds a "beep!".

These actions are performed every time you press and release the [RESET] key on your Apple. At this point, the Autostart ROM peeks into two special locations in memory to see if it's been RESET before or if the Apple has just been powered up (these special locations are described below). If the Apple has just been turned on, then the Autostart ROM performs a "cold start"; otherwise, it does a "warm start".

1) **Cold Start.** On a freshly activated Apple, the RESET cycle continues by clearing the screen and displaying "APPLE II" top and center. It then sets up the special locations in memory to tell itself that it's been powered up and RESET. Then it starts looking through the rightmost seven slots in your Apple's backplane, looking for a Disk II Controller Card. It starts the search with Slot 7 and continues down to Slot 1. If it finds a disk controller card, then it proceeds to bootstrap the Apple Disk Operating System (DOS) from the diskette in the disk drive attached to the controller card it discovered. You can find a description of the disk bootstrapping procedure in **Do's and Don'ts of DOS**, Apple part number A2L0012, page 11.

If the Autostart ROM cannot find a Disk II controller card, or you press [RESET] again before the disk booting procedure has completed, then the RESET cycle will continue with a "lukewarm start". It will initialize and jump into the language which is installed in ROM on your Apple. For a Revision 0 Apple, either without an Applesoft II Firmware card or with such a card with its controlling switch in the DOWN position, the Autostart ROM will start Apple Integer BASIC. For Apple II-Plus systems, or Revision 0 Apple IIs with the Applesoft II Firmware card with the switch in the UP position, the Autostart ROM will begin Applesoft II Floating-Point BASIC.

2) **Warm Start.** If you have an Autostart ROM which has already performed a cold start cycle, then each time you press and release the [RESET] key, you will be returned to the language you were using, with your program and variables intact.

* Power-on RESET cycles occur only on Revision 1 Apples or Revision 0 Apples with at least one Disk II controller card.

AUTOSTART ROM SPECIAL LOCATIONS

The three "special locations" used by the Autostart ROM all reside in an area of RAM memory reserved for such system functions. Following is a table of the special locations used by the Autostart ROM:

Table 13: Autostart ROM Special Locations			
Location:	Decimal	Hex	Contents:
1010	\$3F2		Soft Entry Vector. These two locations contain the address of the reentry point for whatever language is in use. Normally contains \$E003.
1011	\$3F3		
1012	\$3F4		Power-Up Byte. Normally contains \$45. See below.
64367 (-1169)	\$FB6F		This is the beginning of a machine language subroutine which sets up the power-up location.

When the Apple is powered up, the Autostart ROM places a special value in the power-up location. This value is the Exclusive-OR of the value contained in location 1011 with the constant value 165. For example, if location 1011 contains 224 (its normal value), then the power-up value will be:

	Decimal	Hex	Binary
Location 1011	224	\$E0	11100000
Constant	165	\$A5	10100101
Power-Up Value	69	\$45	01000101

Your programs can change the soft entry vector, so that when you press [RESET] you will go to some program other than a language. If you change this soft entry vector, however, you should make sure that you set the value of the power-up byte to the Exclusive-OR of the high part of your new soft entry vector with the constant decimal 165 (hexadecimal \$A5). If you do not set this power-up value, then the next time you press [RESET] the Autostart ROM will believe that the Apple has just been turned on and it will do another cold start.

For example, you can change the soft entry vector to point to the Apple System Monitor, so that when you press [RESET] you will be placed into the Monitor. To make this change, you must place the address of the beginning of the Monitor into the two soft entry vector locations. The Monitor begins at location \$FF69, or decimal 65385. Put the last two hexadecimal digits of this address (\$69) into location \$3F2 and the first two digits (\$FF) into location \$3F3. If you are working in decimal, put 105 (which is the remainder of 65385/256) into location 1010 and the value 255 (which is the integer quotient of 65385/256) into location 1011.

Now you must set up the power-up location. There is a machine-language subroutine in the Autostart ROM which will automatically set the value of this location to the Exclusive-OR mentioned above. All you need to do is to execute a JSR (Jump to SubRoutine) instruction to the address \$FB6F. If you are working in BASIC, you should perform a CALL -1169. Now everything is set, and the next time you press [RESET], you will enter the System Monitor.

To make the [RESET] key work in its usual way, just restore the values in the soft entry vector to their former values (\$E003, or decimal 57347) and again call the subroutine described above.



“OLD MONITOR” ROM RESET

A RESET cycle in the Apple II Monitor ROM begins by setting Normal video mode, a full screen of Primary Page text with the Color Graphics mixed mode behind it, a fully-opened text window, and the Apple's standard keyboard and video screen as the standard input and output devices. It sounds a “beep!”, the cursor leaps to the bottom line of the uncleared text screen, and you find yourself facing an asterisk (*) prompt and talking to the Apple System Monitor.



CHAPTER 3

THE SYSTEM MONITOR

- 40 ENTERING THE MONITOR
- 40 ADDRESSES AND DATA
- 41 EXAMINING THE CONTENTS OF MEMORY
- 41 EXAMINING SOME MORE MEMORY
- 43 EXAMINING STILL MORE MEMORY
- 43 CHANGING THE CONTENTS OF A LOCATION
- 44 CHANGING THE CONTENTS OF CONSECUTIVE LOCATIONS
- 44 MOVING A RANGE OF MEMORY
- 46 COMPARING TWO RANGES OF MEMORY
- 46 SAVING A RANGE OF MEMORY ON TAPE
- 47 READING A RANGE FROM TAPE
- 48 CREATING AND RUNNING MACHINE LANGUAGE PROGRAMS
- 49 THE MINI-ASSEMBLER
- 51 DEBUGGING PROGRAMS
- 53 EXAMINING AND CHANGING REGISTERS
- 54 MISCELLANEOUS MONITOR COMMANDS
- 55 SPECIAL TRICKS WITH THE MONITOR
- 57 CREATING YOUR OWN COMMANDS
- 59 SUMMARY OF MONITOR COMMANDS
- 61 SOME USEFUL MONITOR SUBROUTINES
- 65 MONITOR SPECIAL LOCATIONS
- 66 MINI-ASSEMBLER INSTRUCTION FORMATS

Buried deep within the recesses of the Apple's ROM is a masterful program called the System Monitor. It acts as both a supervisor of the system and a slave to it; it controls all programs and all programs use it. You can use the powerful features of the System Monitor to discover the hidden secrets in all 65,536 memory locations. From the Monitor, you may look at one, some, or all locations; you may change the contents of any location; you can write programs in Machine and Assembly languages to be executed directly by the Apple's microprocessor; you can save vast quantities of data and programs onto cassette tape and read them back in again; you can move and compare thousands of bytes of memory with a single command; and you can leave the Monitor and enter any other program or language on the Apple.

ENTERING THE MONITOR

The Apple System Monitor program begins at location number \$FF69 (decimal 65385 or -151) in memory. To enter the Monitor, you or your BASIC program can CALL this location. The Monitor's prompt (an asterisk [*]) will appear on the left edge of the screen, with a flashing cursor to its right. The Monitor accepts standard input lines (see page 32) just like any other system or language on the Apple. It will not take any action until you press [RETURN]. Your input lines to the Monitor may be up to 255 characters in length. When you have finished your stay in the Monitor, you can return to the language you were previously using by typing [CTRL C] [RETURN] (or, with the Apple DOS, [3] [D] [#] [G] [RETURN]), or simply press [RESET].*

ADDRESSES AND DATA

Talking to the Monitor is somewhat like talking to any other program or language on the Apple: you type a line on the keyboard, followed by a [RETURN], and the Monitor will digest what you typed and act according to those instructions. You will be giving the Monitor three types of information: *addresses*, *values*, and *commands*. Addresses and values are given to the Monitor in hexadecimal notation. Hexadecimal notation uses the ten decimal digits (0-9) to represent themselves and the first six letters (A-F) to represent the numbers 10 through 15. A single hexadecimal digit can, therefore, have one of sixteen values from 0 to 15. A pair of hex digits can assume any value from 0 to 255, and a group of four hex digits can denote any number from 0 to 65,536. It so happens that any address in the Apple can be represented by four hex digits, and any value by two hex digits. This is how you tell the Monitor about addresses and values. When the Monitor is looking for an address, it will take any group of hex digits. If there are fewer than four digits in the group, it will prepend leading zeroes; if there are more than four hex digits, the Monitor will truncate the group and use only the last four hex digits. It follows the same procedure when looking for two-digit data values.

The Monitor recognizes 22 different command characters. Some of these are punctuation marks, others are upper-case letters or control characters. In the following sections, the full name of a command will appear in capital letters. The Monitor needs only the first letter of the command name. Some commands are invoked with control characters. You should note that although the Monitor recognizes and interprets these characters, a control character typed on an input line will *not* appear on the screen.

* This does not work on Apples without the Autostart ROM.

The Monitor remembers the addresses of up to five locations. Two of these are special: they are the addresses of the last location whose value you inquired about, and the location which is next to have its value changed. These are called the *last opened location* and the *next changeable location*. The usefulness of these two addresses will be revealed shortly.

EXAMINING THE CONTENTS OF MEMORY

When you type the address of a location in memory alone on an input line to the Monitor, it will reply* with the address you typed, a dash, a space, and the value** contained in that location, thus:

```
*E000  
E000- 20  
*300  
#300- 99  
*
```

Each time the Monitor displays the value contained in a location, it remembers that location as the *last opened location*. For technical reasons, it also considers that location as the *next changeable location*.

EXAMINING SOME MORE MEMORY

If you type a period(.) on an input line to the Monitor, followed by an address, the Monitor will display a *memory dump*: the values contained in all locations from the last opened location to the location whose address you typed following the period. The Monitor then considers the last location displayed to be both the last opened location and the next changeable location.

* In the examples, your queries are in normal type and the Apple replies in **boldface**.

** The values printed in these examples may differ from the values displayed by your Apple for the same instructions.

```

* 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
*

```

You should notice several things about the format of a memory dump. First, the first line in the dump begins with the address of the location *following* the last opened location; second, all other lines begin with addresses which end alternately in zeroes and eights; and third, there are never more than eight values displayed on a single line in a memory dump. When the Monitor does a memory dump, it starts by displaying the address and value of the location following the last opened location. It then proceeds to the next successive location in memory. If the address of that location ends in an 8 or a 0, the Monitor will "cut" to a new line and display the address of that location and continue displaying values. After it has displayed the value of the location whose address you specified, it stops the memory dump and sets the address of both the last opened and the next changeable location to be the address of the last location in the dump. If the address specified on the input line is less than the address of the last opened location, the Monitor will display the address and value of only the location following the last opened location.

You can combine the two commands (opening and dumping) into one operation by concatenating the second to the first; that is, type the first address, followed by a period and the second address. This two-addresses-separated-by-a-period form is called a *memory range*.

```

* 300 . 32F
0300- 99 B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20 46 03 A5 3D 4D
* 30 . 40
0030- AA 00 FF AA 05 C2 05 C2
0038- 1B FD D0 03 3C 00 40 00
0040- 30
* E015 . E025

```

```
E#15- 4C ED FD  
E#18- A9 2# C5 24 B# #C A9 8D  
E#2#- A# #7 2# ED FD A9  
*
```

EXAMINING STILL MORE MEMORY

A single press of the [RETURN] key will cause the Monitor to respond with one line of a memory dump; that is, a memory dump from the location following the last opened location to the next eight-location "cut". Once again, the last location displayed is considered the last opened and next changeable location.

```
* 5  
  
#005- #  
* [RETURN]  
# # #  
* [RETURN]  
  
#008- # # # # # # # #  
* 32  
  
#032- FF  
* [RETURN]  
AA # # C2 #5 C2  
* [RETURN]  
  
#038- 1B FD D# #3 3C # # 3F #  
*
```

CHANGING THE CONTENTS OF A LOCATION

You've heard all about the "next changeable location"; now you're going to use it. Type a colon followed by a value.

```
* #  
  
#0#- #  
* : 5F
```

Presto! The contents of the next changeable location have just been changed to the value you typed. Check this by examining that location again:

```
* #  
  
#0#- 5F
```

*
You can also combine opening and changing into one operation:

* 302:42

* 302

0302- 42

*

When you change the contents of a location, the old value which was contained in that location disappears, never to be seen again. The new value will stick around until it is replaced by another hexadecimal value.

CHANGING THE CONTENTS OF CONSECUTIVE LOCATIONS

You *don't* have to type an address, a colon, a value, and press [RETURN] for each and every location you wish to change. The Monitor will allow you to change the values of up to eighty-five locations at a time by typing only the initial address and colon, and then all the values separated by spaces. The Monitor will duly file the consecutive values in consecutive locations, starting at the next changeable location. After it has processed the string of values, it will assume that the location following the last changed location is the next changeable location. Thus, you can continue changing consecutive locations without breaking stride on the next input line by typing another colon and more values.

* 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

*

MOVING A RANGE OF MEMORY

You can treat a range of memory (specified by two addresses separated by a period) as an entity

unto itself and move it from one place to another in memory by using the Monitor's MOVE command. In order to move a range of memory from one place to another, the Monitor must be told both where the range is situated in memory and where it is to be moved. You give this information to the Monitor in three parts: the address of the destination of the range, the address of the first location in the range proper, and the address of the last location in the range. You specify the starting and ending addresses of the range in the normal fashion, by separating them with a period. You indicate that this range is to be placed somewhere else by separating the range and the destination address with a left caret (<). Finally, you tell the Monitor that you want to move the range to the destination by typing the letter M, for "MOVE". The final command looks like this:

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

When you type this line to the Monitor, of course, the words in curly brackets should be replaced by hexadecimal addresses and the spaces should be omitted. Here are some real examples of memory moves:

```
*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  
  
#300- A9 8D 20 ED FD A9 45 20  
#308- 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  
  
#310- 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 simply makes a copy of the indicated range and moves it to the specified destination. The original range is left undisturbed. The Monitor remembers the last location in the original range as the last opened location, and the first location in the original range as the next changeable location. If the second address in the 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 original range, then strange and (sometimes) wonderful things happen: the locations between the beginning of the range and the

destination are treated as a sub-range and the values in this sub-range are replicated throughout the original range. See "Special Tricks", page 55, for an interesting application of this feature.

COMPARING TWO RANGES OF MEMORY

You can use the Monitor to compare two ranges of memory using much the same format as 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. In shorthand:

{destination} < {start} . {end} V

The Monitor compares the range specified with the range beginning at the destination address. If there is any discrepancy, the Monitor displays the address at which the difference was found and the two offending values.

```
*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  
#0#6-E4 (EE)  
*
```

Notice that the VERIFY command, if it finds a discrepancy, displays the address of the location in the original range whose value differs from its counterpart in the destination range. If there is no discrepancy, VERIFY displays nothing. It leaves both ranges unchanged. The last opened and next changeable locations are set just as in the MOVE command. As before, 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. VERIFY also does unusual things if the destination is within the original range; see "Special Tricks", page 55.

SAVING A RANGE OF MEMORY ON TAPE

The Monitor has two special commands which allow you to save a range of memory onto cassette tape and recall it again for later use. The first of these two commands, WRITE, lets you save the contents of one to 65,536 memory locations on standard cassette tape.

To save a range of memory to tape, give the Monitor the starting and ending addresses of the range, followed by the letter W (for WRITE):

{start} . {end} W

To get an accurate recording, you should put the tape recorder in *record* mode before you press [RETURN] on the input line. Let the tape run a few seconds, then press [RETURN]. The Monitor will write a ten-second "leader" tone onto the tape, followed by the data. When the Monitor is finished, it will sound a ""beep!" and give you another prompt. You should then rewind the tape, and label the tape with something intelligible about the memory range that's on the tape and what it's supposed to be.

```
*0. FF FF AD 30 C0 88 D0 04 C6 01 F0 08 C |  
A D0 F6 A6 00 4C 02 00 60
```

```
*0. 14
```

```
0000- FF FF AD 30 C0 88 D0 04  
0008- C6 01 F0 08 CA D0 F6 A6  
0010- 00 4C 02 00 60  
*0. 14W
```

```
*
```

It takes about 35 seconds total to save the values of 4,096 memory locations preceded by the ten-second leader onto tape. This works out to a speed of about 1,350 bits per second, average. The WRITE command writes one extra value on the tape after it has written the values in the memory range. This extra value is the *checksum*. It is the partial sum of all values in the range. The READ subroutine uses this value to determine if a READ has been successful (see below).

READING A RANGE FROM TAPE

Once you've saved a memory range onto tape with the Monitor's WRITE command, you can read that memory range back into the Apple by using the Monitor's READ command. The data values which you've stored on the tape need not be read back into the same memory range from whence they came; you can tell the Monitor to put those values into any similarly sized memory range in the Apple's memory.

The format of the READ command is the same as that of the WRITE command, except that the command letter is R, not W:

{start} . {end} R

Once again, after typing the command, don't press [RETURN]. Instead, start the tape recorder in PLAY mode and wait for the tape's nonmagnetic leader to pass by. Although the WRITE command puts a ten-second leader tone on the beginning of the tape, the READ command needs only three seconds of this leader in order to lock on to the frequency. So you should let a few seconds of tape go by before you press [RETURN], to allow the tape recorder's output to settle down to a steady tone.

```
*0:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |  
0 0
```

```
*0. 14
```

```

0000- 00 00 00 00 00 00 00 00 00 00
0008- 00 00 00 00 00 00 00 00 00 00
0010- 00 00 00 00 00 00 00 00 00 00
*0.14R

*0.14

0000- FF FF AD 30 C0 88 D0 #4
0008- C6 #1 F0 #8 CA D0 F6 A6
0010- #0 4C #2 #0 60
*

```

After the Monitor has read in and stored all the values on the tape, it reads in the extra checksum value. It compares the checksum on the tape to its own checksum, and if the two differ, the Monitor beeps the speaker and displays "ERR". This warns you that there was a problem during the READ and that the values stored in memory aren't the values which were recorded on the tape. If, however, the two checksums match, the Monitor will give you another prompt.

CREATING AND RUNNING MACHINE LANGUAGE PROGRAMS

Machine language is certainly the most efficient language on the Apple, albeit the least pleasant in which to code. The Monitor has special facilities for those of you who are determined to use machine language to simplify creating, writing, and debugging machine language programs.

You can write a machine language program, take the hexadecimal values for the opcodes and operands, and store them in memory using the commands covered above. You can get a hexadecimal dump of your program, move it around in memory, or save it to tape and recall it again simply by using the commands you've already learned. The most important command, however, when dealing with machine language programs is the GO command. When you open a location from the Monitor and type the letter G, the Monitor will cause the 6502 microprocessor to start executing the machine language program which begins at the last opened location. The Monitor treats this program as a subroutine: when it's finished, all it need do is execute an RTS (return from subroutine) instruction and control will be transferred back to the Monitor.

Your machine language programs can call many subroutines in the Monitor to do various things. Here is an example of loading and running a machine language program to display the letters A through Z:

```

*300:A9 C1 20 ED FD 18 69 1 C9 DB D0 F6 60
*300.30C
#300- A9 C1 20 ED FD 18 69 #1
#308- C9 DB D0 F6 60
*300G
ABCDEFGHIJKLMNPQRSTUVWXYZ
*
```

(The instruction set of the Apple's 6502 microprocessor is listed in Appendix A of this manual.)

Now, straight hexadecimal code isn't the easiest thing in the world to read or debug. With this in mind, the creators of the Apple's Monitor neatly included a command to list machine language programs in *assembly language* form. This means that instead of having one, two, or three bytes of unformatted hexadecimal gibberish per instruction you now have a three-letter mnemonic and some formatted hexadecimal gibberish to comprehend for each instruction. The LIST command to the Monitor will start at the specified location and display a screenfull (20 lines) of instructions:

```
* 300L  
0300- A9 C1      LDA    #$C1  
0302- 20 ED FD    JSR    $FDED  
0305- 18          CLC      
0306- 69 #1       ADC    #$#1  
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    *  

```

Recognize those first few lines? They're the assembly language form of the program you typed in a page or so ago. The rest of the lines (the BRK instructions) are just there to fill up the screen. The address that you specify is remembered by the Monitor, but not in one of the ways explained before. It's put in the *Program Counter*, which is used solely to point to locations within programs. After a LIST command, the Program Counter is set to point to the location immediately following the last location displayed on the screen, so that if you do another LIST command it will continue with another screenfull of instructions, starting where the first screen left off.

THE MINI-ASSEMBLER

There is another program within the Monitor* which allows you to type programs into the Apple in the same assembly format which the LIST command displays. This program is called the Apple Mini-Assembler. It is a "mini"-assembler because it cannot understand symbolic labels, something that a full-blown assembler must do. To run the Mini-Assembler, type:

* The Mini-Assembler does not actually reside in the Monitor ROM, but is part of the Integer BASIC ROM set. Thus, it is not available on Apple II Plus systems or while Firmware Applesoft II is in use.

*F666G

!

You are now in the Mini-Assembler. The exclamation point (!) is the prompt character. During your stay in the Mini-Assembler, you can execute any Monitor command by preceding it with a dollar sign (\$). Aside from that, the Mini-Assembler has an instruction set and syntax all its own.

The Mini-Assembler remembers one address, that of the Program Counter. Before you start to enter a program, you must set the Program Counter to point to the location where you want your program to go. Do this by typing the address followed by a colon. Follow this with the mnemonic for the first instruction in your program, followed by a space. Now type the operand of the instruction (Formats for operands are listed on page 66). Now press [RETURN]. The Mini-Assembler converts the line you typed into hexadecimal, stores it in memory beginning at the location of the Program Counter, and then disassembles it again and displays the disassembled line on top of your input line. It then poses another prompt on the next line. Now it's ready to accept the second instruction in your program. To tell it that you want the next instruction to follow the first, don't type an address or a colon, but only a space, followed by the next instruction's mnemonic and operand. Press [RETURN]. It assembles that line and waits for another.

If the line you type has an error in it, the Mini-Assembler will beep loudly and display a circumflex (^) under or near the offending character in the input line. Most common errors are the result of typographical mistakes: misspelled mnemonics, missing parentheses, etc. The Mini-Assembler also will reject the input line if you forget the space before or after a mnemonic or include an extraneous character in a hexadecimal value or address. If the destination address of a branch instruction is out of the range of the branch (more than 127 locations distant from the address of the instruction), the Mini-Assembler will also flag this as an error.

```
!300:LDX #02
    0300-A2 02      LDX    #$02
    ! LDA $0,X

    0302-B5 00      LDA    $00,X
    ! STA $10,X

    0304-95 10      STA    $10,X
    ! DEX

    0306-CA         DEX

    ! STA SC030

    0307-8D 30 C0    STA    $C030
    ! BPL $302

    030A-10 F6      BPL    $0302

    030C-00          BRK
```

To exit the Mini-Assembler and re-enter the Monitor, either press [RESET] or type the Monitor

command (preceded by a dollar sign) FF69G:

```
!$FF69G
```

*

Your assembly language program is stored in memory. You can look at it again with the LIST command:

```
*300L
```

0300-	A2 02	LDX	#\$02
0302-	B5 00	LDA	\$00,X
0304-	95 10	STA	\$10,X
0306-	CA	DEX	
0307-	8D 30 C0	STA	\$C030
030A-	10 F6	BPL	\$#302
030C-	00	BRK	
030D-	00	BRK	
030E-	00	BRK	
030F-	00	BRK	
0310-	00	BRK	
0311-	00	BRK	
0312-	00	BRK	
0313-	00	BRK	
0314-	00	BRK	
0315-	00	BRK	
0316-	00	BRK	
0317-	00	BRK	
0318-	00	BRK	
0319-	00	BRK	

*

DEBUGGING PROGRAMS

As put so concisely by Lubarsky*, "There's always one more bug." Don't worry, the Monitor provides facilities for stepping through ornery programs to find that one last bug. The Monitor's STEP** command decodes, displays, and executes one instruction at a time, and the TRACE** command steps quickly through a program, stopping when a BRK instruction is executed.

Each STEP command causes the Monitor to execute the instruction in memory pointed to by the Program Counter. The instruction is displayed in its disassembled form, then executed. The contents of the 6502's internal registers are displayed after the instruction is executed. After execution, the Program Counter is bumped up to point to the next instruction in the program.

Here's what happens when you STEP through the program you entered using the Mini-Assembler, above:

* In *Murphy's Law, and Other Reasons why Things Go Wrong*, edited by Arthur Bloch. Price/Stern/Sloane 1977.

** The STEP and TRACE commands are not available on Apples with the Autostart ROM.

```

*300S

#3#0- A2 #2 LDX #$#2
A=#A X=#2 Y=D8 P=3# S=F8
*S

#3#2- B5 #0 LDA $#0,X
A=#C X=#2 Y=D8 P=3# S=F8
*S

#3#4- 95 1# STA $1#,X
A=#C X=#2 Y=D8 P=3# S=F8
*12

#012- #C
*S

#3#6- CA DEX
A=#C X=#1 Y=D8 P=3# S=F8
*S

#3#7- 8D 3# C# STA $C#3#
A=#C X=#1 Y=D8 P=3# S=F8
*S

#3#A- 1# F6 BPL $#3#2
A=#C X=#1 Y=D8 P=3# S=F8
*S

#3#2- B5 #0 LDA $#0,X
A=#B X=#1 Y=D8 P=3# S=F8
*S

#3#4- 95 1# STA $1#,X
A=#B X=#1 Y=D8 P=3# S=F8
*

```

Notice that after the third instruction was executed, we examined the contents of location 12. They were as we expected, and so we continued stepping. The Monitor keeps the Program Counter and the last opened address separate from one another, so that you can examine or change the contents of memory while you are stepping through your program.

The TRACE command is just an infinite STEPper. It will stop TRACEing the execution of a program only when you push [RESET] or it encounters a BRK instruction in the program. If the TRACE encounters the end of a program which returns to the Monitor via an RTS instruction, the TRACEing will run off into never-never land and must be stopped with the [RESET] button.

```

*T

#3#6- CA DEX
A=#B X=#0 Y=D8 P=32 S=F8
#3#7- 8D 3# C# STA $C#3#
A=#B X=#0 Y=D8 P=32 S=F8
#3#A- 1# F6 BPL $#3#2

```

```

A=0B X=00 Y=D8 P=32 S=F8
#302- B5 00 LDA $00,X
A=0A X=00 Y=D8 P=30 S=F8
#304- 95 1# STA $1#,X
A=0A X=00 Y=D8 P=30 S=F8
#306- CA DEX
A=0A X=FF Y=D8 P=B0 S=F8
#307- 8D 3# C# STA SC#3#
A=0A X=FF Y=D8 P=B0 S=F8
#30A- 1# F6 BPL $#3#2
A=0A X=FF Y=D8 P=B0 S=F8
#30C- 0# BRK
#30C- A=0A X=FF Y=D8 P=B0 S=F8
*

```

EXAMINING AND CHANGING REGISTERS

As you saw above, the STEP and TRACE commands displayed the contents of the 6502's internal registers after each instruction. You can examine these registers at will or pre-set them when you TRACE, STEP, or GO a machine language program.

The Monitor reserves five locations in memory for the five 6502 registers: A, X, Y, P (processor status register), and S (stack pointer). The Monitor's EXAMINE command, invoked by a **CTRL E**, tells the Monitor to display the contents of these locations on the screen, and lets the location which holds the 6502's A-register be the next changeable location. If you want to change the values in these locations, just type a colon and the values separated by spaces. Next time you give the Monitor a GO, STEP, or TRACE command, the Monitor will load these five locations into their proper registers inside the 6502 before it executes the first instruction in your program.

```

* [CTRL E]

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

* [CTRL E]

A=B# X=02 Y=D8 P=B# S=F8
* 3#6S

#3#6- CA DEX
A=B# X=01 Y=D8 P=3# S=F8
*S

#3#7- 8D 3# C# STA SC#3#
A=B# X=01 Y=D8 P=3# S=F8
*S

#3#A- 1# F6 BPL $#3#2
A=B# X=01 Y=D8 P=3# S=F8

*

```

MISCELLANEOUS MONITOR COMMANDS

You can control the setting of the Inverse/Normal location used by the COUT subroutine (see page 32) from the Monitor so that all of the Monitor's output will be in Inverse video. The INVERSE command does this nicely. Input lines are still displayed in Normal mode, however. To return the Monitor's output to Normal mode, use the NORMAL command.

```
* Ø . F  
ØØØØ- ØA ØB ØC ØD ØE ØF DØ Ø4  
ØØØ8- C6 Ø1 FØ Ø8 CA DØ F6 A6  
* I  
  
* Ø . F  
ØØØØ- ØA ØB ØC ØD ØE ØF DØ Ø4  
ØØØ8- C6 Ø1 FØ Ø8 CA DØ F6 A6  
* N  
  
* Ø . F  
ØØØØ- ØA ØB ØC ØD ØE ØF DØ Ø4  
ØØØ8- C6 Ø1 FØ Ø8 CA DØ F6 A6  
*
```

The BASIC command, invoked by a **[CTRL B]**, lets you leave the Monitor and enter the language installed in ROM on your Apple, usually either Apple Integer or Applesoft II BASIC. Any program or variables that you had previously in BASIC will be lost. If you've left BASIC for the Monitor and you want to re-enter BASIC with your program and variables intact, use the **[CTRL C]** (CONTINUE BASIC) command. If you have the Apple Disk Operating System (DOS) active, the '3DØG' command will return you to the language you were using, with your program and variables intact.

The PRINTER command, activated by a **[CTRL P]**, diverts all output normally destined for the screen to an Apple Intelligent Interface® in a given slot in the Apple's backplane. The slot number should be from 1 to 7, and there should be an interface card in the given slot, or you will lose control of your Apple and your program and variables may be lost. The format for the command is:

{slot number} **[CTRL P]**

A PRINTER command to slot number Ø will reset the flow of printed output back to the Apple's video screen.

The KEYBOARD command similarly substitutes the device in a given backplane slot for the Apple's keyboard. For details on how these commands and their BASIC counterparts PR# and IN# work, please refer to "CSW and KSW Switches", page 83. The format for the KEYBOARD command is:

{slot number} **[CTRL K]**

A slot number of \emptyset for the KEYBOARD command will force the Monitor to listen for input from the Apple's built-in keyboard.

The Monitor will also perform simple hexadecimal addition and subtraction. Just type a line in the format:

```
{value} + {value}  
{value} - {value}
```

The Apple will perform the arithmetic and display the result:

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

SPECIAL TRICKS WITH THE MONITOR

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. You can intermix any and all commands freely, 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. The NORMAL command makes a good separator; it usually has no effect and can be used anywhere.

```
* 300.307 300:18 69 1 N 300.302 300S S  
#300- 00 00 00 00 00 00 00 00 00  
#300- 18 69 #1  
#300- 18 CLC  
A=#4 X=#1 Y=D8 P=3# S=F8  
#3#1- 69 #1 ADC #$#1  
A=#5 X=#1 Y=D8 P=3# S=F8  
*
```

Single-letter commands such as L, S, I, and N need not be separated by spaces.

If the Monitor encounters a character in the input line which it does not recognize as either a hexadecimal digit or a valid command character, it will execute all commands on the input line up to that character, and then grind to a halt with a noisy beep, ignoring the remainder of the input line.

The MOVE command can be used to replicate a pattern of values throughout a range in memory.

To do this, first store the pattern in its first position in the range:

```
*300:11 22 33  
*
```

Remember the number of values in the pattern: in this case, 3. Then use this special arrangement of the MOVE command:

```
[start+number] < [start] . [end-number] M
```

This MOVE command will first replicate the pattern at the locations immediately following the original pattern, then re-replicate that pattern following itself, and so on until it fills the entire range.

```
*303<300 . 32DM  
*300 . 32F  
#300- 11 22 33 11 22 33 11 22  
#308- 33 11 22 33 11 22 33 11  
#310- 22 33 11 22 33 11 22 33  
#318- 11 22 33 11 22 33 11 22  
#320- 33 11 22 33 11 22 33 11  
#328- 22 33 11 22 33 11 22 33  
*
```

A similar trick can be done 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:

```
*300:0  
*301<300 . 31FM  
*301<300 . 31FV  
*304:02  
*301<300 . 31FV  
#303-#0 (02)  
#304-#2 (00)  
*
```

You can create a command line which will repeat all or part of itself indefinitely by beginning the part of the command line which is to be repeated with a letter command, such as N, and ending it with the sequence 34:n, where n is a hexadecimal number specifying the character position of the command which begins the loop; 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.

```
*N 300 302 34:0  
#300- 11
```

```
03#2- 33  
03#0- 11  
03#2- 33  
03#0- 11  
03#2- 33  
03#0- 11  
03#2- 33  
03#0- 11  
03#2- 33  
03#0- 11  
03#2- 33  
03#0- 11  
03#2- 33  
03#0- 11  
03#2- 33  
03#0-  
*
```

The only way to stop a loop like this is to press [RESET].

CREATING YOUR OWN COMMANDS

The USER ([CTRL Y]) command, when encountered in the input line, forces the Monitor to jump to location number \$3F8 in memory. You can put your own JMP instruction in this location which will jump to your own program. Your program can then either examine the Monitor's registers and pointers or the input line itself. For example, here is a program which will make the [CTRL Y] command act as a "comment" indicator: everything on the input line following the [CTRL Y] will be displayed and ignored.

```
*F666G  
  
!3#0 : LDY $34  
  
03#0- A4 34 LDY $34  
! LDA 2#0,Y  
  
03#2- B9 #0 #2 LDA $#2#0,Y  
! JSR FD#D  
  
03#5- 2# ED FD JSR $FD#D  
! INY  
  
03#8- C8 INY  
! CMP #$8D  
  
03#9- C9 8D CMP #$8D  
! BNE 3#2  
  
03#B- D# F5 BNE $#3#2  
! JMP $FF69  
  
03#D- 4C 69 FF JMP $FF69  
! 3F8 : JMP $3#0  
  
03F8- 4C #0 #3 JMP $#3#0
```

!\$FF69G

• [CTRL Y] THIS IS A TEST.
THIS IS A TEST.

•

SUMMARY OF MONITOR COMMANDS

Summary of Monitor Commands.

Examining Memory.

{adr\$}	Examines the value contained in one location.
{adr\$1}.{adr\$2}	Displays the values contained in all locations between {adr\$1} and {adr\$2}.
RETURN	Displays the values in up to eight locations following the last opened location.

Changing the Contents of Memory.

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

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}.

Saving and Loading via Tape.

{start}.{end}W	Writes the values in the memory range {start}.{end} onto tape, preceded by a ten-second leader.
{start}.{end}R	Reads values from tape, storing them in memory beginning at {start} and stopping at {end}. Prints "ERR" if an error occurs.

Running and Listing Programs.

{adr\$}G	Transfers control to the machine language program beginning at {adr\$}.
{adr\$}L	Disassembles and displays 20 instructions, starting at {adr\$}. Subsequent L's will display 20 more instructions each.

Summary of Monitor Commands.

The Mini-Assembler

F666G	Invoke the Mini-Assembler.*
\${command}	Execute a Monitor command from the Mini-Assembler.
SFF69G	Leave the Mini-Assembler.
{adr}s S	Disassemble, display, and execute the instruction at {adr}, and display the contents of the 6502's internal registers. Subsequent S's will display and execute successive instructions.**
{adr}s T	Step infinitely. The TRACE command stops only when it executes a BRK instruction or when you press RESET .**

CTRL E

Display the contents of the 6502's registers.

Miscellaneous.

I	Set Inverse display mode.
N	Set Normal display mode.
CTRL B	Enter the language currently installed in the Apple's ROM.
CTRL C	Reenter the language currently installed in the Apple's ROM.
{val}+{val}	Add the two values and print the result.
{val}-{val}	Subtract the second value from the first and print the result.
{slot} CTRL P	Divert output to the device whose interface card is in slot number {slot}. If {slot}= \emptyset , then route output to the Apple's screen.
{slot} CTRL K	Accept input from the device whose interface card is in slot number {slot}. If {slot}= \emptyset , then accept input from the Apple's keyboard.
CTRL Y	Jump to the machine language subroutine at location \$3F8.

* Not available in the Apple II Plus.

** Not available in the Autostart ROM.

SOME USEFUL MONITOR SUBROUTINES

Here is a list of some useful subroutines in the Apple's Monitor and Autostart ROMs. To use these subroutines from machine language programs, load the proper memory locations or 6502 registers as required by the subroutine and execute a JSR to the subroutine's starting address. It will perform the function and return with the 6502's registers set as described.

\$FDED COUT Output a character

COUT is the standard character output subroutine. The character to be output should be in the accumulator. COUT calls the current character output subroutine whose address is stored in CSW (locations \$36 and \$37), usually COUT1 (see below).

\$FDF# COUT1 Output to screen

COUT1 displays the character in the accumulator on the Apple's screen at the current output cursor position and advances the output cursor. It places the character using the setting of the Normal/Inverse location. It handles the control characters RETURN, linefeed, and bell. It returns with all registers intact.

\$FE8# SETINV Set Inverse mode

Sets Inverse video mode for COUT1. All output characters will be displayed as black dots on a white background. The Y register is set to \$3F, all others are unchanged.

\$FE84 SETNORM Set Normal mode

Sets Normal video mode for COUT1. All output characters will be displayed as white dots on a black background. The Y register is set to \$FF, all others are unchanged.

\$FD8E CROUT Generate a RETURN

CROUT sends a RETURN character to the current output device.

\$FD8B CROUT1 RETURN with clear

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

\$FDDA PRBYTE Print a hexadecimal byte

This subroutine outputs the contents of the accumulator in hexadecimal on the current output device. The contents of the accumulator are scrambled.

\$FDE3 PRHEX Print a hexadecimal digit

This subroutine outputs the lower nibble of the accumulator as a single hexadecimal digit. The contents of the accumulator are scrambled.

\$F941 PRNTAX Print A and X in hexadecimal

This outputs the contents of the A and X registers as a four-digit hexadecimal value. The accumulator contains the first byte output, the X register contains the second. The contents of the

accumulator are usually scrambled.

\$F948 PRBLNK Print 3 spaces

Outputs three blank spaces to the standard output device. Upon exit, the accumulator usually contains \$A0, the X register contains 0.

\$F94A PRBL2 Print many blank spaces

This subroutine outputs from 1 to 256 blanks to the standard output device. Upon entry, the X register should contain the number of blanks to be output. If X=\$00, then PRBL2 will output 256 blanks.

\$FF3A BELL Output a "bell" character

This subroutine sends a bell (CTRL G) character to the current output device. It leaves the accumulator holding \$87.

\$FBDD BELL1 Beep the Apple's speaker

This subroutine beeps the Apple's speaker for .1 second at 1KHz. It scrambles the A and X registers.

\$FD0C RDKEY Get an input character

This is the standard character input subroutine. It places a flashing input cursor on the screen at the position of the output cursor and jumps to the current input subroutine whose address is stored in KSW (locations \$38 and \$39), usually KEYIN (see below).

\$FD35 RDCHAR Get an input character or ESC code

RDCHAR is an alternate input subroutine which gets characters from the standard input, but also interprets the eleven escape codes (see page 34).

\$FD1B KEYIN Read the Apple's keyboard

This is the keyboard input subroutine. It reads the Apple's keyboard, waits for a keypress, and randomizes the random number seed (see page 32). When it gets a keypress, it removes the flashing cursor and returns with the keycode in the accumulator.

\$FD6A GETLN Get an input line with prompt

GETLN is the subroutine which gathers input lines (see page 33). Your programs can call GETLN with the proper prompt character in location \$33; GETLN will return with the input line in the input buffer (beginning at location \$200) and the X register holding the length of the input line.

\$FD67 GETLNZ Get an input line

GETLNZ is an alternate entry point for GETLN which issues a carriage return to the standard output before falling into GETLN (see above).

\$FD6F GETLN1 Get an input line, no prompt

GETLN1 is an alternate entry point for GETLN which does not issue a prompt before it gathers the input line. If, however, the user cancels the input line, either with too many backspaces or with a **CTRL X**, then GETLN1 will issue the contents of location \$33 as a prompt when it gets another line.

\$FCA8 WAIT Delay

This subroutine delays for a specific amount of time, then returns to the program which called it. The amount of delay is specified by the contents of the accumulator. With A the contents of the accumulator, the delay is $\frac{1}{2}(26+27A+5A^2)$ microseconds. WAIT returns with the A register zeroed and the X and Y registers undisturbed.

\$F864 SETCOL Set Low-Res Graphics color

This subroutine sets the color used for plotting on the Low-Res screen to the color passed in the accumulator. See page 17 for a table of Low-Res colors.

\$F85F NEXTCOL Increment color by 3

This adds 3 to the current color used for Low-Res Graphics.

\$F800 PLOT Plot a block on the Low-Res screen

This subroutine plots a single block on the Low-Res screen of the prespecified color. The block's vertical position is passed in the accumulator, its horizontal position in the Y register. PLOT returns with the accumulator scrambled, but X and Y unmolested.

\$F819 HLINE Draw a horizontal line of blocks

This subroutine draws a horizontal line of blocks of the predetermined color on the Low-Res screen. You should call HLINE with the vertical coordinate of the line in the accumulator, the leftmost horizontal coordinate in the Y register, and the rightmost horizontal coordinate in location \$2C. HLINE returns with A and Y scrambled, X intact.

\$F828 VLINE Draw a vertical line of blocks

This subroutine draws a vertical line of blocks of the predetermined color on the Low-Res screen. You should call VLINE with the horizontal coordinate of the line in the Y register, the top vertical coordinate in the accumulator, and the bottom vertical coordinate in location \$2D. VLINE will return with the accumulator scrambled.

\$F832 CLRSCR Clear the entire Low-Res screen

CLRSCR clears the entire Low-resolution Graphics screen. If you call CLRSCR while the video display is in Text mode, it will fill the screen with inverse-mode "@" characters. CLRSCR destroys the contents of A and Y.

\$F836 CLRTOP Clear the top of the Low-Res screen

CLRTOP is the same as CLRSCR (above), except that it clears only the top 40 rows of the screen.

\$F871 **SCRN** **Read the Low-Res screen**

This subroutine returns the color of a single block on the Low-Res screen. Call it as you would call PLOT (above). The color of the block will be returned in the accumulator. No other registers are changed.

\$FB1E **PREAD** **Read a Game Controller**

PREAD will return a number which represents the position of a game controller. You should pass the number of the game controller (0 to 3) in the X register. If this number is not valid, strange things may happen. PREAD returns with a number from \$00 to \$FF in the Y register. The accumulator is scrambled.

\$FF2D **PRERR** **Print "ERR"**

Sends the word "ERR", followed by a bell character, to the standard output device. The accumulator is scrambled.

\$FF4A **IOSAVE** **Save all registers**

The contents of the 6502's internal registers are saved in locations \$45 through \$49 in the order A-X-Y-P-S. The contents of A and X are changed; the decimal mode is cleared.

\$FF3F **IOREST** **Restore all registers**

The contents of the 6502's internal registers are loaded from locations \$45 through \$49.

MONITOR SPECIAL LOCATIONS

Table 14: Page Three Monitor Locations			
Address: Decimal	Hex	Use: Monitor ROM	Autostart ROM
1008	\$3F0		Holds the address of the subroutine which handles machine language "BRK" requests (normally SFA59).
1009	\$3F1	None.	
1010	\$3F2		Soft Entry Vector.
1011	\$3F3		
1012	\$3F4	None.	Power-up Byte.
1013	\$3F5		Holds a "JuMP" instruction to the subroutine which handles Applesoft II "&" commands.* Normally \$4C \$58 \$FF.
1014	\$3F6		
1015	\$3F7		
1016	\$3F8		Holds a "JuMP" instruction to the subroutine which handles "USER" (CTRL Y) commands.
1017	\$3F9		
1018	\$3FA		
1019	\$3FB		Holds a "JuMP" instruction to the subroutine which handles Non- Maskable Interrupts.
1020	\$3FC		
1021	\$3FD		
1022	\$3FE		Holds the address of the subroutine which handles Interrupt ReQuests.
1023	\$3FF		

* See page 123 in the **Applesoft II BASIC Reference Manual**.

MINI-ASSEMBLER INSTRUCTION FORMATS

The Apple Mini-Assembler recognizes 56 mnemonics and 13 addressing formats used in 6502 Assembly language programming. The mnemonics are standard, as used in the **MOS Technology/Synertek 6500 Programming Manual** (Apple part number A2L0003), but the addressing formats are different. Here are the Apple standard address mode formats for 6502 Assembly Language:

Table 15: Mini-Assembler Address Formats	
Mode:	Format:
Accumulator	None.
Immediate	<code>#\${value}</code>
Absolute	<code> \${address}</code>
Zero Page	<code> \${address}</code>
Indexed Zero Page	<code> \${address},X</code> <code> \${address},Y</code>
Indexed Absolute	<code> \${address},X</code> <code> \${address},Y</code>
Implied	None.
Relative	<code> \${address}</code>
Indexed Indirect	<code>(\${address},X)</code>
Indirect Indexed	<code>(\${address}),Y</code>
Absolute Indirect	<code>(\${address})</code>

An `{address}` consists of one or more hexadecimal digits. The Mini-Assembler interprets addresses in the same manner that the Monitor does: if an address has fewer than four digits, it adds leading zeroes; if it has more than four digits, then it uses only the last four.

All dollar signs (\$), signifying that the addresses are in hexadecimal notation, are ignored by the Mini-Assembler and may be omitted.

There is no syntactical distinction between the Absolute and Zero Page addressing modes. If you give an instruction to the Mini-Assembler which can be used in both Absolute and Zero-Page mode, then the Mini-Assembler will assemble that instruction in Absolute mode if the operand for that instruction is greater than \$FF, and it will assemble that instruction in Zero Page mode if the operand for that instruction is less than \$0100.

Instructions with the Accumulator and Implied addressing modes need no operand.

Branch instructions, which use the Relative addressing mode, require the *target address* of the branch. The Mini-Assembler will automatically figure out the relative distance to use in the instruction. If the target address is more than 127 locations distant from the instruction, then the Mini-Assembler will sound a "beep", place a circumflex (^) under the target address, and ignore the line.

If you give the Mini-Assembler the mnemonic for an instruction and an operand, and the addressing mode of the operand cannot be used with the instruction you entered, then the Mini-Assembler will not accept the line.

CHAPTER 4

MEMORY ORGANIZATION

- 68 RAM STORAGE
- 70 RAM CONFIGURATION BLOCKS
- 72 ROM STORAGE
- 73 I/O LOCATIONS
- 74 ZERO PAGE MEMORY MAPS

The Apple's 6502 microprocessor can directly reference a total of 65,536 distinct memory locations. You can think of the Apple's memory as a book with 256 "pages", with 256 memory locations on each page. For example, "page \$30" is the 256 memory locations beginning at location \$3000 and ending at location \$30FF. Since the 6502 uses two eight-bit bytes to form the address of any memory location, you can think of one of the bytes as the *page number* and the other as the *location within the page*.

The Apple's 256 pages of memory fall into three categories: Random Access Memory (RAM), Read-Only Memory (ROM), and Input/Output locations (I/O). Different areas of memory are dedicated to different functions. The Apple's basic memory map looks like this:

System Memory Map		
Page Number:	Decimal	Hex
0	\$00	
1	\$01	
2	\$02	
.	.	RAM (48K)
.	.	
190	\$BE	
191	\$BF	
192	SC0	
193	SC1	
.	.	I/O (2K)
.	.	
198	SC6	
199	SC7	
200	SC8	
201	SC9	
.	.	I/O ROM (2K)
.	.	
206	SCE	
207	SCF	
208	SD0	
209	SD1	
.	.	ROM (12K)
.	.	
254	SFE	
255	SFF	

Figure 5. System Memory Map

RAM STORAGE

The area in the Apple's memory map which is allocated for RAM memory begins at the bottom

of Page Zero and extends up to the end of Page 191. The Apple has the capacity to house from 4K (4,096 bytes) to 48K (49,152 bytes) of RAM on its main circuit board. In addition, you can expand the RAM memory of your Apple all the way up to 64K (65,536 bytes) by installing an Apple Language Card (part number A2B0006). This extra 16K of RAM takes the place of the Apple's ROM memory, with two 4K segments of RAM sharing the 4K range from \$D000 to \$DFFF.

Most of your Apple's RAM memory is available to you for the storage of programs and data. The Apple, however, does reserve some locations in RAM for use of the System Monitor, various languages, and other system functions. Here is a map of the available areas in RAM memory:

Table 16: RAM Organization and Usage		
Page Number: Decimal Hex	Used For:	
0 \$00	System Programs	
1 \$01	System Stack	
2 \$02	GETLN Input Buffer	
3 \$03	Monitor Vector Locations	
4 \$04		
5 \$05	Text and Lo-Res Graphics	
6 \$06	Primary Page Storage	
7 \$07		
8 \$08		
9 \$09	Text and Lo-Res Graphics	
10 \$0A	Secondary Page Storage	
11 \$0B		
12 \$0C through 31 \$1F		FREE
32 \$20 through 63 \$3F	Hi-Res Graphics Primary Page Storage	RAM
64 \$40 through 95 \$5F	Hi-Res Graphics Secondary Page Storage	
96 \$60 through 191 \$BF		

Following is a breakdown of which ranges are assigned to which functions:

Zero Page. Due to the construction of the Apple's 6502 microprocessor, the lowermost page in the Apple's memory is prime real estate for machine language programs. The System Monitor uses about 20 locations on Page Zero; Apple Integer BASIC uses a few more; and Applesoft II BASIC and the Apple Disk Operating System use the rest. Tables 18, 19, 20, and 21 show the locations on zero page which are used by these system functions.

Page One. The Apple's 6502 microprocessor reserves all 256 bytes of Page 1 for use as a "stack". Even though the Apple usually uses less than half of this page at any one time, it is not easy to determine just what is being used and what is lying fallow, so you shouldn't try to use

Page 1 to store any data.

Page Two. The GETLN subroutine, which is used to get input lines by most programs and languages, uses Page 2 as its input buffer. If you're sure that you won't be typing any long input lines, then you can (somewhat) safely store temporary data in the upper regions of Page 2.

Page Three. The Apple's Monitor ROM (both the Autostart and the original) use the upper sixteen locations in Page Three, from location \$3F0 to \$3FF (decimal 1008 to 1023). The Monitor's use of these locations is outlined on page 62.

Pages Four through Seven. This 1,024-byte range of memory locations is used for the Text and Low-Resolution Graphics Primary Page display, and is therefore unusable for storage purposes. There are 64 locations in this range which are not displayed on the screen. These 64 locations are reserved for use by the peripheral cards (see page 82).

RAM CONFIGURATION BLOCKS

The Apple's RAM memory is composed of eight to 24 integrated circuits. These IC's reside in three rows of sockets on the Apple board. Each row can hold eight chips of either the 4,096-bit (4K) or 16,384-bit (16K) variety. The 4K RAM chips are of the Mostek "4096" family, and may be marked "MK4096" or "MCM6604". The 16K chips are of the "4116" type, and may have the denomination "MK4116" or "UPD4160". Each row must have eight of the same type of chip, although different rows may hold different types.

A row of eight 16K IC's represents 16,384 eight-bit bytes of RAM. The leftmost IC in a row represents the lowermost (least significant) bit of every byte in that range, and the rightmost IC in a row represents the uppermost (most significant) bit of every byte. The row of RAM IC's which is frontmost on the Apple board holds the RAM memory which begins at location 0 in the memory map; the next row back continues where the first left off.

You can tell the Apple how much memory it has, and of what type it is, by plugging *Memory Configuration Blocks* into three IC sockets on the left side of the Apple board. These configuration blocks are three 14-legged critters which look like big, boxy integrated circuits. But there are no chips inside of them; only three jumper wires in each. The jumper wires "strap" each row of RAM chips into a specific place in the Apple's memory map. All three configuration blocks should be strapped the same way. Apple supplies several types of standard configuration blocks for the most common system sizes. A set of these was installed in your Apple when it was built, and you get a new set each time you purchase additional memory for your Apple. If, however, you want to expand your Apple's memory with some RAM chips that you did not purchase from Apple, you may have to construct your own configuration blocks (or modify the ones already in your Apple).

There are nine different RAM memory configurations possible in your Apple. These nine memory sizes are made up from various combinations of 4K and 16K RAM chips in the three rows of sockets in your Apple. The nine memory configurations are:

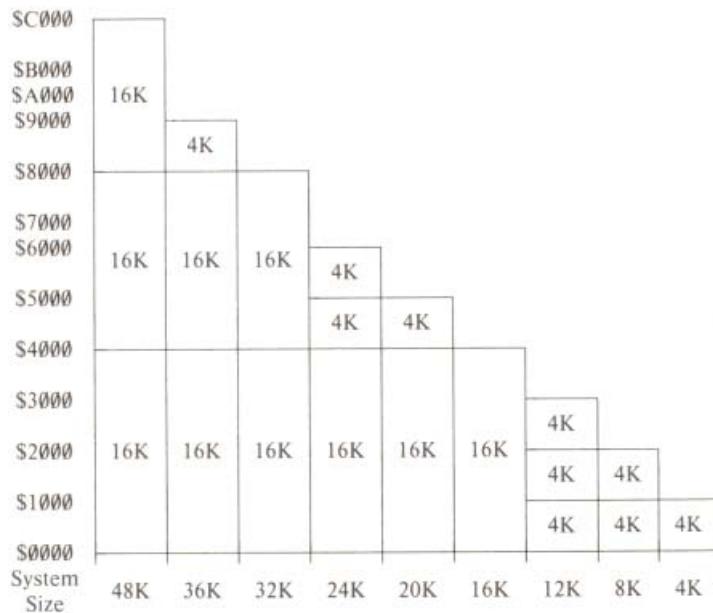


Figure 6. Memory Configurations

Of the fourteen "legs" on each controller block, the three in the upper-right corner (looking at it from above) represent the three rows of RAM chips on the Apple's main board. There should be a wire jumper from each one of these pins to another pin in the configuration block. The "other pin" corresponds to a place in the Apple's memory map where you want the RAM chips in each row to reside. The pins on the configuration block are represented thus:

4K range \$0000-\$0FFF	1	○	14	Frontmost row ("C")
4K range \$1000-\$1FFF	2		13	Middle row ("D")
4K range \$2000-\$2FFF	3		12	Backmost row ("E")
4K range \$3000-\$3FFF	4		11	No connection.
4K range \$4000-\$4FFF	5		10	16K range \$0000-\$3FFF
4K range \$5000-\$5FFF	6		9	16K range \$4000-\$7FFF
4K range \$8000-\$BFFF	7		8	16K range \$8000-\$BFFF

Figure 7. Memory Configuration Block Pinouts

If a row contains eight chips of the 16K variety, then you should connect a jumper wire from the pin corresponding to that row to a pin corresponding to a 16K range of memory. Similarly, if a row contains eight 4K chips, you should connect a jumper wire from the pin for that row to a pin corresponding to a 4K range of memory. You should *never* put 4K chips in a row strapped for 16K, or vice versa. It is also not advisable to leave a row unstrapped, or to strap two rows into the same range of memory.

You should always make sure that there is some kind of memory beginning at location 0. Your Apple's memory should be in one contiguous block, but it does not need to be. For example, if you have only three sets of 4K chips, but you want to use the primary page of the High-

Resolution Graphics mode, then you would strap one row of 4K chips to the beginning of memory (4K range \$0000 through \$0FFF), and strap the other two rows to the memory range used by the High-Resolution Graphics primary page (4K ranges \$2000 through \$2FFF and \$3000 through \$3FFF). This will give you 4K bytes of RAM memory to work with, and 8K bytes of RAM to be used as a picture buffer.

Notice that the configuration blocks are installed into the Apple with their front edges (the edge with the white dot, representing pin 1) towards the front of the Apple.

There is a problem in Apples with Revision 0 boards and 20K or 24K of RAM. In these systems, the 8K range of the memory map from \$4000 to \$5FFF is duplicated in the memory range \$6000 to \$7FFF, regardless of whether it contains RAM or not. So systems with only 20K or 24K of RAM would appear to have 24K or 36K, but this extra RAM would be only imaginary. This has been changed in the Revision 1 Apple boards.

ROM STORAGE

The Apple, in its natural state, can hold from 2K (2,048 bytes) to 12K (12,288 bytes) of Read-Only memory on its main board. This ROM memory can include the System Monitor, a couple of dialects of the BASIC language, various system and utility programs, or pre-packaged subroutines such as are included in Apple's *Programmer's Aid #1* ROM.

The Apple's ROM memory resides in the top 12K (48 pages) of the memory map, beginning at location \$D000. For proper operation of the Apple, there must be some kind of ROM in the uppermost locations of memory. When you turn on the Apple's power supply, the microprocessor must have some program to execute. It goes to the top locations in the memory map for the address of this program. In the Apple, this address is stored in ROM, and is the address of a program within the same ROM. This program initializes the Apple and lets you start to use it. (For a description of the startup cycle, see "The RESET Cycle", page 36.)

Here is a map of the Apple's ROM memory, and of the programs and packages that Apple currently supports in ROM:

Table 17: ROM Organization and Usage		
Page Number: Decimal	Hex	Used By:
208	\$D0	
212	\$D4	Programmer's Aid #1
216	\$D8	
220	\$DC	
224	\$E0	
228	\$E4	
232	\$E8	Integer BASIC
236	\$EC	
240	\$F0	
244	\$F4	Utility Subroutines
248	\$F8	
252	\$FC	Monitor ROM
		Autostart ROM

Six 24-pin IC sockets on the Apple's board hold the ROM integrated circuits. Each socket can hold one of a type 9316B 2,048-byte by 8-bit Read-Only Memory. The leftmost ROM in the Apple's board holds the upper 2K of ROM in the Apple's memory map; the rightmost ROM IC holds the ROM memory beginning at page \$D0 in the memory map. If a ROM is not present in a given socket, then the values contained in the memory range corresponding to that socket will be unpredictable.

The Apple Firmware card can disable some or all of the ROMs on the Apple board, and substitute its own ROMs in their place. When you have an Apple Firmware card installed in any slot in the Apple's board, you can disable the Apple's on-board ROMs by flipping the card's controller switch to its UP position and pressing and releasing the [RESET] button, or by referencing location \$C080 (decimal 49280 or -16256). To enable the Apple's on-board ROMs again, flip the controller switch to the DOWN position and press [RESET], or reference location \$C081 (decimal 49281 or -16255). For more information, see Appendix A of the **Applesoft II BASIC Programming Reference Manual**.

I/O LOCATIONS

4,096 memory locations (16 pages) of the Apple's memory map are dedicated to input and output functions. This 4K range begins at location \$C000 (decimal 49152 or -16384) and extends on up to location \$CFFF (decimal 53247 or -12289). Since these functions are somewhat intricate, they have been given a chapter all to themselves. Please see Chapter 5 for information on the allocation of Input/Output locations.

ZERO PAGE MEMORY MAPS

Table 18: Monitor Zero Page Usage

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
0	\$00															
16	\$10															
32	\$20	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
48	\$30	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
64	\$40	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
80	\$50	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
96	\$60															
112	\$70															
128	\$80															
144	\$90															
160	\$A0															
176	\$B0															
192	\$C0															
208	\$D0															
224	\$E0															
240	\$F0															

Table 19: Applesoft II BASIC Zero Page Usage

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
0	\$00	•	•	•	•	•	•				•	•	•	•	•	•
16	\$10	•	•	•	•	•	•	•	•	•						
32	\$20															
48	\$30															
64	\$40															
80	\$50	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
96	\$60	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
112	\$70	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
128	\$80	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
144	\$90	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
160	\$A0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
176	\$B0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
192	\$C0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
208	\$D0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
224	\$E0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
240	\$F0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Table 20: Apple DOS 3.2 Zero Page Usage

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	Hex	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
0	\$00																
16	\$10																
32	\$20																
48	\$30																
64	\$40	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
80	\$50																
96	\$60																
112	\$70	•															
128	\$80																
144	\$90																
160	\$A0																
176	\$B0	•															
192	\$C0																
208	\$D0								•		•	•	•	•			
224	\$E0																
240	\$F0																

Table 21: Integer BASIC Zero Page Usage

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	Hex	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
0	\$00																
16	\$10																
32	\$20																
48	\$30																
64	\$40																
80	\$50																
96	\$60	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
112	\$70	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
128	\$80	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
144	\$90	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
160	\$A0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
176	\$B0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
192	\$C0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
208	\$D0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
224	\$E0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
240	\$F0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	

CHAPTER 5

INPUT/OUTPUT STRUCTURE

- 78 BUILT-IN I/O
- 79 PERIPHERAL BOARD I/O
- 80 PERIPHERAL CARD I/O SPACE
- 80 PERIPHERAL CARD ROM SPACE
- 81 I/O PROGRAMMING SUGGESTIONS
- 82 PERIPHERAL SLOT SCRATCHPAD RAM
- 83 THE CSW/KSW SWITCHES
- 84 EXPANSION ROM

The Apple's Input and Output functions fall into two basic categories: those functions which are performed on the Apple's board itself, and those functions which are performed by peripheral interface cards plugged into the Apple's eight peripheral "slots". Both of these functions communicate to the microprocessor and your programs via 4,096 locations in the Apple's memory map. This chapter describes the memory mapping and operation of the various input and output controls and functions; the hardware which executes these functions is described in the next chapter.

BUILT-IN I/O

Most of the Apple's inherent I/O facilities are described briefly in Chapter 1, "Approaching your Apple". For a short description of these facilities, please see that chapter.

The Apple's on-board I/O functions are controlled by 128 memory locations in the Apple's memory map, beginning at location \$C000 and extending up through location \$C07F (decimal 49152 through 49279, or -16384 through -16257). Twenty-seven different functions share these 128 locations. Obviously, some functions are affected by more than one location: in some instances, as many as sixteen different locations all can perform exactly the same function. These 128 locations fall into five types: Data Inputs, Strobes, Soft Switches, Toggle Switches, and Flag Inputs.

Data Inputs. The only Data Input on the Apple board is a location whose value represents the current state of the Apple's built-in keyboard. The uppermost bit of this input is akin to the Flag Inputs (see below); the lower seven bits are the ASCII code of the key which was most recently pressed on the keyboard.

Flag Inputs. Most built-in input locations on the Apple are single-bit 'flags'. These flags appear in the highest (eighth) bit position in their respective memory locations. Flags have only two values: 'on' and 'off'. The setting of a flag can be tested easily from any language. A higher-level language can use a "PEEK" or similar command to read the value of a flag location: if the PEEKed value is greater than or equal to 128, then the flag is on; if the value is less than 128, the flag is off. Machine language programs can load the contents of a flag location into one of the 6502's internal registers (or use the BIT instruction) and branch depending upon the setting of the N (sign) flag. A BMI instruction will cause a branch if the flag is on, and a BPL instruction will cause a branch if the flag is off.

The Single-Bit (Pushbutton) inputs, the Cassette input, the Keyboard Strobe, and the Game Controller inputs are all of this type.

Strobe Outputs. The Utility Strobe, the Clear Keyboard Strobe, and the Game Controller Strobe are all 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. In the case of the Utility Strobe, pin 5 on the Game I/O connector will drop from +5 volts to 0 volts for a period of .98 microseconds, then rise back to +5 again; in the case of the Keyboard Strobe, the Keyboard's flag input (see above) will be turned off; and in the case of the Game Controller Strobe, all of the flag inputs of the Game Controllers will be turned off and their timing loops restarted.

Your program can also trigger the Keyboard and Game Controller Strobes by *writing* to their controlling locations, but you should not write to the Utility Strobe location. If you do, you will produce *two* .98 microsecond pulses, about 24.43 nanoseconds apart. This is due to the method in which the 6502 writes to a memory location: first it reads the contents of that location, then it

writes over them. This double pulse will go unnoticed for the Keyboard and Game Controller Strobes, but may cause problems if it appears on the Utility Strobe.

Toggle Switches. Two other strobe outputs are connected internally to two-state "flip-flops". Each time you read from the location associated with the strobe, its flip-flop will "toggle" to its other state. These toggle switches drive the Cassette Output and the internal Speaker. There is no practical way to determine the setting of an internal toggle switch. Because of the nature of the toggle switches, you should only read from their controlling locations, and not write to them (see Strobe Outputs, above).

Soft Switches. Soft Switches are two-position switches in which each side of the switch is controlled by an individual memory location. If you reference the location for one side of the switch, it will throw the switch that way; if you reference the location for the other side, it will throw the switch the other way. It sets the switch without regard to its former setting, and there is no way to determine the position a soft switch is in. You can safely write to soft switch controlling locations: two pulses are as good as one (see Strobe Outputs, above). The Announcer outputs and all of the Video mode selections are controlled by soft switches.

The special memory locations which control the built-in Input and Output functions are arranged thus:

	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
SC000	Keyboard Data Input															
SC010	Clear Keyboard Strobe															
SC020	Cassette Output Toggle															
SC030	Speaker Toggle															
SC040	Utility Strobe															
SC050	gr	tx	nomix	mix	pri	sec	lores	hiress	an0	an1	an2	an3				
SC060	cin	pb1	pb2	pb3	gc0	gc1	gc2	gc3							repeat SC060-SC067	
SC070	Game Controller Strobe															

Key to abbreviations:

gr	Set GRAPHICS mode	tx	Set TEXT mode
nomix	Set all text or graphics	mix	Mix text and graphics
pri	Display primary page	sec	Display secondary page
lores	Display Low-Res Graphics	hiress	Display Hi-Res Graphics
an	Announcer outputs	pb	Pushbutton inputs
gc	Game Controller inputs	cin	Cassette Input

PERIPHERAL BOARD I/O

Along the back of the Apple's main board is a row of eight long "slots", or Peripheral Connectors. Into seven of these eight slots, you can plug any of many Peripheral Interface boards designed especially for the Apple. In order to make the peripheral cards simpler and more versatile, the Apple's circuitry has allocated a total of 280 byte locations in the memory map for each

of seven slots. There is also a 2K byte "common area", which all peripheral cards in your Apple can share.

Each slot on the board is individually numbered, with the leftmost slot called "Slot 0" and the rightmost called "Slot 7". Slot 0 is special: it is meant for RAM, ROM, or Interface expansion. All other slots (1 through 7) have special control lines going to them which are active at different times for different slots.

PERIPHERAL CARD I/O SPACE

Each slot is given sixteen locations beginning at location \$C080 for general input and output purposes. For slot 0, these sixteen locations fall in the memory range \$C080 through \$C08F; for slot 1, they're in the range \$C090 through \$C09F, *et cetera*. Each peripheral card can use these locations as it pleases. Each peripheral card can determine when it is being selected by listening to pin 41 (called DEVICE SELECT) on its peripheral connector. Whenever the voltage on this pin drops to 0 volts, the address which the microprocessor is calling is somewhere in that peripheral card's 16-byte allocation. The peripheral card can then look at the bottom four address lines to determine which of its sixteen addresses is being called.

Table 23: Peripheral Card I/O Locations

	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
\$C080											0					
\$C090											1					
\$C0A0											2					
\$C0B0											3					
\$C0C0											4					
\$C0D0											5					
\$C0E0											6					
\$C0F0											7					

Input/Output for slot number

PERIPHERAL CARD ROM SPACE

Each peripheral slot also has reserved for it one 256-byte page of memory. This page is usually used to house 256 bytes of ROM or Programmable ROM (PROM) memory, which contains driving programs or subroutines for the peripheral card. In this way, the peripheral interface cards can be "intelligent": they contain their own driving software; you do not need to load separate programs in order to use the interface cards.

The page of memory reserved for each peripheral slot has the page number \$Cn, where *n* is the slot number. Slot 0 does not have a page reserved for it, so you cannot use most Apple interface cards in that slot. The signal on Pin 1 (called I/O SELECT) of each peripheral slot will become active (drop from +5 volts to ground) when the microprocessor is referencing an address within that slot's reserved page. Peripheral cards can use this signal to enable their PROMs, and use the lower eight address lines to address each byte in the PROM.

	Table 24: Peripheral Card PROM Locations															
	\$00	\$10	\$20	\$30	\$40	\$50	\$60	\$70	\$80	\$90	\$A0	\$B0	\$C0	\$D0	\$E0	\$F0
SC100											1					
SC200											2					
SC300											3					
SC400											PROM space for slot number	4				
SC500												5				
SC600												6				
SC700												7				

I/O PROGRAMMING SUGGESTIONS

The programs in peripheral card PROMs should be portable; that is, they should be able to function correctly regardless of where they are placed in the Apple's memory map. They should contain no absolute references to themselves. They should perform all JUMPs with conditional or forced branches.

Of course, you can fill a peripheral card PROM with subroutines which are *not* portable, and your only loss would be that the peripheral card would be slot-dependent. If you're cramped for space in a peripheral card PROM, you can save many bytes by making the subroutines slot-dependent.

The first thing that a subroutine in a peripheral card PROM should do is to save the values of *all* of the 6502's internal registers. There is a subroutine called IOSAVE in the Apple's Monitor ROM which does just this. It saves the contents of all internal registers in memory locations \$45 through \$49, in the order A-X-Y-P-S. This subroutine starts at location \$FF4A. A companion subroutine, called IORESTORE, restores *all* of the internal registers from these storage locations. You should call this subroutine, located at \$FF3F, before your PROM subroutine finishes.

Most single-character input and output is passed in the 6502's Accumulator. During output, the character to be displayed is in the Accumulator, with its high bit set. During input, your subroutine should pass the character received from the input device in the Accumulator, also with its high bit set.

A program in a peripheral card's PROM can determine which slot the card is plugged into by executing this sequence of instructions:

```

0300 - 20 4A FF      JSR    $FF4A
0303 - 78             SEI
0304 - 20 58 FF      JSR    $FF58
0307 - BA             TSX
0308 - BD 00 01      LDA    $0100,X
030B - 8D F8 07      STA    $07F8
030E - 29 0F           AND   #$0F
0310 - A8             TAY

```

After a program executes these steps, the slot number which its card is in will be stored in the 6502's Y index register in the format \$0n, where n is the slot number. A program in the ROM can further process this value by shifting it four bits to the left, to obtain \$n0.

0311 - 98 TYA

0312 -	0A	ASL
0313 -	0A	ASL
0314 -	0A	ASL
0315 -	0A	ASL
0316 -	AA	TAX

A program can use this number in the X index register with the 6502's indexed addressing mode to refer to the sixteen I/O locations reserved for each card. For example, the instruction

0317 - BD 80 C0 LDA \$C080,X

will load the 6502's accumulator with the contents of the first I/O location used by the peripheral card. The address SC080 is the *base address* for the first location used by all eight peripheral slots. The address SC081 is the base address for the second I/O location, and so on. Here are the base addresses for all sixteen I/O locations on each card:

Base Address	Slot							
	0	1	2	3	4	5	6	7
SC080	SC080	SC090	SC0A0	SC0B0	SC0C0	SC0D0	SC0E0	SC0F0
SC081	SC081	SC091	SC0A1	SC0B1	SC0C1	SC0D1	SC0E1	SC0F1
SC082	SC082	SC092	SC0A2	SC0B2	SC0C2	SC0D2	SC0E2	SC0F2
SC083	SC083	SC093	SC0A3	SC0B3	SC0C3	SC0D3	SC0E3	SC0F3
SC084	SC084	SC094	SC0A4	SC0B4	SC0C4	SC0D4	SC0E4	SC0F4
SC085	SC085	SC095	SC0A5	SC0B5	SC0C5	SC0D5	SC0E5	SC0F5
SC086	SC086	SC096	SC0A6	SC0B6	SC0C6	SC0D6	SC0E6	SC0F6
SC087	SC087	SC097	SC0A7	SC0B7	SC0C7	SC0D7	SC0E7	SC0F7
SC088	SC088	SC098	SC0A8	SC0B8	SC0C8	SC0D8	SC0E8	SC0F8
SC089	SC089	SC099	SC0A9	SC0B9	SC0C9	SC0D9	SC0E9	SC0F9
SC08A	SC08A	SC09A	SC0AA	SC0BA	SC0CA	SC0DA	SC0EA	SC0FA
SC08B	SC08B	SC09B	SC0AB	SC0BB	SC0CB	SC0DB	SC0EB	SC0FB
SC08C	SC08C	SC09C	SC0AC	SC0BC	SC0CC	SC0DC	SC0EC	SC0FC
SC08D	SC08D	SC09D	SC0AD	SC0BD	SC0CD	SC0DD	SC0ED	SC0FD
SC08E	SC08E	SC09E	SC0AE	SC0BE	SC0CE	SC0DE	SC0EE	SC0FE
SC08F	SC08F	SC09F	SC0AF	SC0BF	SC0CF	SC0DF	SC0EF	SC0FF
I/O Locations								

PERIPHERAL SLOT SCRATCHPAD RAM

Each of the eight peripheral slots has reserved for it 8 locations in the Apple's RAM memory. These 64 locations are actually in memory pages \$04 through \$07, inside the area reserved for the Text and Low-Resolution Graphics video display. The contents of these locations, however, are *not* displayed on the screen, and their contents are not changed by normal screen operations.* The peripheral cards can use these locations for temporary storage of data while the cards are in operation. These "scratchpad" locations have the following addresses:

* See "But Soft...", page 31.

Base Address	Slot Number						
	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

Slot 0 does not have any scratchpad RAM addresses reserved for it. The Base Address locations are used by Apple DOS 3.2 and are also shared by all peripheral cards. Some of these locations have dedicated functions: location \$7F8 holds the slot number (in the format \$Cn) of the peripheral card which is currently active, and location \$5F8 holds the slot number of the disk controller card from which any active DOS was booted.

By using the slot number \$0n, derived in the program example above, a subroutine can directly reference any of its eight scratchpad locations:

```

031A- B9 78 04    LDA   $0478,Y
031D- 99 F8 04    STA   $04F8,Y
0320- B9 78 05    LDA   $0578,Y
0323- 99 F8 05    STA   $05F8,Y
0326- B9 78 06    LDA   $0678,Y
0329- 99 F8 06    STA   $06F8,Y
032C- B9 78 07    LDA   $0778,Y
032F- 99 F8 07    STA   $07F8,Y

```

THE CSW/KSW SWITCHES

The pair of locations \$36 and \$37 (decimal 54 and 55) is called CSW, for "Character output SWitch". Individually, location \$36 is called CSWL (CSW Low) and location \$37 is called CSWH (CSW High). This pair of locations holds the address of the subroutine which the Apple is currently using for single-character output. This address is normally \$FDF0, the address of the COUT subroutine (see page 30). The Monitor's PRINTER ([CTRL P]) command, and the BASIC command PR#, can change this address to be the address of a subroutine in a PROM on a peripheral card. Both of these commands put the address \$Cn00 into this pair of locations, where n is the slot number given in the command. This is the address of the first location in whatever PROM happens to be on the peripheral card plugged into that slot. The Apple will then call this subroutine every time it wishes to output one character. This subroutine can use the instruction sequences given above to find its slot number and use the I/O and RAM scratchpad locations for its slot. When it is finished, it can either execute an RTS (ReTurn from Subroutine) instruction, to return to the program or language which is sending the output, or it can jump to the COUT subroutine at location \$FDF0, to display the character on the screen and then return to the program which is producing output.

Similarly, locations \$38 and 39 (decimal 56 and 57), called KSWL and KSWH separately or KSW

(Keyboard input SWitch) together, hold the address of the subroutine the Apple is currently using for single-character input. This address is normally \$FD1B, the address of the KEYIN subroutine. The Monitor's KEYBOARD command (**CTRL K**) and the BASIC command IN# both change this address to \$Cn00, again with *n* the slot number given in the command. The Apple will call the subroutine at the beginning of the PROM on the peripheral card in this slot whenever it wishes to get a single character from the input device. The subroutine should place the input character into the 6502's accumulator and ReTurn from Subroutine (RTS). The subroutine should set the high bit of the character before it returns.

The subroutines in a peripheral card's PROM can change the addresses in the CSW and KSW switches to point to places in the PROM other than the very beginning. For example, a certain PROM could begin with a segment of code to determine what slot it is in and do some initialization, and then jump in to the actual character handling subroutine. As part of its initialization sequence, it could change KSW or CSW (whichever is applicable) to point directly to the beginning of the character handling subroutine. Then the next time the Apple asks for input or output from that card, the handling subroutines will skip the already-done initialization sequence and go right in to the task at hand. This can save time in speed-sensitive situations.

A peripheral card can be used for both input and output if its PROM has separate subroutines for the separate functions and changes CSW and KSW accordingly. The initialization sequence in a peripheral card PROM can determine if it is being called for input or output by looking at the high parts of the CSW and KSW switches. Whichever switch contains \$Cn is currently calling that card to perform its function. If both switches contain \$Cn, then your subroutine should assume that it is being called for output.

EXPANSION ROM

The 2K memory range from location \$C800 to \$CFFF is reserved for a 2K ROM or PROM on a peripheral card, to hold large programs or driving subroutines. The expansion ROM space also has the advantage of being absolutely located in the Apple's memory map, which gives you more freedom in writing your interface programs.

This PROM space is available to all peripheral slots, and more than one card in your Apple can have an expansion ROM. However, only one expansion ROM can be active at one time.

Each peripheral card's expansion ROM should have a flip-flop to enable it. This flip-flop should be turned "on" by the DEVICE SELECT signal (the one which enables the 256-byte PROM). This means that the expansion ROM on any card will be partially enabled after you first reference the card it is on. The other enable to the expansion ROM should be the I/O STROBE line, pin 20 on each peripheral connector. This line becomes active whenever the Apple's microprocessor is referencing a location inside the expansion ROM's domain. When this line becomes active, and the aforementioned flip-flop has been turned "on", then the Apple is referencing the expansion ROM on this particular board (see figure 8).

A peripheral card's 256-byte PROM can gain sole access to the expansion ROM space by referring to location \$CFFF in its initialization subroutine. This location is a special location, and all peripheral cards should recognize it as a signal to turn their flip-flops "off" and disable their expansion ROMs. Of course, this will also disable the expansion ROM on the card which is trying to grab the ROM space, but the ROM will be enabled again when the microprocessor gets another instruction from the 256-byte driving PROM. Now the expansion ROM is enabled, and its space is clear. The driving subroutines can then jump directly into the programs in the ROM, where

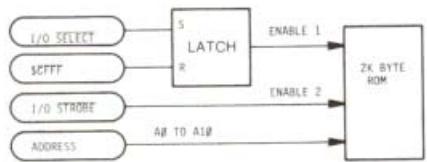


Figure 8. Expansion ROM Enable Circuit

they can enjoy the 2K of unobstructed, absolutely located memory space:

0332 -	2C FF CF	BIT	\$CFFF
0335 -	4C 00 C8	JMP	SC800

It is possible to save circuitry (at the expense of ROM space) on the peripheral card by not fully decoding the special location address, \$CFFF. In fact, if you can afford to lose the last 256 bytes of your ROM space, the following simple circuit will do just fine:

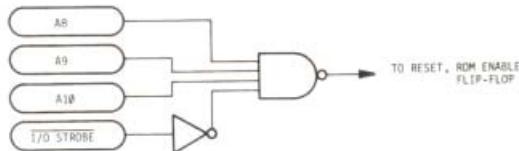


Figure 9. \$CFXX Decoding

III. III.

CHAPTER 6

HARDWARE CONFIGURATION

- 88 THE MICROPROCESSOR
- 90 SYSTEM TIMING
- 92 POWER SUPPLY
- 94 ROM MEMORY
- 95 RAM MEMORY
- 96 THE VIDEO GENERATOR
- 97 VIDEO OUTPUT JACKS
- 98 BUILT-IN I/O
- 99 "USER 1" JUMPER
- 100 THE GAME I/O CONNECTOR
- 100 THE KEYBOARD
- 102 KEYBOARD CONNECTOR
- 103 CASSETTE INTERFACE JACKS
- 104 POWER CONNECTOR
- 105 SPEAKER
- 105 PERIPHERAL CONNECTORS

THE MICROPROCESSOR

The 6502 Microprocessor

Model:	MCS6502/SY6502
Manufactured by:	MOS Technology, Inc. Synertek Rockwell
Number of instructions:	56
Addressing modes:	13
Accumulators:	1 (A)
Index registers:	2 (X,Y)
Other registers:	Stack pointer (S) Processor status (P)
Stack:	256 bytes, fixed
Status flags:	N (sign) C (carry) V (overflow)
Other flags:	I (Interrupt disable) D (Decimal arithmetic) B (Break)
Interrupts:	2 (IRQ, NMI)
Resets:	1 (RES)
Addressing range:	2^{16} (64K) locations
Address bus:	16 bits, parallel
Data bus:	8 bits, parallel Bidirectional
Voltages:	+5 volts
Power dissipation:	.25 watt
Clock frequency:	1.023MHz

The microprocessor gets its main timing signals, Φ_0 and Φ_1 , from the timing circuits described below. These are complimentary 1.023MHz clock signals. Various manuals, including the MOS

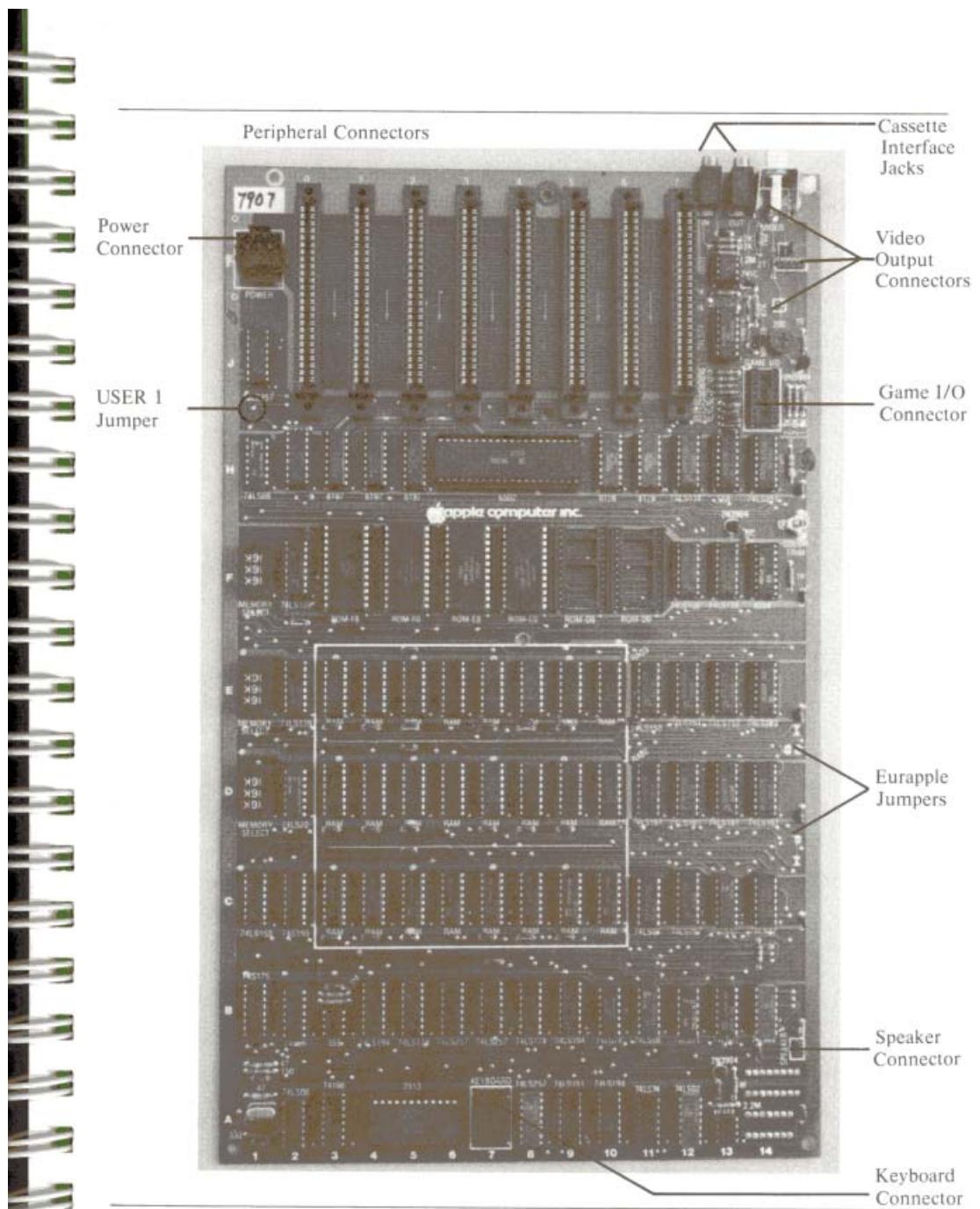


Figure 10. The Apple Main Board

Technology Hardware manual, use the designation Φ_2 for the Apple's $\Phi\emptyset$ clock.

The microprocessor uses its address and data buses only during the time period when $\Phi\emptyset$ is active. When $\Phi\emptyset$ is low, the microprocessor is doing internal operations and does not need the data and address buses.

The microprocessor has a 16-bit address bus and an 8-bit bidirectional data bus. The Address bus lines are buffered by three 8T97 three-state buffers at board locations H3, H4, and H5. The address lines are held open only during a DMA cycle, and are active at all other times. The address on the address bus becomes valid about 300ns after Φ_1 goes high and remains valid through all of $\Phi\emptyset$.

The data bus is buffered through two 8T28 bidirectional three-state buffers at board locations H10 and H11. Data from the microprocessor is put onto the bus about 300ns after Φ_1 and the READ/WRITE signal (R/W) both drop to zero. At all other times, the microprocessor is either listening to or ignoring the data bus.

The RDY, RES, \overline{IRQ} , and \overline{NMI} lines to the microprocessor are all held high by 3.3K Ohm resistors to +5v. These lines also appear on the peripheral connectors (see page 105).

The SET OVERFLOW (SO) line to the microprocessor is permanently tied to ground.

SYSTEM TIMING

Table 27: Timing Signal Descriptions

14M:	Master Oscillator output, 14.318 MHz. All timing signals are derived from this signal.
7M:	Intermediate timing signal, 7.159 MHz.
COLOR REF:	Color reference frequency, 3.580MHz. Used by the video generation circuitry.
$\Phi\emptyset$ (Φ_2) :	Phase \emptyset system clock, 1.023MHz, compliment to Φ_1 .
Φ_1 :	Phase 1 system clock, 1.023 MHz, compliment to $\Phi\emptyset$.
Q3:	A general-purpose timing signal, twice the frequency of the system clocks, but asymmetrical.

All peripheral connectors get the timing signals 7M, $\Phi\emptyset$, Φ_1 , and Q3. The timing signals 14M and COLOR REF are not available on the peripheral connectors.

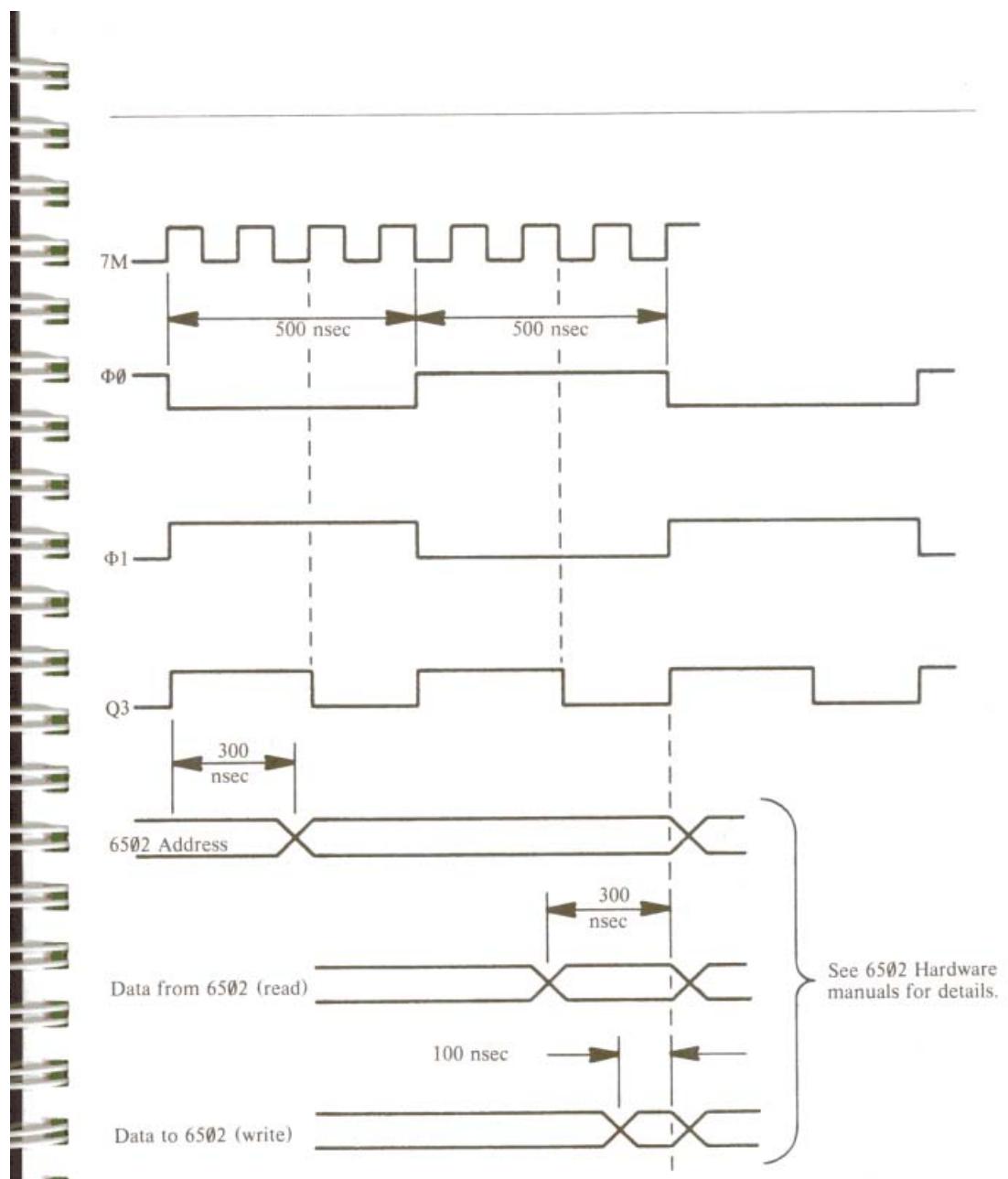


Figure 11. Timing Signals and Relationships

POWER SUPPLY

The Apple Power Supply (U. S. Patent #4,130,862)

Input voltage: 107 VAC to 132 VAC, or
214 VAC to 264 VAC
(switch selectable*)

Supply voltages: +5.0
+11.8
-12.0
-5.2

Power Consumption: 60 watts max. (full load)
79 watts max. (intermittent**)

Full load power output: +5v: 2.5 amp
-5v: 250ma
+12v: 1.5 amp (~ 2.5 amp intermittent**)
-12v: 250ma

Operating temperature: 55c (131° Farenheit)

The Apple Power Supply is a high-voltage "switching" power supply. While most other power supplies use a large transformer with many windings to convert the input voltage into many lesser voltages and then rectify and regulate these lesser voltages, the Apple power supply first converts the AC line voltage into a DC voltage, and then uses this DC voltage to drive a high-frequency oscillator. The output of this oscillator is fed into a small transformer with many windings. The voltages on the secondary windings are then regulated to become the output voltages.

The +5 volt output voltage is compared to a reference voltage, and the difference error is fed back into the oscillator circuit. When the power supply's output starts to move out of its tolerances, the frequency of the oscillator is altered and the voltages return to their normal levels.

If by chance one of the output voltages of the power supply is short-circuited, a feedback circuit in the power supply stops the oscillator and cuts all output circuits. The power supply then pauses for about $\frac{1}{2}$ second and then attempts to restart the oscillations. If the output is still shorted, it will stop and wait again. It will continue this cycle until the short circuit is removed or the power is turned off.

If the output connector of the power supply is disconnected from the Apple board, the power supply will notice this "no load" condition and effectively short-circuit itself. This activates the protection circuits described above, and cuts all power output. This prevents damage to the power supply's internals.

* The voltage selector switch is not present on some Apples.

** The power supply can run 20 minutes with an intermittent load if followed by 10 minutes at normal load without damage.

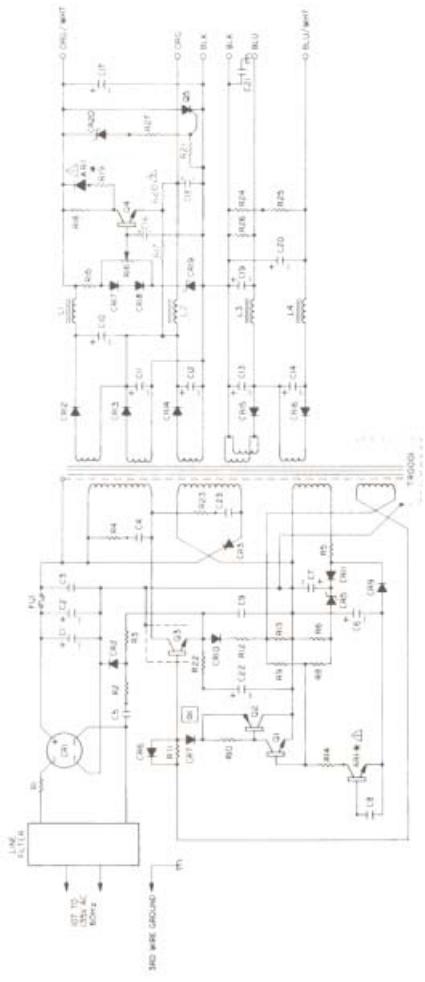


Figure 12. Power Supply Schematic Drawing

If one of the output voltages leaves its tolerance range, due to any problem either within or external to the power supply, it will again shut itself down to prevent damage to the components on the Apple board. This insures that all voltages will either be correct and in proportion, or they will be shut off.

When one of the above fault conditions occurs, the internal protection circuits will stop the oscillations which drive the transformer. After a short while, the power supply will perform a restart cycle, and attempt to oscillate again. If the fault condition has not been removed, the supply will again shut down. This cycle can continue infinitely without damage to the power supply. Each time the oscillator shuts down and restarts, its frequency passes through the audible range and you can hear the power supply squeal and squeak. Thus, when a fault occurs, you will hear a steady "click click click" emanating from the power supply. This is your warning that something is wrong with one of the voltage outputs.

Under no circumstances should you apply more than 140 VAC to the input of the transformer (or more than 280 VAC when the supply's switch is in the 220V position). Permanent damage to the supply will result.

You should connect your Apple's power supply to a properly grounded 3-wire outlet. It is very important that the Apple be connected to a good earth ground.

CAUTION: There are dangerous high voltages inside the power supply's case. Much of the internal circuitry is *not* isolated from the power line, and special equipment is needed for service. **DO NOT ATTEMPT TO REPAIR YOUR POWER SUPPLY!** Send it to your Apple dealer for service.

ROM MEMORY

The Apple can support up to six 2K by 8 mask programmed Read-Only Memory ICs. One of these six ROMs is enabled by a 74LS138 at location F12 on the Apple's board whenever the microprocessor's address bus holds an address between \$D000 and SFFFF. The eight Data outputs of all ROMs are connected to the microprocessor's data line buffers, and the ROM's address lines are connected to the buffers driving the microprocessor's address lines A0 through A10.

The ROMs have three "chip select" lines to enable them. CS1 and CS3, both active low, are connected together to the 74LS138 at location F12 which selects the individual ROMs. CS2, which is active high, is common to all ROMs and is connected to the INH (ROM Inhibit) line on the peripheral connectors. If a card in any peripheral slot pulls this line low, all ROMs on the Apple board will be disabled.

The ROMs are similar to type 2316 and 2716 programmable ROMs. However, the chip selects on most of these PROMs are of a different polarity, and they cannot be plugged directly into the Apple board.



A7	1 ○	24	+5v
A6	2	23	A8
A5	3	22	A9
A4	4	21	<u>CS3</u>
A3	5	20	<u>CS1</u>
A2	6	19	A10
A1	7	18	CS2
A0	8	17	D7
D0	9	16	D6
D1	10	15	D5
D2	11	14	D4
Gnd	12	13	D3

Figure 13. 9316B ROM Pinout.

RAM MEMORY

The Apple uses 4K and 16K dynamic RAMs for its main RAM storage. This RAM memory is used by both the microprocessor and the video display circuitry. The microprocessor and the video display interleave their use of RAM: the microprocessor reads from or writes to RAM only during Φ_0 , and the video display refreshes its screen from RAM memory during Φ_1 .

The three 74LS153s at E11, E12, and E13, the 74LS283 at E14, and half of the 74LS257 at C12 make up the address multiplexer for the RAM memory. They take the addresses generated by the microprocessor and the video generator and multiplex them onto six RAM address lines. The other RAM addressing signals, RAS and CAS, and the signal which is address line 6 for 16K RAMs and CS for 4K RAMs, are generated by the RAM select circuit. This circuit is made up of two 74LS139s at E2 and F2, half of a 74LS153 at location C1, one and a half 74LS257s at C12 and J1, and the three Memory Configuration blocks at D1, E1, and F1. This circuit routes signals to each row of RAM, depending upon what type of RAM (4K or 16K) is in that row.

The dynamic RAMs are refreshed automatically during Φ_1 by the video generator circuitry. Since the video screen is always displaying at least a 1K range of memory, it needs to cycle through every location in that 1K range sixty times a second. It so happens that this action automatically refreshes every bit in all 48K bytes of RAM. This, in conjunction with the interleaving of the video and microprocessor access cycles, lets the video display, the microprocessor, and the RAM refresh run at full speed, without interfering with each other.

The data inputs to the RAMs are drawn directly off of the system's data bus. The data outputs of the RAMs are latched by two 74LS174s at board locations B5 and B8, and are multiplexed with the seven bits of data from the Apple's keyboard. These latched RAM outputs are fed directly to the video generator's character, color, and dot generators, and also back onto the system data bus by two 74LS257s at board locations B6 and B7.

-5v	<i>I</i>	○	16	Gnd
Data In	2		15	CAS
R/W	3		14	Data Out
<u>RAS</u>	4		13	<u>CS</u>
A5	5		12	A2
A4	6		11	A1
A3	7		10	A0
+12v	8		9	+5v

4096 4K RAM
Pinout

-5v	<i>I</i>	○	16	Gnd
Data In	2		15	CAS
R/W	3		14	Data Out
<u>RAS</u>	4		13	<u>CS</u>
A5	5		12	A2
A4	6		11	A1
A3	7		10	A0
+12v	8		9	+5v

4116 16K RAM
Pinout

Figure 14. RAM Pinouts

THE VIDEO GENERATOR

There are 192 scan lines on the video screen, grouped in 24 lines of eight scan lines each. Each scan line displays some or all of the contents of forty bytes of memory.

The video generation circuitry derives its synchronization and timing signals from a chain of 74LS161 counters at board locations D11 through D14. These counters generate fifteen synchronization signals:

H0	H1	H2	H3	H4	H5
V0	V1	V2	V3	V4	
VA	VB	VC			

The "H" family of signals is the horizontal byte position on the screen, from **000000** to binary **100111** (decimal 39). The signals V0 through V4 are the vertical line position on the screen, from binary **00000** to binary **10111** (decimal 23). The VA, VB, and VC signals are the vertical scan line position within the vertical screen line, from binary **000** to 111 (decimal 7).

These signals are sent to the RAM address multiplexer, which turns them into the address of a single RAM location, dependent upon the setting of the video display mode soft switches (see below). The RAM multiplexer then sends this address to the array of RAM memory during Φ_1 . The latches which hold the RAM data sent by the RAM array reroute it to the video generation circuit. The 74LS283 at location rearranges the memory addresses so that the memory mapping on the screen is scrambled.

If the current area on the screen is to be a text character, then the video generator will route the lower six bits of the data to a type 2513 character generator at location A5. The seven rows in each character are scanned by the VA, VB, and VC signals, and the output of the character generator is serialized into a stream of dots by a 74166 at location A3. This bit stream is routed to an exclusive-OR gate, where it is inverted if the high bit of the data byte is off and either the sixth bit is low or the 555 timer at location B3 is high. This produces inverse and flashing characters. The text bit stream is then sent to the video selector/multiplexer (below).

If the Apple's video screen is in a graphics mode, then the data from RAM is sent to two 74LS194 shift registers at board locations B4 and B9. Here each nybble is turned into a serial data stream. These two data streams are also sent to the video selector/multiplexer.

The 74LS257 multiplexer at board position A8 selects between Color and High-Resolution graphics displays. The serialized Hi-res dot stream is delayed one-half clock cycle by the 74LS74 at location A11 if the high bit of the byte is set. This produces the alternate color set in High-Resolution graphics mode.

The video selector/multiplexer mixes the two data streams from the above sources according to the setting of the video screen soft switches. The 74LS194 at location A10 and the 74LS151 at A9 select one of the serial bit streams for text, color graphics, or high-resolution graphics depending upon the screen mode. The final serial output is mixed with the composite synchronization signal and the color burst signal generated by the video sync circuits, and sent to the video output connectors.

The video display soft switches, which control the video modes, are decoded as part of the Apple's on-board I/O functions. Logic gates in board locations B12, B13, B11, A12, and A11 are used to control the various video modes.

The color burst signal is created by logic gates at B12, B13, and C13 and is conditioned by R5, coil L1, C2, and trimmer capacitor C3. This trimmer capacitor can be tuned to vary the tint of colors produced by the video display. Transistor Q6 and its companion resistor R27 disable the color burst signal when the Apple is displaying text.

VIDEO OUTPUT JACKS

The video signal generated by the aforementioned circuitry is an NTSC compatible, similar to an EIA standard, positive composite video signal which can be fed to any standard closed-circuit or studio video monitor. This signal is available in three places on the Apple board:

RCA Jack. On the back of the Apple board, near the right edge, is a standard RCA phono jack. The sleeve of this jack is connected to the Apple's common ground and the tip is connected to the video output signal through a 200 Ohm potentiometer. This potentiometer can adjust the voltage on this connector from 0 to 1 volt peak.

Auxiliary Video Connector. On the right side of the Apple board near the back is a Molex KK100 series connector with four square pins, .25" tall, on .10" centers. This connector supplies the composite video output and two power supply voltages. This connector is illustrated in figure 15.

Table 28: Auxiliary Video Output Connector Signal Descriptions		
Pin	Name	Description
1	GROUND	System common ground; 0 volts.
2	VIDEO	NTSC compatible positive composite video. Black level is about .75 volt, white level about 2.0 volt, sync tip level is 0 volts. - Output level is not adjustable. This is not protected against short circuits.
3	+12v	+12 volt power supply.
4	-5v	-5 volt line from power supply.

Auxiliary Video Pin. This single metal wire-wrap pin below the Auxiliary Video Output Connector supplies the same video signal available on that connector. It is meant to be a connection point for Eurapple PAL/SECAM encoder boards.

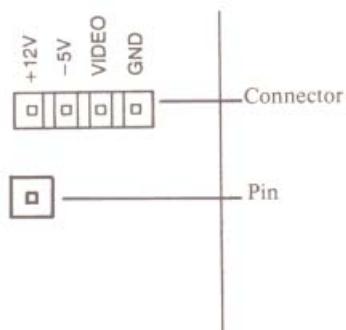


Figure 15. Auxiliary Video Output Connector and Pin.

BUILT-IN I/O

The Apple's built-in I/O functions are mapped into 128 memory locations beginning at \$C000. On the Apple board, a 74LS138 at location F13 called the I/O selector decodes these 128 special addresses and enables the various functions.

The 74LS138 is enabled by another '138 at location H12 whenever the Apple's address bus contains an address between \$C000 and \$C0FF. The I/O selector divides this 256-byte range into eight sixteen-byte ranges, ignoring the range \$C080 through \$C0FF. Each output line of the '138 becomes active (low) when its associated 16-byte range is being referenced.

The "0" line from the I/O selector gates the data from the keyboard connector into the RAM data multiplexer.

The "1" line from the I/O selector resets the 74LS74 flip-flop at B10, which is the keyboard flag.

The "2" line toggles one half of a 74LS74 at location K13. The output of this flip-flop is connected through a resistor network to the tip of the cassette output jack.

The "3" line toggles the other half of the 74LS74 at K13. The output of this flip-flop is connected through a capacitor and Darlington amplifier circuit to the Apple's speaker connector on the right edge of the board under the keyboard.

The "4" line is connected directly to pin 5 of the Game I/O connector. This pin is the utility C040 STROBE.

The "5" line is used to enable the 74LS259 at location F14. This IC contains the soft switches for the video display and the Game I/O connector annunciator outputs. The switches are selected

by the address lines 1 through 3 and the setting of each switch is controlled by address line \emptyset .

The "6" line is used to enable a 74LS251 eight-bit multiplexer at location H14. This multiplexer, when enabled, connects one of its eight input lines to the high order bit (bit 7) of the three-state system data bus. The bottom three address lines control which of the eight inputs the multiplexer chooses. Four of the mux's inputs come from a 553 quad timer at location H13. The inputs to this timer are the game controller pins on the Game I/O connector. Three other inputs to the multiplexer come from the single-bit (pushbutton) inputs on the Game I/O connector. The last multiplexer input comes from a 741 operational amplifier at location K13. The input to this op amp comes from the cassette input jack.

The "7" line from the I/O selector resets all four timers in the 553 quad timer at location H13. The four inputs to this timer come from an RC network made up of four $0.022\mu F$ capacitors, four 100 Ohm resistors, and the variable resistors in the game controllers attached to the Game I/O connector. The total resistance in each of the four timing circuits determines the timing characteristics of that circuit.

"USER 1" JUMPER

There is an unlabeled pair of solder pads on the Apple board, to the left of slot \emptyset , called the "User 1" jumper. This jumper is illustrated in Photo 8. If you connect a wire between these two pads, then the USER 1 line on each peripheral connectors becomes active. If any peripheral card pulls this line low, *all* internal I/O decoding is disabled. The I/O SELECT and the DEVICE SELECT lines all go high and will remain high while USER 1 is low, regardless of the address on the address bus.

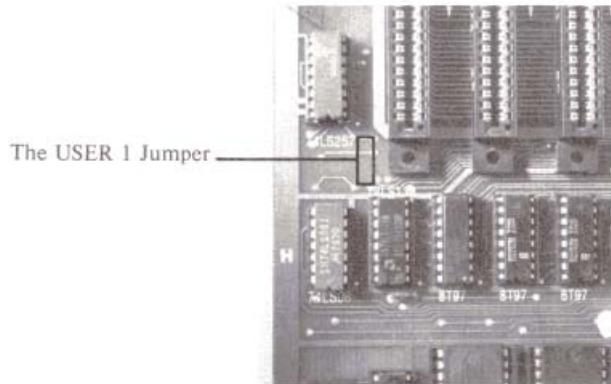


Photo 8. The USER 1 Jumper.

THE GAME I/O CONNECTOR

+5v	1 ○	16	NC
PB0	2	15	AN0
PB1	3	14	AN1
PB2	4	13	AN2
C040 STROBE	5	12	AN3
GC0	6	11	GC3
GC2	7	10	GC1
Gnd	8	9	NC

Figure 16.
Game I/O Connector Pinouts

Table 29: Game I/O Connector Signal Descriptions		
Pin:	Name:	Description:
1	+5v	+5 volt power supply. Total current drain on this pin must be less than 100mA.
2-4	PB0-PB2	Single-bit (Pushbutton) inputs. These are standard 74LS series TTL inputs.
5	C040 STROBE	A general-purpose strobe. This line, normally high, goes low during Φ_0 of a read or write cycle to any address from \$C040 through \$C04F. This is a standard 74LS TTL output.
6,7,10,11	GC0-GC3	Game controller inputs. These should each be connected through a 150K Ohm variable resistor to +5v.
8	Gnd	System electrical ground.
12-15	AN0-AN3	Annunciator outputs. These are standard 74LS series TTL outputs and must be buffered if used to drive other than TTL inputs.
9,16	NC	No internal connection.

THE KEYBOARD

The Apple's built-in keyboard is built around a MM5740 monolithic keyboard decoder ROM. The inputs to this ROM, on pins 4 through 12 and 22 through 31, are connected to the matrix of keyswitches on the keyboard. The outputs of this ROM are buffered by a 7404 and are connected to the Apple's Keyboard Connector (see below).

The keyboard decoder rapidly scans through the array of keys on the keyboard, looking for one which is pressed. This scanning action is controlled by the free-running oscillator made up of three sections of a 7400 at keyboard location U4. The speed of this oscillation is controlled by C6, R6, and R7 on the keyboard's printed-circuit board.

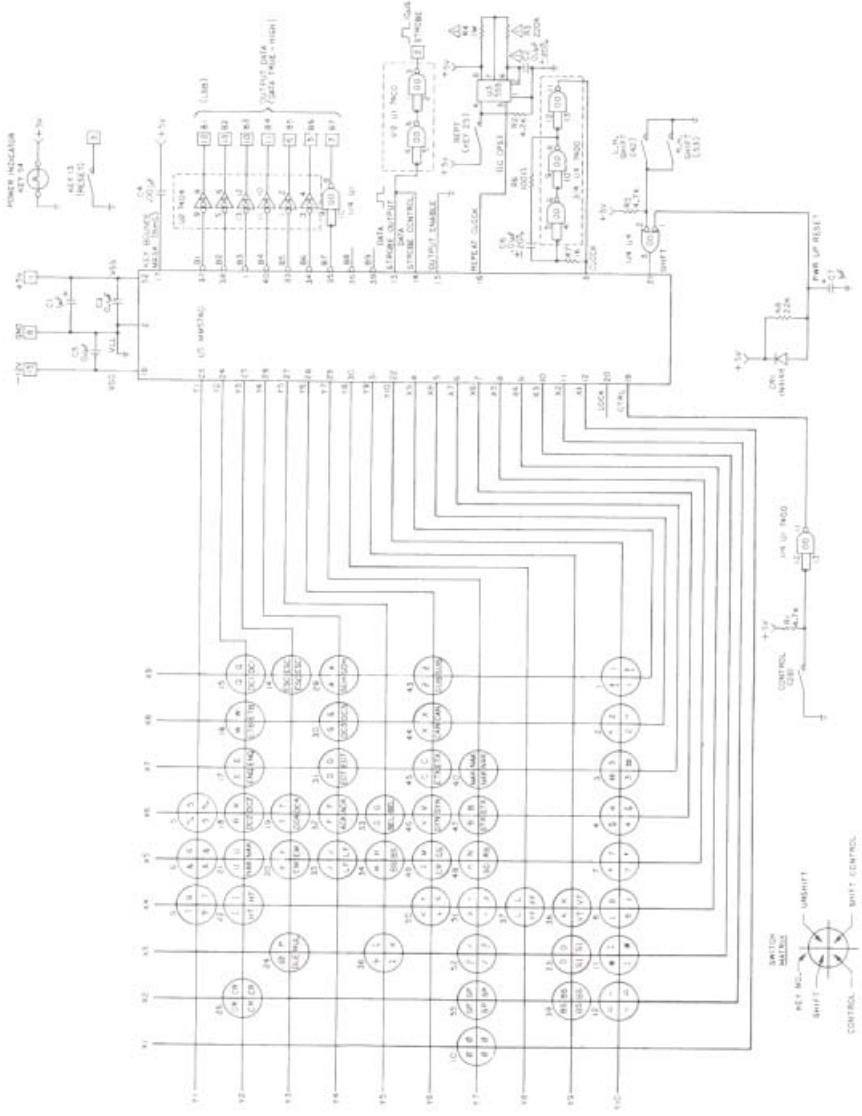


Figure 17. Schematic of the Apple Keyboard

The [REPT] key on the keyboard is connected to a 555 timer circuit at board location U3 on the keyboard. This chip and the capacitor and three resistors around it generate the 10Hz "REPeT" signal. If the 220K Ohm resistor R3 is replaced with a resistor of a lower value, then the [REPT] key will repeat characters at a faster rate.

See Figure 17 for a schematic diagram of the Apple Keyboard.

KEYBOARD CONNECTOR

The data from the Apple's keyboard goes directly to the RAM data multiplexers and latches, the two 74LS257s at locations B6 and B7. The STROBE line on the keyboard connector sets a 74LS74 flip-flop at location B10. When the I/O selector activates its "0" line, the data which is on the seven inputs on the keyboard connector, and the state of the strobe flip-flop, are multiplexed onto the Apple's data bus.

Table 30: Keyboard Connector Signal Descriptions

Pin:	Name:	Description:
1	+5v	+5 volt power supply. Total current drain on this pin must be less than 120mA.
2	STROBE	Strobe output from keyboard. This line should be given a pulse at least $10\mu s$ long each time a key is pressed on the keyboard. The strobe can be of either polarity.
3	RESET	Microprocessor's RESET line. Normally high, this line should be pulled low when the [RESET] button is pressed.
4,9,16	NC	No connection.
5-7, 10-13	Data	Seven bit ASCII keyboard data input.
8	Gnd	System electrical ground.
15	-12v	-12 volt power supply. Keyboard should draw less than 50mA.



+5v	1	○	16	NC
STROBE	2		15	-12v
RESET	3		14	NC
NC	4		13	Data 1
Data 5	5		12	Data Ø
Data 4	6		11	Data 3
Data 6	7		10	Data 2
Gnd	8		9	NC

Figure 18.
Keyboard Connector Pinouts

CASSETTE INTERFACE JACKS

The two female miniature phone jacks on the back of the Apple II board can connect your Apple to a normal home cassette tape recorder.

Cassette Input Jack: This jack is designed to be connected to the "Earphone" or "Monitor" output jacks on most tape recorders. The input voltage should be 1 volt peak-to-peak (nominal). The input impedance is 12K Ohms.

Cassette Output Jack: This jack is designed to be connected to the "Microphone" input on most tape recorders. The output voltage is 25mv into a 100 Ohm impedance load.

POWER CONNECTOR

This connector mates with the cable from the Apple Power Supply. This is an AMP #9-35028-1 six-pin male connector.

Table 31: Power Connector Pin Descriptions

Pin:	Name:	Description:
1,2	Ground	Common electrical ground for Apple board.
3	+5v	+5.0 volts from power supply. An Apple with 48K of RAM and no peripherals draws ~1.5 amp from this supply.
4	+12v	+12.0 volts from power supply. An Apple with 48K of RAM and no peripherals draws ~400ma from this supply.
5	-12v	-12.0 volts from power supply. An Apple with 48K of RAM and no peripherals draws ~12.5ma from this supply.
6	-5v	-5.0 volts from power supply. An Apple with 48K of RAM and no peripherals draws ~0.0ma from this supply.

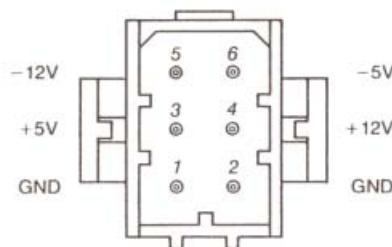


Figure 19. Power Connector

SPEAKER

The Apple's internal speaker is driven by half of a 74LS74 flip-flop through a Darlington amplifier circuit. The speaker connector is a Molex KK100 series connector, with two square pins, .25" tall, on .10" centers.

Table 32: Speaker Connector Signal Descriptions

Pin:	Name:	Description:
1	SPKR	Speaker signal. This line will deliver about .5 watt into an 8 Ohm load.
2	+5V	+5 volt power supply.



Figure 20. Speaker Connector

PERIPHERAL CONNECTORS

The eight peripheral connectors along the back edge of the Apple's board are Winchester #2HW25C0-111 50-pin PC card edge connectors with pins on .10" centers. The pinout for these connectors is given in Figure 21, and the signal descriptions are given on the following pages.

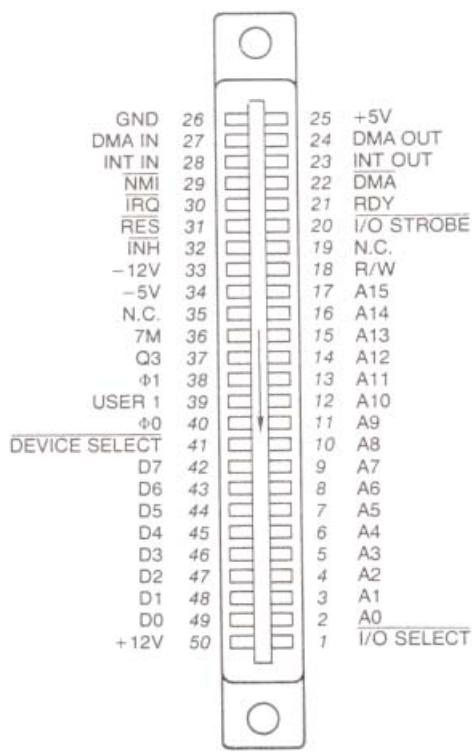


Figure 21. Peripheral Connector Pinout

Table 33: Peripheral Connector Signal Description

Pin:	Name:	Description:
1	I/O SELECT	This line, normally high, will become low when the microprocessor references page \$C _n , where <i>n</i> is the individual slot number. This signal becomes active during Φ_0 and will drive 10 LSTTL loads*. This signal is not present on peripheral connector 0.
2-17	A0-A15	The buffered address bus. The address on these lines becomes valid during Φ_1 and remains valid through Φ_0 . These lines will each drive 5 LSTTL loads*.
18	R/W	Buffered Read/Write signal. This becomes valid at the same time the address bus does, and goes high during a read cycle and low during a write. This line can drive up to 2 LSTTL loads*.
19	SYNC	On peripheral connector 7 <i>only</i> , this pin is connected to the video timing generator's SYNC signal.
20	I/O STROBE	This line goes low during Φ_0 when the address bus contains an address between \$C800 and \$CFFF. This line will drive 4 LSTTL loads*.
21	RDY	The 6502's RDY input. Pulling this line low during Φ_1 will halt the microprocessor, with the address bus holding the address of the current location being fetched.
22	DMA	Pulling this line low disables the 6502's address bus and halts the microprocessor. This line is held high by a 3KΩ resistor to +5v.
23	INT OUT	Daisy-chained interrupt output to lower priority devices. This pin is usually connected to pin 28 (INT IN).
24	DMA OUT	Daisy-chained DMA output to lower priority devices. This pin is usually connected to pin 22 (DMA IN).
25	+5v	+5 volt power supply. 500mA current is available for <i>all</i> peripheral cards.
26	GND	System electrical ground.

* Loading limits are for each peripheral card.

Table 33 (cont'd): Peripheral Connector Signal Description

Table 33 (cont'd): Peripheral Connector Signal Description		
Pin:	Name:	Description:
27	DMA IN	Daisy-chained DMA input from higher priority devices. Usually connected to pin 24 (DMA OUT).
26	INT IN	Daisy-chained interrupt input from higher priority devices. Usually connected to pin 23 (INT OUT).
29	NMI	Non-Maskable Interrupt. When this line is pulled low the Apple begins an interrupt cycle and jumps to the interrupt handling routine at location \$3FB.
30	IRQ	Interrupt ReQuest. When this line is pulled low the Apple begins an interrupt cycle only if the 6502's I (Interrupt disable) flag is not set. If so, the 6502 will jump to the interrupt handling subroutine whose address is stored in locations \$3FE and \$3FF.
31	RES	When this line is pulled low the microprocessor begins a RESET cycle (see page 36).
32	INH	When this line is pulled low, all ROMs on the Apple board are disabled. This line is held high by a $3K\Omega$ resistor to +5v.
33	-12v	-12 volt power supply. Maximum current is 200mA for all peripheral boards.
34	-5v	-5 volt power supply. Maximum current is 200mA for all peripheral boards.
35	COLOR REF	On peripheral connector 7 <i>only</i> , this pin is connected to the 3.5MHz COLOR REference signal of the video generator.
36	7M	7MHz clock. This line will drive 2 LSTTL loads*.
37	Q3	2MHz asymmetrical clock. This line will drive 2 LSTTL loads*.
38	Φ_1	Microprocessor's phase one clock. This line will drive 2 LSTTL loads*.
39	USER 1	This line, when pulled low, disables <i>all</i> internal I/O address decoding**.

* Loading limits are for each peripheral card.
** See page 99.

Table 33 (cont'd): Peripheral Connector Signal Description

Pin:	Name:	Description:
40	$\Phi\emptyset$	Microprocessor's phase zero clock. This line will drive 2 LSTTL loads*.
41	DEVICE SELECT	This line becomes active (low) on each peripheral connector when the address bus is holding an address between $\$C0n\emptyset$ and $\$C0nF$, where n is the slot number plus \$8. This line will drive 10 LSTTL loads*.
42-49	D \emptyset -D7	Buffered bidirectional data bus. The data on this line becomes valid 300nS into $\Phi\emptyset$ on a write cycle, and should be stable no less than 100ns before the end of $\Phi\emptyset$ on a read cycle. Each data line can drive one LSTTL load.
50	+12v	+12 volt power supply. This can supply up to 250mA total for all peripheral cards.

* Loading limits are for each peripheral card.

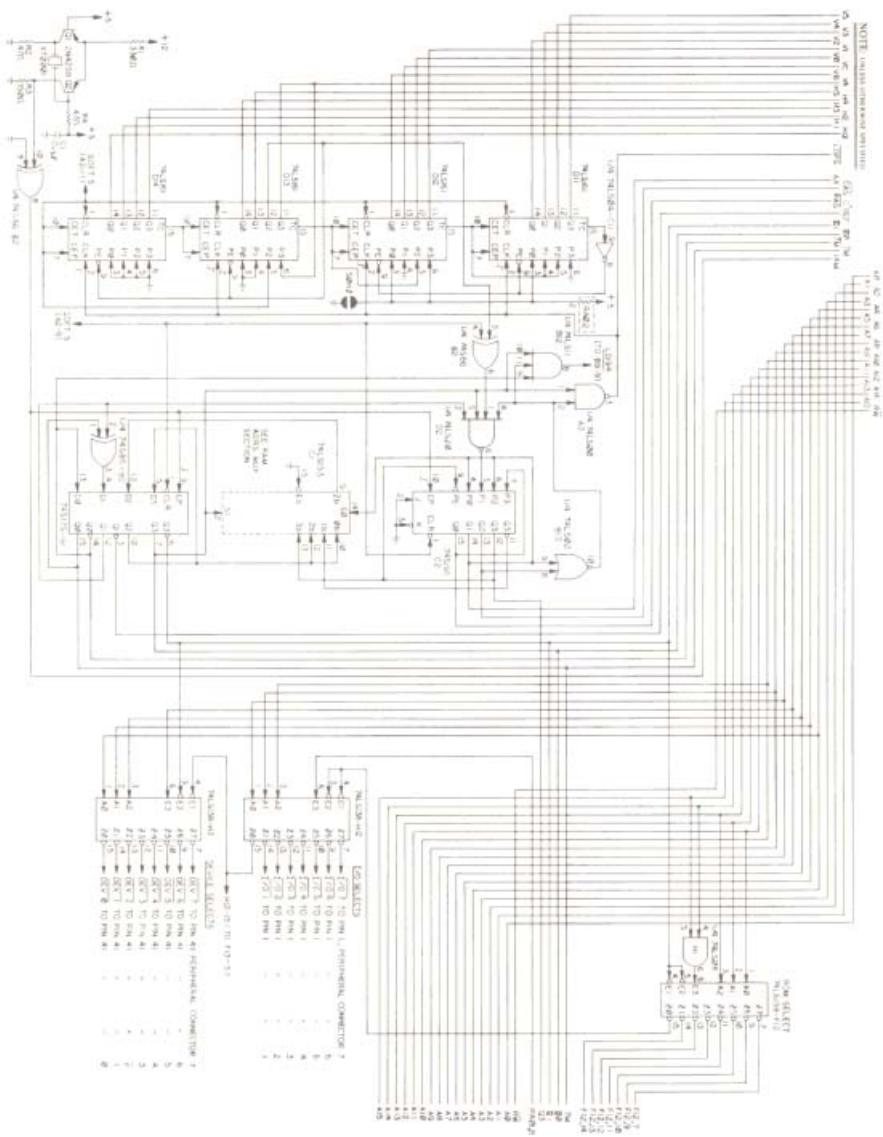


Figure 22-1. Schematic Diagram of the Apple II

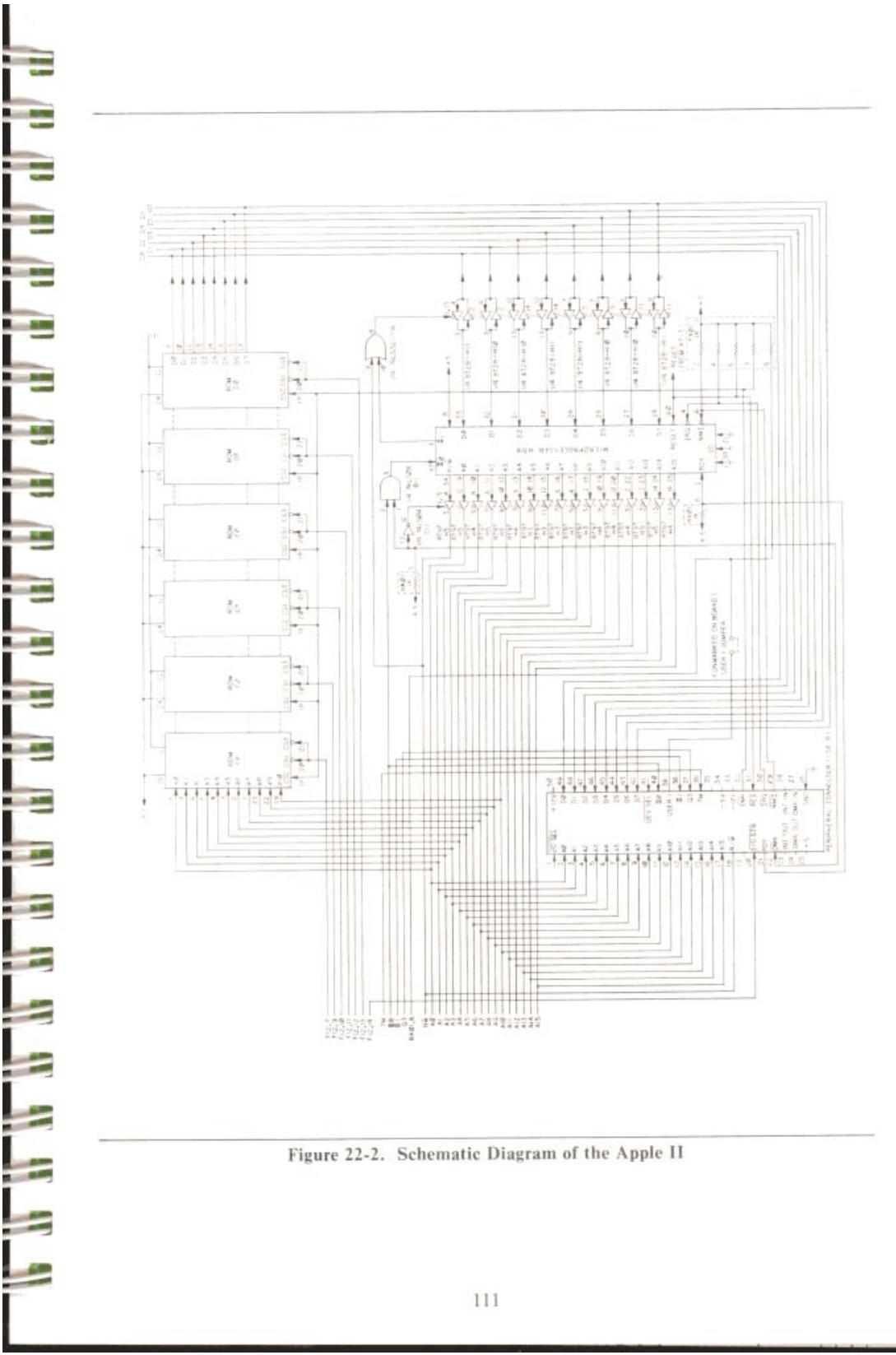


Figure 22-2. Schematic Diagram of the Apple II

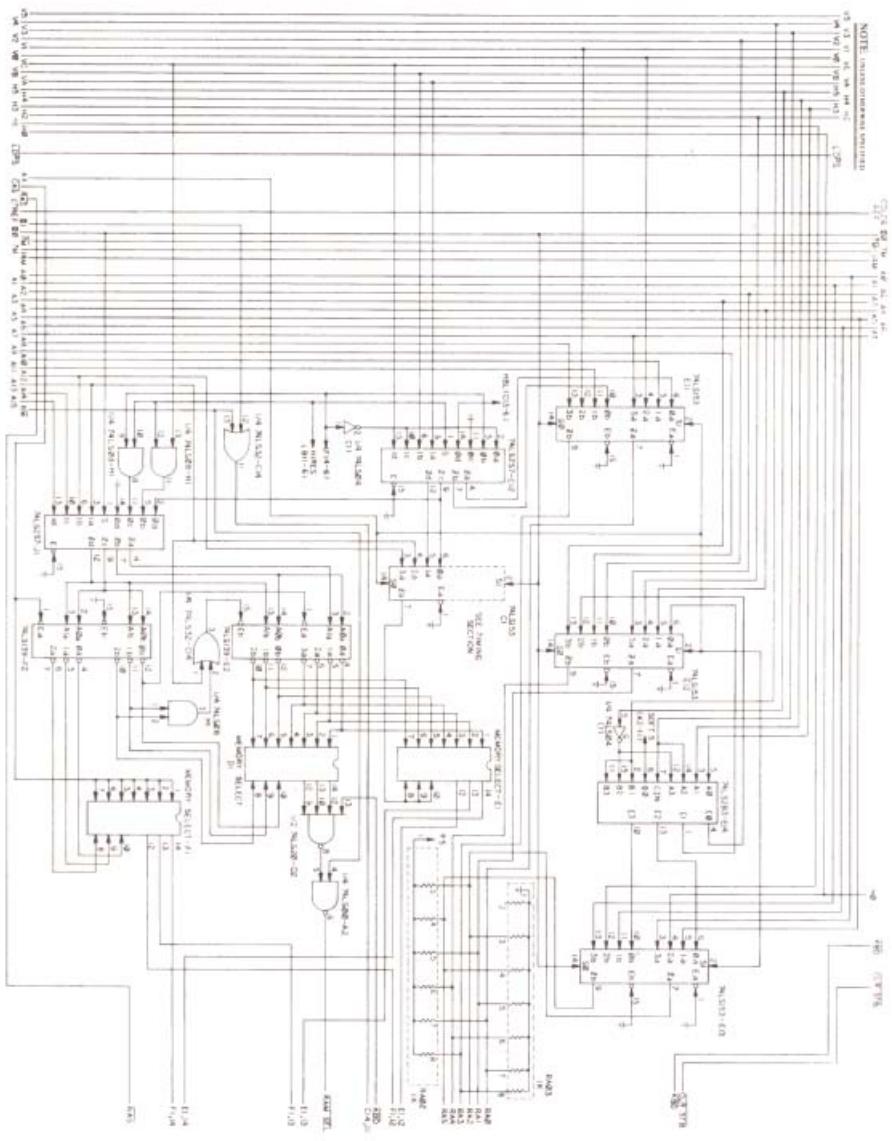


Figure 22-3. Schematic Diagram of the Apple II

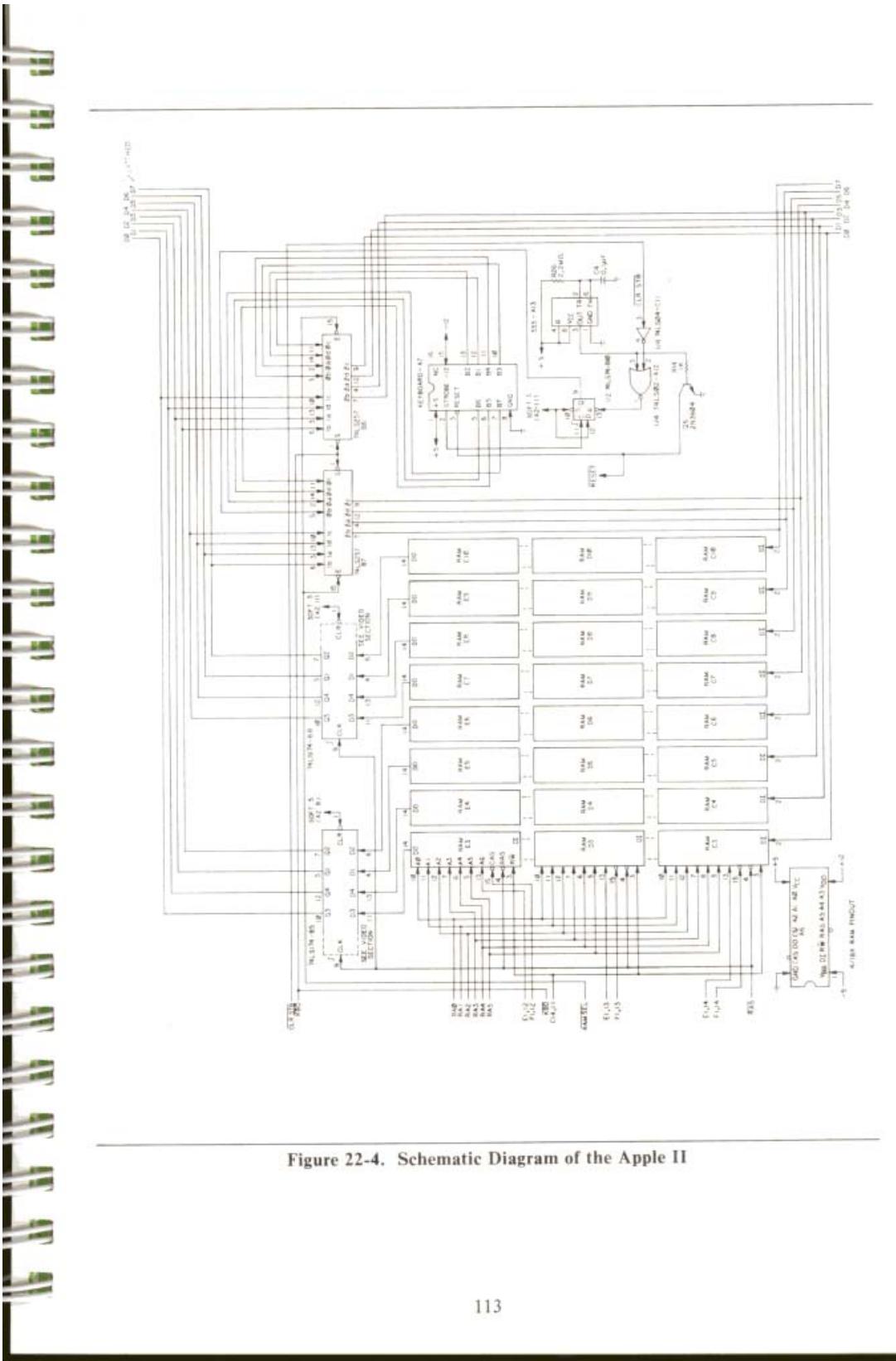


Figure 22-4. Schematic Diagram of the Apple II

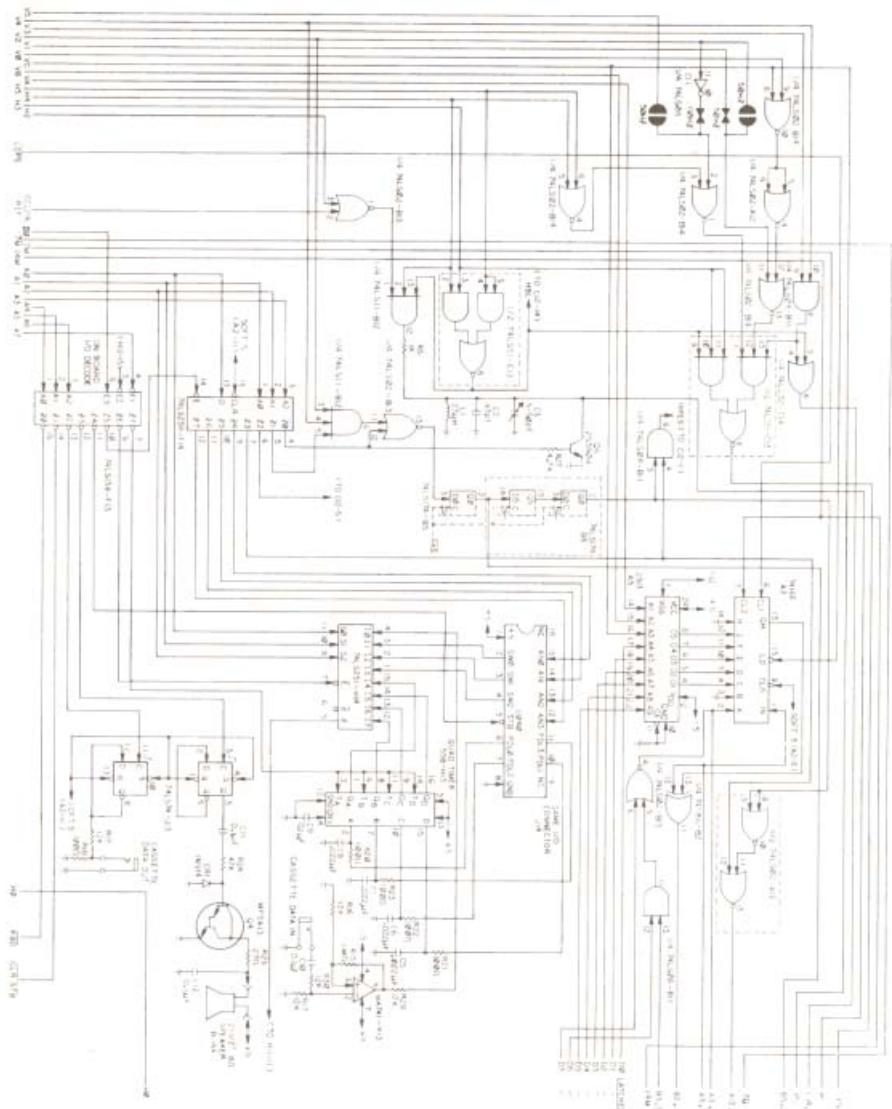


Figure 22-5. Schematic Diagram of the Apple II

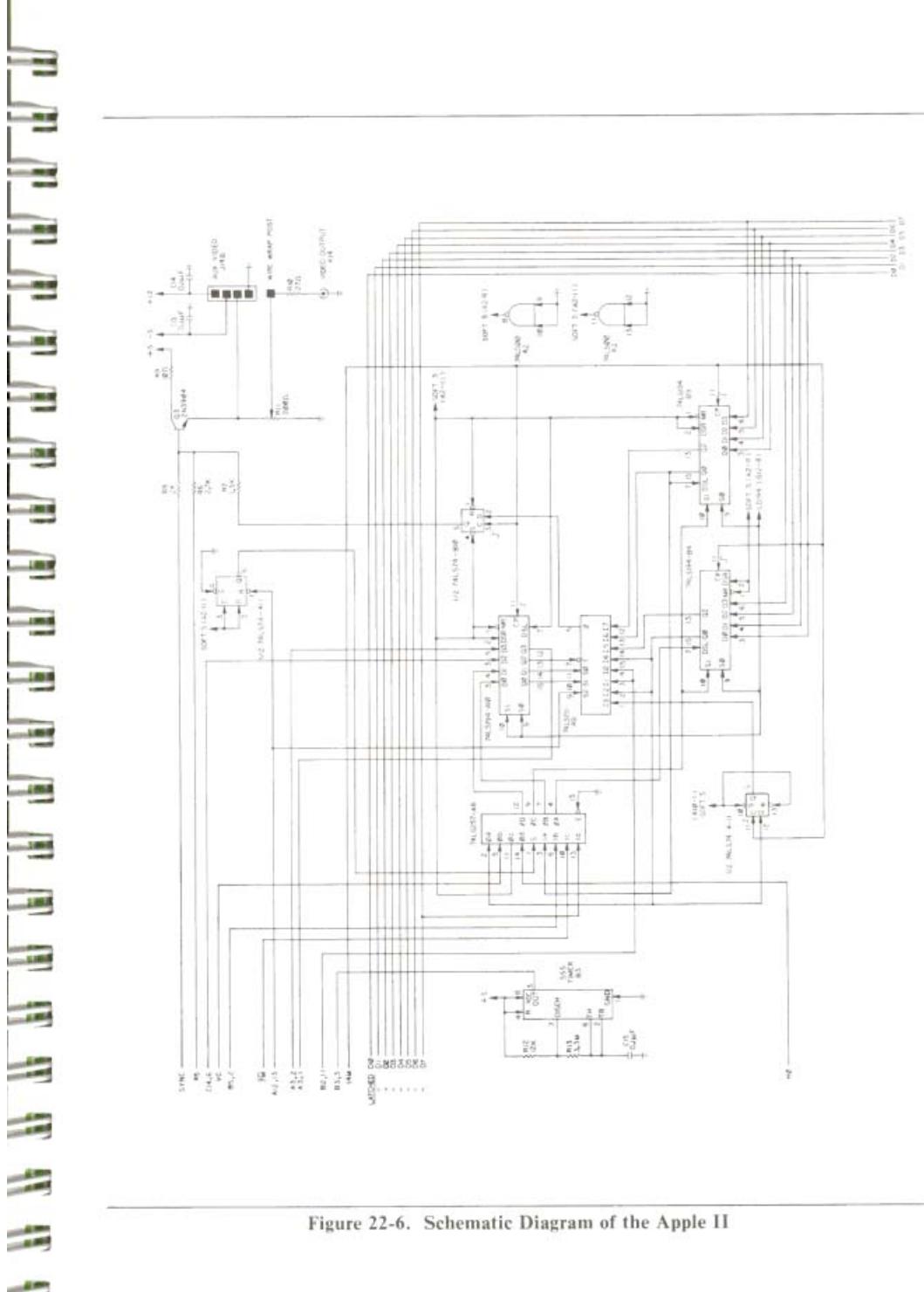


Figure 22-6. Schematic Diagram of the Apple II

III. III.



APPENDIX A

THE 6502 INSTRUCTION SET

6502 MICROPROCESSOR INSTRUCTIONS

ADC	Add Memory to Accumulator with Carry	LDA	Load Accumulator with Memory
AND	"AND" Memory with Accumulator	LDX	Load Index X with Memory
ASL	Shift Left One Bit (Memory or Accumulator)	LDY	Load Index Y with Memory
BCC	Branch on Carry Clear	LSR	Shift Right one Bit (Memory or Accumulator)
BCS	Branch on Carry Set	NOP	No Operation
BEQ	Branch on Result Zero	ORA	"OR" Memory with Accumulator
BIT	Test Bits in Memory with Accumulator	PHA	Push Accumulator on Stack
BMI	Branch on Result Minus	PHP	Push Processor Status on Stack
BNE	Branch on Result not Zero	PLA	Pull Accumulator from Stack
BPL	Branch on Result Plus	PLP	Pull Processor Status from Stack
BRK	Force Break	ROL	Rotate One Bit Left (Memory or Accumulator)
BVC	Branch on Overflow Clear	ROR	Rotate One Bit Right (Memory or Accumulator)
BVS	Branch on Overflow Set	RTI	Return from Interrupt
CLC	Clear Carry Flag	RTS	Return from Subroutine
CLD	Clear Decimal Mode	SBC	Subtract Memory from Accumulator with Borrow
CLI	Clear Interrupt Disable Bit	SEC	Set Carry Flag
CLV	Clear Overflow Flag	SED	Set Decimal Mode
CMP	Compare Memory and Accumulator	SEI	Set Interrupt Disable Status
CPX	Compare Memory and Index X	STA	Store Accumulator in Memory
CPY	Compare Memory and Index Y	STX	Store Index X in Memory
DEC	Decrement Memory by One	STY	Store Index Y in Memory
DEX	Decrement Index X by One	TAX	Transfer Accumulator to Index X
DEY	Decrement Index Y by One	TAY	Transfer Accumulator to Index Y
EOR	"Exclusive-Or" Memory with Accumulator	TSX	Transfer Stack Pointer to Index X
INC	Increment Memory by One	TXA	Transfer Index X to Accumulator
INX	Increment Index X by One	TXS	Transfer Index X to Stack Pointer
INY	increment Index Y by One	TYA	Transfer Index Y to Accumulator
JMP	Jump to New Location		
JSR	Jump to New Location Saving Return Address.		

THE FOLLOWING NOTATION APPLIES TO THIS SUMMARY:

A	Accumulator
X, Y	Index Registers
M	Memory
C	Borrow
P	Processor Status Register
S	Stack Pointer
✓	Change
-	No Change
+	Add
Λ	Logical AND
-	Subtract
∨	Logical Exclusive Or
↓	Transfer From Stack
↑	Transfer To Stack
→	Transfer To
←	Transfer To
V	Logical OR
PC	Program Counter
PCH	Program Counter High
PCL	Program Counter Low
OPER	Operand
#	Immediate Addressing Mode

FIGURE 1 ASL-SHIFT LEFT ONE BIT OPERATION



FIGURE 2. ROTATE ONE BIT LEFT (MEMORY
OR ACCUMULATOR)

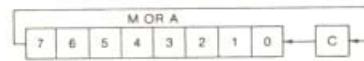


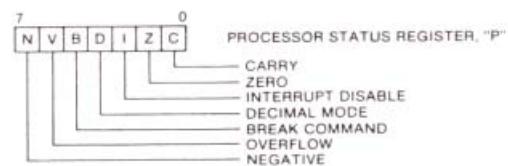
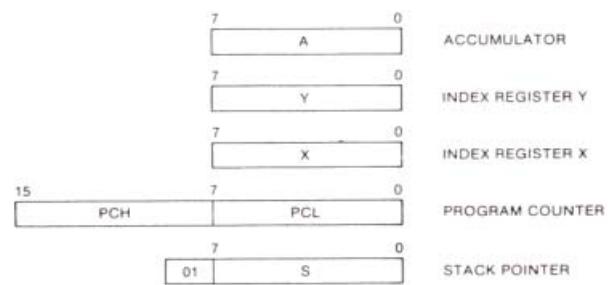
FIGURE 3.



NOTE 1: BIT — TEST BITS

Bit 6 and 7 are transferred to the status register. If the result of A Λ M is zero then Z=1, otherwise Z=0.

PROGRAMMING MODEL



INSTRUCTION CODES

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	$\neg P$ " Status Reg. N Z C I O V
ADC Add memory to accumulator with carry	A - M - C \rightarrow A.C	Immediate Zero Page Zero Page.X Absolute Absolute.X Absolute.Y (Indirect.X) (Indirect).Y	ADC #Oper ADC Oper ADC Oper.X ADC Oper ADC Oper.X ADC Oper.Y ADC (Oper.X) ADC (Oper).Y	69 65 75 60 70 79 61 71	2 2 2 3 3 3 2 2	$\checkmark \checkmark \checkmark \neg \neg \neg$
AND "AND" memory with accumulator	A A M \rightarrow A	Immediate Zero Page Zero Page.X Absolute Absolute.X Absolute.Y (Indirect.X) (Indirect).Y	AND #Oper AND Oper AND Oper.X AND Oper AND Oper.X AND Oper.Y AND (Oper.X) AND (Oper).Y	29 25 35 20 30 39 21 31	2 2 2 3 3 3 2 2	$\checkmark \checkmark \neg \neg \neg \neg$
ASL Shift left one bit (Memory or Accumulator)	(See Figure 1)	Accumulator Zero Page Zero Page.X Absolute Absolute.X	ASL A ASL Oper ASL Oper.X ASL Oper ASL Oper.X	0A 06 16 0E 1E	1 2 2 3 3	$\checkmark \checkmark \checkmark \neg \neg \neg$
BCC Branch on carry clear	Branch on C=0	Relative	BCC Oper	90	2	-----
BCS Branch on carry set	Branch on C=1	Relative	BCS Oper	B0	2	-----
BEQ Branch on result zero	Branch on Z=1	Relative	BEQ Oper	F0	2	-----
BIT Test bits in memory with accumulator	A \wedge M, M ₇ \rightarrow N, M ₆ \rightarrow V	Zero Page Absolute	BIT* Oper BIT* Oper	24 2C	2 3	M ₇ $\checkmark \neg \neg \neg$ M ₆
BMI Branch on result minus	Branch on N=1	Relative	BMI Oper	30	2	-----
BNE Branch on result not zero	Branch on Z=0	Relative	BNE Oper	D0	2	-----
BPL Branch on result plus	Branch on N=0	Relative	BPL Oper	10	2	-----
BRK Force Break	Forced interrupt PC+2 \neq P	Implied	BRK*	00	1	----1----
BVC Branch on overflow clear	Branch on V=0	Relative	BVC Oper	50	2	-----

Note 1: Bits 5 and 7 are transferred to the status register. If the result of A \wedge M is
then M₇ = 1 otherwise M₇ = 0

Note 2: A BRK command cannot be masked by setting:

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX Op Code	No. Bytes	F ¹ Status Reg. N Z C I D V
BVS Branch on overflow set	Branch on V=1	Relative	BVS Oper	70	2	-----
CLC Clear carry flag	0 → C	Implied	CLC	18	1	---0---
CLD Clear decimal mode	0 → D	Implied	CLD	D8	1	-0-----
CLI	0 → I	Implied	CLI	58	1	---0--
CLV Clear overflow flag	0 → V	Implied	CLV	B8	1	0-----
CMP Compare memory and accumulator	A — M	Immediate Zero Page Zero Page, X Absolute Absolute,X Absolute,Y (Indirect,X) (Indirect),Y	CMP #Oper CMP Oper CMP Oper,X CMP Oper CMP Oper,X CMP Oper,Y CMP (Oper,X) CMP (Oper),Y	C9 C5 D5 CD DD D9 C1 D1	2 2 2 3 3 3 2 2	✓✓✓---
CPX Compare memory and index X	X — M	Immediate Zero Page Absolute	CPX #Oper CPX Oper CPX Oper	E0 E4 EC	2 2 3	✓✓✓---
CPY Compare memory and index Y	Y — M	Immediate Zero Page Absolute	CPY #Oper CPY Oper CPY Oper	C0 C4 CC	2 2 3	✓✓✓---
DEC Decrement memory by one	M — 1 → M	Zero Page Zero Page,X Absolute Absolute,X	DEC Oper DEC Oper,X DEC Oper DEC Oper,X	C6 D6 CE DE	2 2 3 3	✓✓-----
DEX Decrement index X by one	X — 1 → X	Implied	DEX	CA	1	✓v-----
DEY Decrement index Y by one	Y — 1 → Y	Implied	DEY	B8	1	✓v-----

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg. N Z C I D V
EOR "Exclusive-Or" memory with accumulator	A V M → A	Immediate Zero Page Zero Page,X Absolute Absolute,X Absolute,Y (Indirect,X) (Indirect)Y	EOR #Oper EOR Oper EOR Oper,X EOR Oper EOR Oper,X EOR Oper,Y EOR (Oper,X) EOR (Oper),Y	49 45 55 40 50 59 41 51	2 2 2 3 3 3 2 2	✓✓-----
INC Increment memory by one	M + 1 → M	Zero Page Zero Page,X Absolute Absolute,X	INC Oper INC Oper,X INC Oper INC Oper,X	E6 F6 EE FE	2 2 3 3	✓✓-----
INX Increment index X by one	X + 1 → X	implied	INX	EB	1	✓✓-----
INY Increment index Y by one	Y + 1 → Y	implied	INY	C8	1	✓✓-----
JMP Jump to new location	(PC+1) → PCL (PC+2) → PCH	Absolute Indirect	JMP Oper JMP (Oper)	4C 6C	3 3	-----
JSR Jump to new location saving return address	PC+2 ↓ (PC+1) → PCL (PC+2) → PCH	Absolute	JSR Oper	20	3	-----
LDA Load accumulator with memory	M → A	Immediate Zero Page Zero Page,X Absolute Absolute,X Absolute,Y (Indirect,X) (Indirect)Y	LDA #Oper LDA Oper LDA Oper,X LDA Oper LDA Oper,X LDA Oper,Y LDA (Oper,X) LDA (Oper),Y	A9 A5 B5 AD BD B9 A1 B1	2 2 2 3 3 3 2 2	✓✓-----
LDX Load index X with memory	M → X	immediate Zero Page Zero Page,Y Absolute Absolute,Y	LDX #Oper LDX Oper LDX Oper,Y LDX Oper LDX Oper,Y	A2 A6 B6 AE BE	2 2 2 3 3	✓✓-----
LDY Load index Y with memory	M → Y	immediate Zero Page Zero Page,X Absolute Absolute,X	LDY #Oper LDY Oper LDY Oper,X LDY Oper LDY Oper,X	A0 A4 B4 AC BC	2 2 2 3 3	✓✓-----

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg. N Z C I D V
LSR Shift right one bit (memory or accumulator)	(See Figure 1)	Accumulator Zero Page Zero Page,X Absolute Absolute,X	LSR A LSR Oper LSR Oper,X LSR Oper LSR Oper,X	4A 46 56 4E 5E	1 2 2 3 3	0✓✓----
NOP No operation	No Operation	Implied	NOP	EA	1	-----
ORA "OR" memory with accumulator	A V M → A	Immediate Zero Page Zero Page,X Absolute Absolute,X Absolute,Y Absolute,Y (Indirect,X) (Indirect),Y	ORA #Oper ORA Oper ORA Oper,X ORA Oper ORA Oper,X ORA Oper,Y ORA (Oper,X) ORA (Oper),Y	09 05 15 00 1D 19 01 11	2 2 2 3 3 3 2 2	✓✓----
PHA Push accumulator on stack	A ↓	Implied	PHA	48	1	-----
PHP Push processor status on stack	P ↓	Implied	PHP	08	1	-----
PLA Pull accumulator from stack	A ↓	Implied	PLA	68	1	✓✓----
PLP Pull processor status from stack	P ↓	Implied	PLP	28	1	From Stack
ROL Rotate one bit left (memory or accumulator)	(See Figure 2)	Accumulator Zero Page Zero Page,X Absolute Absolute,X	ROL A ROL Oper ROL Oper,X ROL Oper ROL Oper,X	2A 26 36 2E 3E	1 2 2 3 3	✓✓✓----
ROR Rotate one bit right (memory or accumulator)	(See Figure 3)	Accumulator Zero Page Zero Page,X Absolute Absolute,X	ROR A ROR Oper ROR Oper,X ROR Oper ROR Oper,X	6A 66 76 6E 7E	1 2 2 3 3	✓✓✓---

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX Op Code	No. Bytes	"P" Status Reg. N Z C I O V
RTI Return from interrupt	P ← PC +	Implied	RTI	40	1	From Stack
RTS Return from subroutine	PC# → PC - 1	Implied	RTS	60	1	-----
SBC Subtract memory from accumulator with borrow	A - M - C → A	Immediate Zero Page Zero Page,X Absolute Absolute,X Absolute,Y (Indirect,X) (Indirect),Y	SBC #Oper SBC Oper SBC Oper,X SBC Oper SBC Oper,X SBC Oper,Y SBC (Oper,X) SBC (Oper),Y	E9 E5 F5 ED FD F9 E1 F1	2 2 2 3 3 3 2 2	✓✓✓----
SEC Set carry flag	1 → C	Implied	SEC	38	1	---1----
SED Set decimal mode	1 → D	Implied	SED	F8	1	-----1--
SEI Set interrupt disable status	1 → I	Implied	SEI	78	1	-----1--
STA Store accumulator in memory	A → M	Zero Page Zero Page,X Absolute Absolute,X Absolute,Y (Indirect,X) (Indirect),Y	STA Oper STA Oper,X STA Oper STA Oper,X STA Oper,Y STA (Oper,X) STA (Oper),Y	85 95 8D 9D 99 81 91	2 2 3 3 3 2 2	-----
STX Store index X in memory	X → M	Zero Page Zero Page,Y Absolute	STX Oper STX Oper,Y STX Oper	86 96 8E	2 2 3	-----
STY Store index Y in memory	Y → M	Zero Page Zero Page,X Absolute	STY Oper STY Oper,X STY Oper	84 94 8C	2 2 3	-----
TAX Transfer accumulator to index X	A → X	Implied	TAX	AA	1	✓✓----
TAY Transfer accumulator to index Y	A → Y	Implied	TAY	AB	1	✓✓----
TSX Transfer stack pointer to index X	S → X	Implied	TSX	BA	1	✓✓----

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg. N Z C I D V
TXA Transfer index X to accumulator	X \leftarrow A	Implied	TXA	8A	1	$\checkmark\checkmark$ -----
TXS Transfer index X to stack pointer	X \leftarrow S	Implied	TXS	9A	1	-----
TYA Transfer index Y to accumulator	Y \leftarrow A	Implied	TYA	9B	1	$\checkmark\checkmark$ -----

HEX OPERATION CODES

00 — BRK	2F — NOP	5E — LSR — Absolute, X
01 — ORA — (Indirect, X)	30 — BMI	5F — NOP
02 — NOP	31 — AND — (Indirect), Y	60 — RTS
03 — NOP	32 — NOP	61 — ADC — (Indirect, X)
04 — NOP	33 — NOP	62 — NOP
05 — ORA — Zero Page	34 — NOP	63 — NOP
06 — ASL — Zero Page	35 — AND — Zero Page, X	64 — NOP
07 — NOP	36 — ROL — Zero Page, X	65 — ADC — Zero Page
08 — PHP	37 — NOP	66 — ROR — Zero Page
09 — ORA — Immediate	38 — SEC	67 — NOP
0A — ASL — Accumulator	39 — AND — Absolute, Y	68 — PLA
0B — NOP	3A — NOP	69 — ADC — Immediate
0C — NOP	3B — NOP	6A — ROR — Accumulator
0D — ORA — Absolute	3C — NOP	6B — NOP
0E — ASL — Absolute	3D — AND — Absolute, X	6C — JMP — Indirect
0F — NOP	3E — ROL — Absolute, X	6D — ADC — Absolute
10 — BPL	3F — NOP	6E — ROR — Absolute
11 — ORA — (Indirect), Y	40 — RTI	6F — NOP
12 — NOP	41 — EOR — (Indirect, X)	70 — BVS
13 — NOP	42 — NOP	71 — ADC — (Indirect), Y
14 — NOP	43 — NOP	72 — NOP
15 — ORA — Zero Page, X	44 — NOP	73 — NOP
16 — ASL — Zero Page, X	45 — EOR — Zero Page	74 — NOP
17 — NOP	46 — LSR — Zero Page	75 — ADC — Zero Page, X
18 — CLC	47 — NOP	76 — ROR — Zero Page, X
19 — ORA — Absolute, Y	48 — PHA	77 — NOP
1A — NOP	49 — EOR — Immediate	78 — SEI
1B — NOP	4A — LSR — Accumulator	79 — ADC — Absolute, Y
1C — NOP	4B — NOP	7A — NOP
1D — ORA — Absolute, X	4C — JMP — Absolute	7B — NOP
1E — ASL — Absolute, X	4D — EOR — Absolute	7C — NOP
1F — NOP	4E — LSR — Absolute	7D — ADC — Absolute, X NOP
20 — JSR	4F — NOP	7E — ROR — Absolute, X NOP
21 — AND — (Indirect, X)	50 — BVC	7F — NOP
22 — NOP	51 — EOR — (Indirect), Y	80 — NOP
23 — NOP	52 — NOP	81 — STA — (Indirect, X)
24 — BIT — Zero Page	53 — NOP	82 — NOP
25 — AND — Zero Page	54 — NOP	83 — NOP
26 — ROL — Zero Page	55 — EOR — Zero Page, X	84 — STY — Zero Page
27 — NOP	56 — LSR — Zero Page, X	85 — STA — Zero Page
28 — PLP	57 — NOP	86 — STX — Zero Page
29 — AND — Immediate	58 — CLI	87 — NOP
2A — ROL — Accumulator	59 — EOR — Absolute, Y	88 — DEY
2B — NOP	5A — NOP	89 — NOP
2C — BIT — Absolute	5B — NOP	8A — TXA
2D — AND — Absolute	5C — NDP	8B — NOP
2E — ROL — Absolute	5D — EOR — Absolute, X	8C — STY — Absolute

8D — STA — Absolute	B4 — LDY — Zero Page, X	DB — NOP
8E — STX — Absolute	B5 — LDA — Zero Page, X	DC — NOP
8F — NOP	B6 — LDX — Zero Page, Y	DD — CMP — Absolute, X
90 — BCC	B7 — NOP	DE — DEC — Absolute, X
91 — STA — Indirect, Y	B8 — CLV	DF — NOP
92 — NOP	B9 — LDA — Absolute, Y	E0 — CPX — Immediate
93 — NOP	BA — TSX	E1 — SBC — Indirect, X
94 — STY — Zero Page, X	BB — NOP	E2 — NOP
95 — STA — Zero Page, X	BC — LDY — Absolute, X	E3 — NOP
96 — STX — Zero Page, Y	BD — LDA — Absolute, X	E4 — CPX — Zero Page
97 — NOP	BE — LDX — Absolute, Y	E5 — SBC — Zero Page
98 — TYA	BF — NOP	E6 — INC — Zero Page
99 — STA — Absolute, Y	C0 — CPY — Immediate	E7 — NOP
9A — TXS	C1 — CMP — Indirect, X	E8 — INX
9B — NOP	C2 — NOP	E9 — SBC — Immediate
9C — NOP	C3 — NOP	EA — NOP
9D — STA — Absolute, X	C4 — CPY — Zero Page	EB — NOP
9E — NOP	C5 — CMP — Zero Page	EC — CPX — Absolute
9F — NOP	C6 — DEC — Zero Page	ED — SBC — Absolute
A0 — LDY — Immediate	C7 — NOP	EE — INC — Absolute
A1 — LDA — Indirect, X	C8 — INY	EF — NOP
A2 — LDX — Immediate	C9 — CMP — Immediate	F0 — BEQ
A3 — NOP	CA — DEX	F1 — SBC — Indirect, Y
A4 — LDY — Zero Page	CB — NOP	F2 — NOP
A5 — LDA — Zero Page	CC — CPY — Absolute	F3 — NOP
A6 — LDX — Zero Page	CD — CMP — Absolute	F4 — NOP
A7 — NOP	CE — DEC — Absolute	F5 — SBC — Zero Page, X
A8 — TAY	CF — NOP	F6 — INC — Zero Page, X
A9 — LDA — Immediate	D0 — BNE	F7 — NOP
AA — TAX	D1 — CMP — Indirect, Y	F8 — SED
AB — NOP	D2 — NOP	F9 — SBC — Absolute, Y
AC — LDY — Absolute	D3 — NOP	FA — NOP
AD — Absolute	D4 — NOP	FB — NOP
AE — LDX — Absolute	D5 — CMP — Zero Page, X	FC — NOP
AF — NOP	D6 — DEC — Zero Page, X	FD — SBC — Absolute, X
B0 — BCS	D7 — NOP	FE — INC — Absolute, X
B1 — LDA — Indirect, Y	D8 — CLD	FF — NOP
B2 — NOP	D9 — CMP — Absolute, Y	DA — NOP
B3 — NOP		



APPENDIX B

SPECIAL LOCATIONS

Table 1: Keyboard Special Locations

Location:		Description:	
Hex	Decimal		
\$C000	49152	-16384	Keyboard Data
\$C010	49168	-16368	Clear Keyboard Strobe

Table 4: Video Display Memory Ranges

Screen	Page	Begins at:		Ends at:	
		Hex	Decimal	Hex	Decimal
Text/Lo-Res	Primary	\$400	1024	\$7FF	2047
	Secondary	\$800	2048	\$BFF	3071
Hi-Res	Primary	\$2000	8192	\$3FFF	16383
	Secondary	\$4000	16384	\$5FFF	24575

Table 5: Screen Soft Switches

Location:		Description:	
Hex	Decimal		
SC050	49232	-16304	Display a GRAPHICS mode.
SC051	49233	-16303	Display TEXT mode.
SC052	49234	-16302	Display all TEXT or GRAPHICS.
SC053	49235	-16301	Mix TEXT and a GRAPHICS mode.
SC054	49236	-16300	Display the Primary page (Page 1).
SC055	49237	-16299	Display the Secondary page (Page 2).
SC056	49238	-16298	Display LO-RES GRAPHICS mode.
SC057	49239	-16297	Display HI-RES GRAPHICS mode.

Table 9: Annunciator Special Locations

Ann.	State	Address:		
		Decimal	Hex	
0	off	49240	-16296	SC058
	on	49241	-16295	SC059
1	off	49242	-16294	SC05A
	on	49243	-16293	SC05B
2	off	49244	-16292	SC05C
	on	49245	-16291	SC05D
3	off	49246	-16290	SC05E
	on	49247	-16289	SC05F

Table 10: Input/Output Special Locations

Function	Address:		Read/Write
	Decimal	Hex	
Speaker	49200	-16336	\$C030
Cassette Out	49184	-16352	\$C020
Cassette In	49256	-16288	\$C060
Annunciators	49240 through 49247	-16296 through -16289	\$C058 through \$C05F
Flag inputs	49249 49250 49251	-16287 -16286 -16285	\$C061 \$C062 \$C063
Analog Inputs	49252 49253 49254 49255	-16284 -16283 -16282 -16281	\$C064 \$C065 \$C066 \$C067
Analog Clear	49264	-16272	\$C070
Utility Strobe	49216	-16320	\$C040
			R/W

Table 11: Text Window Special Locations

Function	Location:	Minimum/Normal/Maximum Value	
	Decimal	Hex	Decimal Hex
Left Edge	32	\$20	0/0/39 \$0/\$0/\$17
Width	33	\$21	0/40/40 \$0/\$28/\$28
Top Edge	34	\$22	0/0/24 \$0/\$0/\$18
Bottom Edge	35	\$23	0/24/24 \$0/\$18/\$18

Table 12: Normal/Inverse Control Values

Value:		Effect:
Decimal	Hex	
255	SFF	COUT will display characters in Normal mode.
63	\$3F	COUT will display characters in Inverse mode.
127	\$7F	COUT will display letters in Flashing mode, all other characters in Inverse mode.

Table 13: Autostart ROM Special Locations

Location:		
Decimal	Hex	Contents:
1010	\$3F2	Soft Entry Vector. These two locations contain the address of the reentry point for whatever language is in use. Normally contains \$E003.
1011	\$3F3	
1012	\$3F4	Power-Up Byte. Normally contains \$45.
64367 (-1169)	\$FB6F	This is the beginning of a machine language subroutine which sets up the power-up location.

Table 14: Page Three Monitor Locations

Address: Decimal	Hex	Use: Monitor ROM	Autostart ROM
1008	\$3F0		Holds the address of the subroutine which handles machine language “BRK” requests (normally \$FA59).
1009	\$3F1	None.	
1010	\$3F2	None.	Soft Entry Vector.
1011	\$3F3		
1012	\$3F4	None.	Power-up byte.
1013	\$3F5		
1014	\$3F6		
1015	\$3F7		
1016	\$3F8		
1017	\$3F9		
1018	\$3FA		
1019	\$3FB		
1020	\$3FC		
1021	\$3FD		
1022	\$3FE		
1023	\$3FF		

Table 22: Built-In I/O Locations

	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
\$C000	Keyboard Data Input															
\$C010	Clear Keyboard Strobe															
\$C020	Cassette Output Toggle															
\$C030	Speaker Toggle															
\$C040	Utility Strobe															
\$C050	gr	tx	nomix	mix	pri	sec	lores	hi-res	an0	an1	an2	an3				
\$C060	cin	pb1	pb2	pb3	gc0	gc1	gc2	gc3					repeat \$C060-\$C067			
\$C070	Game Controller Strobe															

Key to abbreviations:

gr	Set GRAPHICS mode	tx	Set TEXT mode
nomix	Set all text or graphics	mix	Mix text and graphics
pri	Display primary page	sec	Display secondary page
lores	Display Low-Res Graphics	hi-res	Display Hi-Res Graphics
an	Annunciator outputs	pb	Pushbutton inputs
gc	Game Controller inputs	cin	Cassette Input

Table 23: Peripheral Card I/O Locations

Table 24: Peripheral Card PROM Locations

Table 25: I/O Location Base Addresses

Base Address	Slot							
	0	1	2	3	4	5	6	7
\$C080	\$C080	\$C090	\$C0A0	\$C0B0	\$C0C0	\$C0D0	\$C0E0	\$C0F0
\$C081	\$C081	\$C091	\$C0A1	\$C0B1	\$C0C1	\$C0D1	\$C0E1	\$C0F1
\$C082	\$C082	\$C092	\$C0A2	\$C0B2	\$C0C2	\$C0D2	\$C0E2	\$C0F2
\$C083	\$C083	\$C093	\$C0A3	\$C0B3	\$C0C3	\$C0D3	\$C0E3	\$C0F3
\$C084	\$C084	\$C094	\$C0A4	\$C0B4	\$C0C4	\$C0D4	\$C0E4	\$C0F4
\$C085	\$C085	\$C095	\$C0A5	\$C0B5	\$C0C5	\$C0D5	\$C0E5	\$C0F5
\$C086	\$C086	\$C096	\$C0A6	\$C0B6	\$C0C6	\$C0D6	\$C0E6	\$C0F6
\$C087	\$C087	\$C097	\$C0A7	\$C0B7	\$C0C7	\$C0D7	\$C0E7	\$C0F7
\$C088	\$C088	\$C098	\$C0A8	\$C0B8	\$C0C8	\$C0D8	\$C0E8	\$C0F8
\$C089	\$C089	\$C099	\$C0A9	\$C0B9	\$C0C9	\$C0D9	\$C0E9	\$C0F9
\$C08A	\$C08A	\$C09A	\$C0AA	\$C0BA	\$C0CA	\$C0DA	\$C0EA	\$C0FA
\$C08B	\$C08B	\$C09B	\$C0AB	\$C0BB	\$C0CB	\$C0DB	\$C0EB	\$C0FB
\$C08C	\$C08C	\$C09C	\$C0AC	\$C0BC	\$C0CC	\$C0DC	\$C0EC	\$C0FC
\$C08D	\$C08D	\$C09D	\$C0AD	\$C0BD	\$C0CD	\$C0DD	\$C0ED	\$C0FD
\$C08E	\$C08E	\$C09E	\$C0AE	\$C0BE	\$C0CE	\$C0DE	\$C0EE	\$C0FE
\$C08F	\$C08F	\$C09F	\$C0AF	\$C0BF	\$C0CF	\$C0DF	\$C0EF	\$C0FF

Table 26: I/O Scratchpad RAM Addresses

Base Address	Slot Number						
	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



APPENDIX C ROM LISTINGS

136 AUTOSTART ROM LISTING
155 MONITOR ROM LISTING

AUTOSTART ROM LISTING

```
0000:          ****
0000:          3 4
0000:          4 * APPLE II
0000:          5 * MONITOR II
0000:          6 *
0000:          7 * COPYRIGHT 1978 BY
0000:          8 * APPLE COMPUTER, INC.
0000:          9 *
0000:         10 * ALL RIGHTS RESERVED
0000:         11 *
0000:         12 * STEVE WOZNIAK
0000:         13 *
0000:         14 ****
0000:         15 *
0000:         16 * MODIFIED NOV 1978
0000:         17 * BY JOHN A
0000:         18 *
0000:         19 ****
FB00:          20 ORG $FB00
FB00:          21 OBJ $2000
FB00:          22 ****
FB00:          23 LOCO EQU $00
FB00:          24 LOC1 EQU $01
FB00:          25 WNDLFT EQU $20
FB00:          26 WNDWDTH EQU $21
FB00:          27 WNDTOP EQU $22
FB00:          28 WNDBTM EQU $23
FB00:          29 CH EQU $24
FB00:          30 CV EQU $25
FB00:          31 GBASL EQU $26
FB00:          32 GBASH EQU $27
FB00:          33 BASL EQU $28
FB00:          34 BASH EQU $29
FB00:          35 BAS2L EQU $2A
FB00:          36 BAS2H EQU $2B
FB00:          37 H2 EQU $2C
FB00:          38 LMNEM EQU $2C
FB00:          39 V2 EQU $2D
FB00:          40 RMNEM EQU $2D
FB00:          41 MASK EQU $2E
FB00:          42 CHKSUM EQU $2E
FB00:          43 FORMAT EQU $2E
FB00:          44 LASTIN EQU $2F
FB00:          45 LENGTH EQU $2F
FB00:          46 SIGN EQU $2F
FB00:          47 COLOR EQU $30
FB00:          48 MODE EQU $31
FB00:          49 INVFLG EQU $32
FB00:          50 PROMPT EQU $33
FB00:          51 YSAV EQU $34
FB00:          52 YSAV1 EQU $35
FB00:          53 CSWL EQU $36
FB00:          54 CSWH EQU $37
FB00:          55 KSWL EQU $38
FB00:          56 KSWH EQU $39
FB00:          57 PCL EQU $3A
FB00:          58 PCH EQU $3B
FB00:          59 A1L EQU $3C
FB00:          60 A1H EQU $3D
FB00:          61 A2L EQU $3E
FB00:          62 A2H EQU $3F
FB00:          63 A3L EQU $40
FB00:          64 A3H EQU $41
FB00:          65 A4L EQU $42
FB00:          66 A4H EQU $43
FB00:          67 A5L EQU $44
FB00:          68 A5H EQU $45
```

```

FB00:      69 ACC      EQU $45      ; NOTE OVERLAP WITH A5H!
FB00:      70 XREG     EQU $46
FB00:      71 YREG     EQU $47
FB00:      72 STATUS    EQU $48
FB00:      73 SPNT      EQU $49
FB00:      74 RNDL      EQU $4E
FB00:      75 RNDH      EQU $4F
FB00:      76 PICK      EQU $95
FB00:      77 IN        EQU $0200
FB00:      78 BRKV      EQU $3F0   ; NEW VECTOR FOR BRK
FB00:      79 SOFTEV    EQU $3F2   ; VECTOR FOR WARM START
FB00:      80 PWREDUP   EQU $3F4   ; THIS MUST = EOR #$A5 OF SOFTEV+1
FB00:      81 AMPERV    EQU $3F5   ; APPLESOFT & EXIT VECTOR
FB00:      82 USRADR    EQU $03FB
FB00:      83 NMI       EQU $03FB
FB00:      84 IRQLOC    EQU $3FE
FB00:      85 LINE1     EQU $400
FB00:      86 MSLOT     EQU $07FB
FB00:      87 IODR      EQU $C000
FB00:      88 KBD       EQU $C000
FB00:      89 KBDSTRB   EQU $C010
FB00:      90 TAPEOUT   EQU $C020
FB00:      91 SPKR      EQU $C030
FB00:      92 TXTCLR    EQU $C050
FB00:      93 TXTSET    EQU $C051
FB00:      94 MIXCLR    EQU $C052
FB00:      95 MIXSET    EQU $C053
FB00:      96 LOWSCR    EQU $C054
FB00:      97 HISCR     EQU $C055
FB00:      98 LORES     EQU $C056
FB00:      99 HIRES     EQU $C057
FB00:     100 SETANO    EQU $C058
FB00:     101 CLRANO    EQU $C059
FB00:     102 SETAN1    EQU $C05A
FB00:     103 CLRAN1   EQU $C05B
FB00:     104 SETAN2    EQU $C05C
FB00:     105 CLRAN2   EQU $C05D
FB00:     106 SETAN3    EQU $C05E
FB00:     107 CLRAN3   EQU $C05F
FB00:     108 TAPEIN    EQU $C060
FB00:     109 PADDLO    EQU $C064
FB00:     110 PTRIG     EQU $C070
FB00:     111 CLRRDM    EQU $CFFF
FB00:     112 BASIC     EQU $E000
FB00:     113 BASIC2    EQU $E003
FB00:     114 PAGE      EQU $E004
FB00:     4A          115 PLOT    LSR A
FB01:     08          116          PHP
FB02: 20 47 FB     117          JSR QBASCALC
FB05: 2B          118          PLP
FB06: A9 0F        119          LDA #$0F
FB08: 90 02        120          BCC RTMASK
FB0A: 69 E0        121          ADC #$E0
FB0C: B5 2E        122          RTMASK STA MASK
FB0E: B1 26        123          PLOT1 LDA (GBASL),Y
FB10: 45 30        124          EOR COLOR
FB12: 25 2E        125          AND MASK
FB14: 51 26        126          EOR (GBASL),Y
FB16: 91 26        127          STA (GBASL),Y
FB18: 60          128          RTS
FB19: 20 00 FB     129          HLINE JSR PLOT
FB1C: C4 20        130          HLINE1 CPY H2
FB1E: B0 11        131          BCS RTS1
FB20: C8          132          INY
FB21: 20 0E FB     133          JSR PLOT1
FB24: 90 F6        134          BCC HLINE1
FB26: 69 01        135          VLINEZ ADC #$01
FB28: 4B          136          VLINE PHA
FB29: 20 00 FB     137          JSR PLOT
FB2C: 6B          138          PLA
FB2D: C5 2D        139          CMP V2
FB2F: 90 F5        140          BCC VLINEZ
FB31: 60          141          RTS

```

FB32: A0 2F	142 CLRSCR	LDY #\$2F
FB34: D0 02	143 BNE CLRSC2	
FB36: A0 27	144 CLRTOP	LDY #\$27
FB38: B4 2D	145 CLRSC2	STY V2
FB3A: A0 27	146 CLRSC3	LDY #\$27
FB3C: A9 00	147 CLRSC3	LDA #\$00
FB3E: B5 30	148 STA COLOR	
FB40: 20 28 FB	149 JSR VLINE	
FB43: B8	150 DEY	
FB44: 10 F6	151 BPL CLRSC3	
FB46: 60	152 RTS	
FB47:	153 PAGE	
FB47: 48	154 GBASCALC PHA	
FB48: 44	155 LSR A	
FB49: 29 03	156 AND #\$03	
FB4B: 09 04	157 ORA #\$04	
FB4D: B5 27	158 STA GBASH	
FB4F: 68	159 PLA	
FB50: 29 18	160 AND #\$1B	
FB52: 90 02	161 BCC GB CALC	
FB54: 69 7F	162 ADC #\$7F	
FB56: B5 26	163 GBCALC STA GBASL	
FB58: 04	164 ASL A	
FB59: 0A	165 ASL A	
FB5A: 05 26	166 ORA GBASL	
FB5C: B5 26	167 STA GBASL	
FB5E: 60	168 RTS	
FB5F: A5 30	169 LDA COLOR	
FB61: 1B	170 CLC	
FB62: 69 03	171 ADC #\$03	
FB64: 29 0F	172 SETCOL AND #\$0F	
FB66: B5 30	173 STA COLOR	
FB68: 0A	174 ASL A	
FB69: 0A	175 ASL A	
FB6A: 0A	176 ASL A	
FB6B: 0A	177 ASL A	
FB6C: 05 30	178 ORA COLOR	
FB6E: B5 30	179 STA COLOR	
FB70: 60	180 RTS	
FB71: 4A	181 SCR N	LSR A
FB72: 0E	182 PHP	
FB73: 20 47 FB	183 JSR GBASCALC	
FB76: B1 26	184 LDA (GBASL), Y	
FB78: 2B	185 PLP	
FB79: 90 04	186 SCR N2 BCC RTMSKZ	
FB7B: 4A	187 LSR A	
FB7C: 4A	188 LSR A	
FB7D: 4A	189 LSR A	
FB7E: 4A	190 LSR A	
FB7F: 29 0F	191 RTMSKZ AND #\$0F	
FBB1: 60	192 RTS	
FBB2:	193 PAGE	
FBB2: A5 3A	194 INSDS1 LDX PCL	
FBB4: A4 3B	195 LDY PCH	
FBB6: 20 96 FD	196 JSR PRYX2	
FBB9: 20 48 F9	197 JSR PRBLNK	
FBBC: A1 3A	198 INSDS2 LDA (PCL, X)	
FBBE: AB	199 TAY	
FBBF: 4A	200 LSR A	
FB90: 90 09	201 BCC IEVEN	
FB92: 6A	202 ROR A	
FB93: B0 10	203 BCS ERR	
FB95: C9 A2	204 CMP #\$A2	
FB97: F0 0C	205 BEQ ERR	
FB99: 29 87	206 AND #\$B7	
FB9B: 4A	207 IEVEN LSR A	
FB9C: AA	208 TAX	
FB9D: BD 62 F9	209 LDA FMT1, X	
FBA0: 20 79 FB	210 JSR SCR N2	
FBA3: D0 04	211 BNE GETFMT	
FBA5: A0 B0	212 ERR LDY #\$B0	
FBA7: A9 00	213 LDA #\$00	
FBA9: AA	214 GETFMT TAX	

FBA0:	BD A6 F9	215	LDA FMT2, X
FBA0:	B5 2E	216	STA FORMAT
FBA0:	29 03	217	AND #\$03
FBA1:	B5 2F	218	STA LENGTH
FBA3:	98	219	TYA
FBA4:	29 BF	220	AND #\$BF
FBA6:	AA	221	TAX
FBA7:	98	222	TYA
FBBB:	A0 03	223	LDY #\$03
FBBB:	E0 BA	224	CPX #\$BA
FBC0:	FO 0B	225	BEG MNNDX3
FBBE:	4A	226	MNNDX1 LSR A
FBBF:	90 0B	227	BCC MNNDX3
FBC1:	4A	228	LSR A
FBC2:	4A	229	MNNDX2 LSR A
FBC3:	09 20	230	ORA #\$20
FBC5:	BB	231	DEY
FBC6:	D0 FA	232	BNE MNNDX2
FBC8:	CB	233	INY
FBC9:	BB	234	MNNDX3 DEY
FBCA:	D0 F2	235	BNE MNNDX1
FBCC:	60	236	RTS
FBCD:	FF FF FF	237	DFB \$FF, \$FF, \$FF
FBD0:	238		PAGE
FBD0:	20 B2 FB	239	INSTDSP JSR INSDS1
FBD3:	4B	240	PHA
FBD4:	B1 3A	241	PRNTOP LDA (PCL), Y
FBD6:	20 DA FD	242	JSR PRBYTE
FBD9:	A2 01	243	LDX #\$01
FBD9:	20 4A F9	244	PRNTBL JSR PRBL2
FBD8:	C4 2F	245	CPY LENGTH
FBE0:	CB	246	INY
FBE1:	90 F1	247	BCC PRNTOP
FBE3:	A2 03	248	LDX #\$03
FBE5:	C0 04	249	CPY #\$04
FBE7:	90 F2	250	BCC PRNTBL
FBE9:	6B	251	PLA
FBEA:	AB	252	TAY
FBEB:	B9 C0 F9	253	LDA MNEML, Y
FBEF:	B5 2C	254	STA LMNEM
FBF0:	B9 00 FA	255	LDA MNEMR, Y
FBF3:	B5 2D	256	STA RMNEM
FBF5:	A9 00	257	NXTCOL LDA #\$00
FBF7:	A0 05	258	LDY #\$05
FBF9:	06 2D	259	PRMN2 ASL RMNEM
FBFB:	26 2C	260	ROL LMNEM
FBFD:	2A	261	RGL A
FBEF:	BB	262	DEY
FBF0:	D0 FB	263	BNE PRMN2
F901:	69 BF	264	ADC #\$BF
F903:	20 ED FD	265	JSR COUT
F906:	CA	266	DEX
F907:	D0 EC	267	BNE NXTCOL
F909:	20 4B F9	268	JSR PRBLNK
F90C:	A4 2F	269	LDY LENGTH
F90E:	A2 06	270	LDX #\$06
F910:	E0 03	271	PRADR1 CPX #\$03
F912:	F0 1C	272	BEG PRADR5
F914:	06 2E	273	PRADR2 ASL FORMAT
F916:	90 0E	274	BCC PRADR3
F918:	BD B3 F9	275	LDA CHAR1-1, X
F918:	20 ED FD	276	JSR COUT
F91E:	BD B9 F9	277	LDA CHAR2-1, X
F921:	F0 03	278	BEG PRADR3
F923:	20 ED FD	279	JSR COUT
F926:	CA	280	PRADR3 DEX
F927:	D0 E7	281	BNE PRADR1
F929:	60	282	RTS
F92A:	BB	283	PRADR4 DEY
F92B:	30 E7	284	BMI PRADR2
F92D:	20 DA FD	285	JSR PRBYTE
F930:	A5 2E	286	PRADR5 LDA FORMAT
F932:	C9 E8	287	CMP #\$EB

F934:	B1	3A	288	LDA (PCL),Y
F936:	90	F2	289	BCC PRADR4
F938:	20	56	290	PAGE
F938:	20	56	291	RELADR JSR PCADJ3
F93D:	AA		292	TAX
F93C:	EB		293	INX
F93D:	D0	01	294	BNE PRNTYX
F93F:	CB		295	INY
F940:	98		296	PRNTYX TYA
F941:	20	DA	297	PRNTAX JSR PRBYTE
F944:	BA		298	PRNTX TXA
F945:	4C	DA	299	JMP PRBYTE
F948:	A2	03	300	PRBLNK LDX #\$03
F94A:	A9	A0	301	PRBL2 LDA #\$A0
F94C:	20	ED	302	PRBL3 JSR COUT
F94F:	CA		303	DEX
F950:	D0	FB	304	BNE PRBL2
F952:	60		305	RTS
F953:	3B		306	PCADJ SEC
F954:	A5	2F	307	PCADJ2 LDA LENGTH
F956:	A4	3B	308	PCADJ3 LDY PCH
F958:	AA		309	TAX
F959:	10	01	310	BPL PCADJ4
F95B:	BB		311	DEY
F95C:	65	3A	312	PCADJ4 ADC PCL
F95E:	90	01	313	BCC RTS2
F960:	CB		314	INV
F961:	60		315	RTS2 RTS
F962:	04		316	FMT1 DFB \$04
F963:	20		317	DFB \$20
F964:	54		318	DFB \$54
F965:	30		319	DFB \$30
F966:	0D		320	DFB \$0D
F967:	80		321	DFB \$80
F968:	04		322	DFB \$04
F969:	90		323	DFB \$90
F96A:	03		324	DFB \$03
F96B:	22		325	DFB \$22
F96C:	54		326	DFB \$54
F96D:	33		327	DFB \$33
F96E:	0D		328	DFB \$0D
F96F:	80		329	DFB \$80
F970:	04		330	DFB \$04
F971:	90		331	DFB \$90
F972:	04		332	DFB \$04
F973:	20		333	DFB \$20
F974:	54		334	DFB \$54
F975:	33		335	DFB \$33
F976:	0D		336	DFB \$0D
F977:	80		337	DFB \$80
F978:	04		338	DFB \$04
F979:	90		339	DFB \$90
F97A:	04		340	DFB \$04
F97B:	20		341	DFB \$20
F97C:	54		342	DFB \$54
F97D:	33		343	DFB \$33
F97E:	0D		344	DFB \$0D
F97F:	80		345	DFB \$80
F980:	04		346	DFB \$04
F981:	90		347	DFB \$90
F982:	00		348	DFB \$00
F983:	22		349	DFB \$22
F984:	44		350	DFB \$44
F985:	33		351	DFB \$33
F986:	0D		352	DFB \$0D
F987:	CB		353	DFB \$CB
F988:	44		354	DFB \$44
F989:	00		355	DFB \$00
F98A:	11		356	DFB \$11
F98B:	22		357	DFB \$22
F98C:	44		358	DFB \$44
F98D:	33		359	DFB \$33
F98E:	0D		360	DFB \$0D

F9BF:	CB	361	DFB \$CB
F990:	44	362	DFB \$44
F991:	A9	363	DFB \$A9
F992:	01	364	DFB \$01
F993:	22	365	DFB \$22
F994:	44	366	DFB \$44
F995:	33	367	DFB \$33
F996:	00	368	DFB \$00
F997:	80	369	DFB \$80
F998:	04	370	DFB \$04
F999:	90	371	DFB \$90
F99A:	01	372	DFB \$01
F99B:	22	373	DFB \$22
F99C:	44	374	DFB \$44
F99D:	33	375	DFB \$33
F99E:	00	376	DFB \$00
F99F:	80	377	DFB \$80
F9A0:	04	378	DFB \$04
F9A1:	90	379	DFB \$90
F9A2:	26	380	DFB \$26
F9A3:	31	381	DFB \$31
F9A4:	87	382	DFB \$87
F9A5:	9A	383	DFB \$9A
F9A6:	00	384	FMT2
F9A7:	21	385	DFD \$21
F9A8:	B1	386	DFB \$B1
F9A9:	B2	387	DFB \$B2
F9AA:	00	388	DFB \$00
F9AB:	00	389	DFB \$00
F9AC:	59	390	DFB \$59
F9AD:	4D	391	DFB \$4D
F9AE:	91	392	DFB \$91
F9AF:	92	393	DFB \$92
F9B0:	B6	394	DFB \$B6
F9B1:	4A	395	DFB \$4A
F9B2:	B5	396	DFB \$B5
F9B3:	9D	397	DFB \$9D
F9B4:	AC	398	CHAR1
F9B5:	A9	399	DFB \$A9
F9B6:	AC	400	DFB \$AC
F9B7:	A3	401	DFB \$A3
F9B8:	AB	402	DFB \$AB
F9B9:	A4	403	DFB \$A4
F9B9A:	D9	404	CHAR2
F9B9B:	00	405	DFB \$00
F9B9C:	DB	406	DFB \$DB
F9B9D:	A4	407	DFB \$A4
F9B9E:	A4	408	DFB \$A4
F9B9F:	00	409	DFB \$00
F9C0:	1C	410	MNEML
F9C1:	8A	411	DFB \$8A
F9C2:	1C	412	DFD \$1C
F9C3:	23	413	DFB \$23
F9C4:	5D	414	DFD \$5D
F9C5:	BB	415	DFB \$BB
F9C6:	1B	416	DFB \$1B
F9C7:	A1	417	DFB \$A1
F9C8:	9D	418	DFB \$9D
F9C9:	84	419	DFB \$8A
F9CA:	1D	420	DFB \$1D
F9CB:	23	421	DFB \$23
F9CC:	9D	422	DFB \$9D
F9CD:	BB	423	DFB \$BB
F9CE:	1D	424	DFB \$1D
F9CF:	A1	425	DFB \$A1
F9D0:	00	426	DFB \$00
F9D1:	29	427	DFB \$29
F9D2:	19	428	DFB \$19
F9D3:	AE	429	DFB \$AE
F9D4:	69	430	DFB \$69
F9D5:	AB	431	DFB \$AB
F9D6:	19	432	DFB \$19
F9D7:	23	433	DFB \$23

F9DB: 24	434	DFB \$24
F9D9: 53	435	DFB \$53
F9DA: 1B	436	DFB \$1B
F9DB: 23	437	DFB \$23
F9DC: 24	438	DFB \$24
F9DD: 53	439	DFB \$53
F9DE: 19	440	DFB \$19
F9DF: A1	441	DFB \$A1
F9E0: 00	442	DFB \$00
F9E1: 1A	443	DFB \$1A
F9E2: 5B	444	DFB \$5B
F9E3: 5B	445	DFB \$5B
F9E4: A5	446	DFB \$A5
F9E5: 69	447	DFB \$69
F9E6: 24	448	DFB \$24
F9E7: 24	449	DFB \$24
F9EB: AE	450	DFB \$AE
F9E9: AE	451	DFB \$AE
F9EA: AB	452	DFB \$AB
F9EB: AD	453	DFB \$AD
F9EC: 29	454	DFB \$29
F9ED: 00	455	DFB \$00
F9EE: 7C	456	DFB \$7C
F9EF: 00	457	DFB \$00
F9F0: 15	458	DFB \$15
F9F1: 9C	459	DFB \$9C
F9F2: 6D	460	DFB \$6D
F9F3: 9C	461	DFB \$9C
F9F4: A5	462	DFB \$A5
F9F5: 69	463	DFB \$69
F9F6: 29	464	DFB \$29
F9F7: 53	465	DFB \$53
F9FB: B4	466	DFB \$B4
F9F9: 13	467	DFB \$13
F9FA: 34	468	DFB \$34
F9FB: 11	469	DFB \$11
F9FC: A5	470	DFB \$A5
F9FD: 69	471	DFB \$69
F9FE: 23	472	DFB \$23
F9FF: A0	473	DFB \$A0
FA00: D8	474 MNEMR	DFB \$D8
FA01: 62	475	DFB \$62
FA02: 5A	476	DFB \$5A
FA03: 4B	477	DFB \$4B
FA04: 26	478	DFB \$26
FA05: 62	479	DFB \$62
FA06: 94	480	DFB \$94
FA07: 88	481	DFB \$88
FA08: 54	482	DFB \$54
FA09: 44	483	DFB \$44
FA0A: C8	484	DFB \$C8
FA0B: 54	485	DFB \$54
FA0C: 6B	486	DFB \$6B
FA0D: 44	487	DFB \$44
FA0E: EB	488	DFB \$EB
FA0F: 94	489	DFB \$94
FA10: 00	490	DFB \$00
FA11: B4	491	DFB \$B4
FA12: 0B	492	DFB \$0B
FA13: 84	493	DFB \$84
FA14: 74	494	DFB \$74
FA15: B4	495	DFB \$B4
FA16: 2B	496	DFB \$2B
FA17: 6E	497	DFB \$6E
FA18: 74	498	DFB \$74
FA19: F4	499	DFB \$F4
FA1A: CC	500	DFB \$CC
FA1B: 4A	501	DFB \$4A
FA1C: 72	502	DFB \$72
FA1D: F2	503	DFB \$F2
FA1E: A4	504	DFB \$A4
FA1F: 8A	505	DFB \$8A
FA20: 00	506	DFB \$00

FA21: AA	507	DFB \$AA
FA22: A2	508	DFB \$A2
FA23: A2	509	DFB \$A2
FA24: 74	510	DFB \$74
FA25: 74	511	DFB \$74
FA26: 74	512	DFB \$74
FA27: 72	513	DFB \$72
FA28: 44	514	DFB \$44
FA29: 68	515	DFB \$68
FA2A: B2	516	DFB \$B2
FA2B: 32	517	DFB \$32
FA2C: B2	518	DFB \$B2
FA2D: 00	519	DFB \$00
FA2E: 22	520	DFB \$22
FA2F: 00	521	DFB \$00
FA30: 1A	522	DFB \$1A
FA31: 1A	523	DFB \$1A
FA32: 26	524	DFB \$26
FA33: 26	525	DFB \$26
FA34: 72	526	DFB \$72
FA35: 72	527	DFB \$72
FA36: B8	528	DFB \$B8
FA37: CB	529	DFB \$CB
FA38: C4	530	DFB \$C4
FA39: CA	531	DFB \$CA
FA3A: 26	532	DFB \$26
FA3B: 48	533	DFB \$48
FA3C: 44	534	DFB \$44
FA3D: 44	535	DFB \$44
FA3E: A2	536	DFB \$A2
FA3F: CB	537	DFB \$CB
FA40:	538	PAGE
FA40: 85 45	539 IRQ	STA ACC
FA42: 68	540	PLA
FA43: 48	541	PHA
FA44: 04	542	ASL A
FA45: 04	543	ASL A
FA46: 04	544	ASL A
FA47: 30 03	545	BMI BREAK
FA49: 6C FE 03	546	JMP (IRGLOC)
FA4C: 28	547 BREAK	PLP
FA4D: 20 4C FF	548	JSR SAVI
FA50: 68	549	PLA
FA51: 85 3A	550	STA PCL
FA53: 68	551	PLA
FA54: 85 3B	552	STA PCH
FA56: 6C F0 03	553	JMP (BRKV) ; BRKV WRITTEN OVER BY DISK BOOT
FA59: 20 82 F8	554	JSR INSDS1
FA5C: 20 DA FA	555	JSR RDSP1
FA5F: 4C 65 FF	556	JMP MON
FA62: D8	557 RESET	CLD ; DO THIS FIRST THIS TIME
FA63: 20 B4 FE	558	JSR SETNORM
FA66: 20 2F FD	559	JSR INIT
FA69: 20 93 FE	560	JSR SETVID
FA6C: 20 89 FE	561	JSR SETKBD
FA6F: AD 58 CO	562 INITAN	LDA SETANO ; AN0 = TTL HI
FA72: AD 5A CO	563	LDA SETAN1 ; AN1 = TTL HI
FA75: AD 5D CO	564	LDA CLRAN2 ; AN2 = TTL LO
FA78: AD 5F CO	565	LDA CLRAN3 ; AN3 = TTL LO
FA7B: AD FF CF	566	LDA CLRROM ; TURN OFF EXTNSN ROM
FA7E: 2C 10 CO	567	BIT KBDSTRB ; CLEAR KEYBOARD
FAB1: DB	568 NEWMON	CLD
FAB2: 20 3A FF	569	JSR BELL ; CAUSES DELAY IF KEY BOUNCES
FAB5: AD F3 03	570	LDA SOFTEV+1 ; IS RESET HI
FABB: 49 A5	571	EOR #\$E5 ; A FUNNY COMPLEMENT OF THE
FABA: CD F4 03	572	CMP PWREDUP ; PWR UP BYTE ???
FABD: D0 17	573	BNE PWRUP ; NO SO PWRUP
FABF: AD F2 03	574	LDA SOFTEV ; YES SEE IF COLD START
FA92: D0 0F	575	BNE NOFIX ; HAS BEEN DONE YET?
FA94: A9 E0	576	LDA #\$E0 ; ??
FA96: CD F3 03	577	CMP SOFTEV+1 ; ??
FA99: D0 08	578	BNE NOFIX ; YES SO REENTER SYSTEM
FA9B: A0 03	579 FIXSEV	LDY #3 ; NO SO POINT AT WARM START

```

FA9D: BC F2 03      580      STY SOFTEV ; FOR NEXT RESET
FAAO: 4C 00 E0      581      JMP BASIC ; AND DO THE COLD START
FAA3: 6C F2 03      582      NOFIX   JMP (SOFTEV) ; SOFT ENTRY VECTOR
FAA6:                   583 ****
FAA6: 20 60 FB      584      PWRUP   JSR APPLEII
FAA9:                   585 SETPG3 EQU *      ; SET PAGE 3 VECTORS
FAA9: A2 05          586      LDX #5
FAAB: BD FC FA      587      SETPLP  LDA PWRCON-1,X ; WITH CNTRL B ADRS
FAAE: 9D EF 03      588      STA BRKV-1,X ; OF CURRENT BASIC
FAB1: CA             589      DEX
FAB2: D0 F7          590      BNE SETPLP
FAB4: A9 CB          591      LDA #$CB ; LOAD HI SLOT +1
FAB6: B6 00          592      STX LOCO ; SETPG3 MUST RETURN X=0
FAB8: B5 01          593      STA LOC1 ; SET PTR H
FABA: A0 07          594      SLOOP  LDY #7 ; Y IS BYTE PTR
FABC: C6 01          595      DEC LOC1
FABE: A5 01          596      LDA LOC1
FAC0: C9 C0          597      CMP #$C0 ; AT LAST SLOT YET?
FAC2: F0 D7          598      BEG FIXSEV ; YES AND IT CANT BE A DISK
FAC4: BD F8 07          599      STA MSLOT
FAC7: B1 C0          600      NXTBYT LDA (LOCO),Y ; FETCH A SLOT BYTE
FAC9: D9 01 FB          601      CMP DISKID-1,Y ; IS IT A DISK ??
FACC: D0 EC          602      BNE SLOOP ; NO SO NEXT SLOT DOWN
FACE: BB             603      DEY
FACF: BB             604      DEY      ; YES SO CHECK NEXT BYTE
FAD0: 10 F5          605      BPL NXTBYT ; UNTIL 4 CHECKED
FAD2: 6C 00 00          606      JMP (LOCO)
FAD5: EA             607      NOP
FAD6: EA             608      NOP
FAD7:                   609 * REGDSP MUST ORG $FAD7
FAD7: 20 BE FD          610      REGDSP JSR CROUT
FADA: A9 45          611      RGDSP1 LDA #45
FADC: B5 40          612      STA A3L
FADE: A9 00          613      LDA #00
FAE0: B5 41          614      STA A3H
FAE2: A2 FB          615      LDX #$FB
FAE4: A9 A0          616      RDSP1  LDA #AA
FAE6: 20 ED FD          617      JSR COUT
FAE9: BD 1E FA          618      LDA RTBL-251,X
FAEC: 20 ED FD          619      JSR COUT
FAEF: A9 BD          620      LDA ##BD
FAF1: 20 ED FD          621      JSR COUT
FAF4: B5 4A          622 * LDA ACC+5,X
FAF6: 20 DA FD          623      DFB $B5,$4A
FAF9: E8             624      JSR PRBYTE
FAFA: 30 EB          625      INX
FAFC: 60             626      BMI RDSP1
FAFD: 59 FA          628      PWRCON DW OLDBRK
FAFF: 00 E0 45          629      DFB $00,$E0,$45
FB02: 20 FF 00          630      DISKID  DFB $20,$FF,$00,$FF
FB06: 03 FF 3C          631      DFB $03,$FF,$3C
FB09: C1 D0 D0          632      TITLE   DFB $C1,$D0,$D0
FB0C: CC C5 A0          633      DFB $CC,$C5,$A0
FB0F: DD DB          634      DFB $DD,$DB
FB11:                   635 XLTBL EQU *
FB11: C4 C2 C1          636      DFB $C4,$C2,$C1
FB14: FF C3          637      DFB $FF,$C3
FB16: FF FF FF          638      DFB $FF,$FF,$FF
FB19:                   639 * MUST ORG $FB19
FB19: C1 DB D9          640      RTBL   DFB $C1,$DB,$D9
FB1C: D0 D3          641      DFB $D0,$D3
FB1E: AD 70 C0          642      PREAD  LDA PTRIG
FB21:                   643      LST ON
FB21: A0 00          644      LDY #$00
FB23: EA             645      NOP
FB24: EA             646      NOP
FB25: BD 64 C0          647      PREAD2 LDA PADDLO,X
FB28: 10 04          648      BPL RTS2D
FB2A: C8             649      INY
FB2B: D0 FB          650      BNE PREAD2
FB2D: BB             651      DEY

```

```

FB2E: 60      652 RTS2D   RTS
FB2F: A9 00    2 INIT    LDA #$00
FB31: B5 4B    3 STA STATUS
FB33: AD 56 C0 4 LDA LORES
FB36: AD 54 C0 5 LDA LOWSCR
FB39: AD 51 C0 6 SETTXT
FB3C: A9 00    7 LDA #$00
FB3E: F0 00    8 BEG SETWND
FB40: AD 50 C0 9 SETGR
FB43: AD 53 C0 10 LDA MIXSET
FB46: 20 36 FB 11 JSR CLRTOP
FB49: A9 14    12 LDA #$14
FB4B: B5 22    13 SETWND STA WNDTOP
FB4D: A9 00    14 LDA #$00
FB4F: B5 20    15 STA WNDLFT
FB51: A9 28    16 LDA #$28
FB53: B5 21    17 STA WNDWDTH
FB55: A9 18    18 LDA #$18
FB57: B5 23    19 STA WNDBTM
FB59: A9 17    20 LDA #$17
FB5B: B5 25    21 TABV   STA CV
FB5D: 4C 22 FC 22 JMP VTAB
FB60: 20 5B FC 23 APPLEII JSR HOME ; CLEAR THE SCRN
FB63: A0 08    24 LDY #8
FB65: B9 0B FB 25 STITLE LDA TITLE-1,Y ; GET A CHAR
FB68: 99 0E 04 26 STA LINE1+14,Y
FB6B: B6        27 DEY
FB6C: D0 F7    28 BNE STITLE
FB6E: 60        29 RTS
FB6F: AD F3 03 30 SETPWRC LDA SOFTEV+1
FB72: 49 A5    31 EOR #$A5
FB74: BD F4 03 32 STA PWREDUP
FB77: 60        33 RTS
FB78:          34 VIDWAIT EQU *      ; CHECK FOR A PAUSE
FB7B: C9 BD    35 CMP #$BD ; ONLY WHEN I HAVE A CR
FB7A: D0 18    36 BNE NOWAIT ; NOT SO, DO REGULAR
FB7C: AC 00 C0 37 LDY KBD ; IS KEY PRESSED?
FB7F: 10 13    38 BPL NOWAIT ; NO
FB81: C0 93    39 CPY #$93 ; IS IT CTL S ?
FB83: D0 0F    40 BNE NOWAIT ; NO SO IGNORE
FB85: 2C 10 CO 41 BIT KBDSTRB ; CLEAR STROBE
FB88: AC 00 C0 42 KBDWAIT LDY KBD ; WAIT TILL NEXT KEY TO RESUME
FB8B: 10 FB    43 BPL KBDWAIT ; WAIT FOR KEYPRESS
FB8D: C0 B3    44 BIT KBDSTRB ; CLR STROBE
FB8F: F0 03    45 BEG NOWAIT ; YES SO LEAVE IT
FB91: 2C 10 CO 46 BIT KBDSTRB ; CLR STROBE
FB94: 4C FD FB 47 NOWAIT JMP VIDOUT ; DO AS BEFORE
FB97:          48 PAGE
FB97: 38        49 ESCOLD SEC      ; INSURE CARRY SET
FB98: 4C 2C FC 50 JMP ESC1
FB9B: A9        51 ESCNOW TAY      ; USE CHAR AS INDEX
FB9C: B9 4B FA 52 LDA XLTBL-$C9,Y ; XLATE IJKM TO CBAD
FB9F: 20 97 FB 53 JSR ESCOLD ; DO THIS CURSOR MOTION
FBAD: 20 0C FD 54 JSR RDKEY ; AND GET NEXT
FBAA: C9 CE    55 ESCNEW CMP #$CE ; IS THIS AN N ?
FBAB: B0 EE    56 BCS ESCOLD ; N OR GREATER DO IT
FBAC: C9 C9    57 CMP #$C9 ; LESS THAN I ?
FBAD: 90 EA    58 BCC ESCOLD ; YES SO OLD WAY
FBAD: C9 CC    59 CMP #$CC ; IS IT A L ?
FBAF: F0 E6    60 BEG ESCOLD ; DO NORMAL
FBB1: D0 EB    61 BNE ESCNOW ; GO DO IT
FBB3: EA        62 NOP
FBB4: EA        63 NOP
FBB5: EA        64 NOP
FBB6: EA        65 NOP
FBB7: EA        66 NOP
FBB8: EA        67 NOP
FBB9: EA        68 NOP
FBBA: EA        69 NOP

```

FBBB:	EA	70	NDP
FDBC:	EA	71	NDP
FBDI:	EA	72	NDP
FBBE:	EA	73	NDP
FBBF:	EA	74	NDP
FBC0:	EA	75	NDP
FBC1:		76	* MUST ORG \$FBC1
FBC1:	4B	77	BASCALC PHA
FBC2:	4A	78	LSR A
FBC3:	29 03	79	AND #\$03
FBC5:	09 04	80	ORA #\$04
FBC7:	85 29	81	STA BASH
FBC9:	6B	82	PLA
FBCA:	29 1B	83	AND #\$1B
FBCB:	90 02	84	BCC BASCLC2
FBCF:	69 7F	85	ADC #\$7F
FBD0:	85 28	86	BASCLC2 STA BASL
FBD2:	0A	87	ASL A
FBD3:	0A	88	ASL A
FBD4:	05 2B	89	ORA BASL
FBD6:	85 2B	90	STA BASL
FBD8:	60	91	RTS
FBD9:	C9 87	92	BELL1 CMP #\$B7
FBD8:	D0 12	93	BNE RTS2B
FBDI:	A9 40	94	LDA #\$40
FBDF:	20 AB FC	95	JSR WAIT
FBE2:	A0 C0	96	LDY #\$C0
FBE4:	A9 0C	97	BELL2 LDA #\$0C
FBE6:	20 AB FC	98	JSR WAIT
FBE9:	AD 30 C0	99	LDA SPKR
FBEc:	8B	100	DEY
FBED:	D0 F5	101	BNE BELL2
FBEF:	60	102	RTS2B RTS
FBF0:		103	PAGE
FBF0:	A4 24	104	STORADV LDY CH
FBF2:	91 2B	105	STA (BASL), Y
FBF4:	E6 24	106	ADVANCE INC CH
FBF6:	A5 24	107	LDA CH
FBF8:	C5 21	108	CMP WNDWDTH
FBF9:	B0 66	109	BCS CR
FBFC:	60	110	RTS3 RTS
FBFD:	C9 A0	111	VIDOUT CMP #\$A0
FBFF:	B0 EF	112	BCS STORADV
FC01:	AB	113	TAY
FC02:	10 EC	114	BPL STORADV
FC04:	C9 BD	115	CMP #\$BD
FC06:	F0 5A	116	BEG CR
FC08:	C9 BA	117	CMP #\$BA
FC0A:	F0 5A	118	BEG LF
FC0C:	C9 BB	119	CMP #\$BB
FC0E:	D0 C9	120	BNE BELL1
FC10:	C6 24	121	BS DEC CH
FC12:	10 EB	122	BPL RTS3
FC14:	A5 21	123	LDA WNDWDTH
FC16:	B5 24	124	STA CH
FC18:	C6 24	125	DEC CH
FC1A:	A5 22	126	UP LDA WNDTOP
FC1C:	C5 25	127	CMP CV
FC1E:	B0 0B	128	BCS RTS4
FC20:	C6 25	129	DEC CV
FC22:	A5 25	130	VTAB LDA CV
FC24:	20 C1 FB	131	VTABZ JSR BASCALC
FC27:	65 20	132	ADC WNDLFT
FC29:	85 28	133	STA BASL
FC2B:	60	134	RTS4 RTS
FC2C:	49 C0	135	ESC1 EOR #\$C0 ; ESC @ ?
FC2E:	F0 28	136	BEQ HOME ; IF SO DO HOME AND CLEAR
FC30:	69 FD	137	ADC #\$FD ; ESC-A OR B CHECK
FC32:	90 C0	138	BCC ADVANCE ; A, ADVANCE
FC34:	F0 DA	139	BEG BS ; B, BACKSPACE
FC36:	69 FD	140	ADC #\$FD ; ESC-C OR D CHECK
FC38:	90 2C	141	BCC LF ; C, DOWN
FC3A:	F0 DE	142	BEG UP ; D, GO UP

FC3C: 69 FD	143	ADC #\$FD ; ESC-E OR F CHECK
FC3E: 90 5C	144	BCC CLREOL ; E, CLEAR TO END OF LINE
FC40: D0 E9	145	BNE RTS4 ; ELSE NOT F, RETURN
FC42: A4 24	146	CLREOP LDY CH ; ESC F IS CLR TO END OF PAGE
FC44: A5 25	147	LDA CV
FC46: 4B	148	CLEOP1 PHA
FC47: 20 24 FC	149	JSR VTABZ
FC4A: 20 9E FC	150	JSR CLEOLZ
FC4D: A0 00	151	LDY #\$00
FC4F: 6B	152	PLA
FC50: 69 00	153	ADC #\$00
FC52: C5 23	154	CMP WNDBTM
FC54: 90 F0	155	BCC CLEOP1
FC56: B0 CA	156	BCS VTAB
FC58: A5 22	157	HOME LDA WNDTOP
FC5A: B5 25	158	STA CV
FC5C: A0 00	159	LDY #\$00
FC5E: B4 24	160	STY CH
FC60: F0 E4	161	BEG CLEOP1
FC62:	162	PAGE
FC62: A9 00	163	CR LDA #\$00
FC64: B5 24	164	STA CH
FC66: E6 25	165	LF INC CV
FC68: A5 25	166	LDA CV
FC6A: C5 23	167	CMP WNDBTM
FC6C: 90 B6	168	BCC VTABZ
FC6E: C6 25	169	DEC CV
FC70: A5 22	170	SCROLL LDA WNDTOP
FC72: 4B	171	PHA
FC73: 20 24 FC	172	JSR VTABZ
FC76: A5 2B	173	SCRL1 LDA BASL
FC78: B5 2A	174	STA BASL2
FC7A: A5 29	175	LDA BASH
FC7C: B5 2B	176	STA BASL2H
FC7E: A4 21	177	LDY WNDWDTH
FC80: B8	178	DEY
FC81: 6B	179	PLA
FC82: 69 01	180	ADC #\$01
FC84: C5 23	181	CMP WNDBTM
FC86: B0 0D	182	BCS SCRL3
FC88: 4B	183	PHA
FC89: 20 24 FC	184	JSR VTABZ
FC8C: B1 2B	185	SCRL2 LDA (BASL), Y
FC8E: 91 2A	186	STA (BASL2), Y
FC90: B8	187	DEY
FC91: 10 F9	188	BPL SCRL2
FC93: 30 E1	189	BMI SCRL1
FC95: A0 00	190	SCRL3 LDY #\$00
FC97: 20 9E FC	191	JSR CLEOLZ
FC9A: B0 B6	192	BCS VTAB
FC9C: A4 24	193	CLREOL LDY CH
FC9E: A9 A0	194	CLEOLZ LDA #\$A0
FCA0: 91 2B	195	CLEOL2 STA (BASL), Y
FCA2: CB	196	INY
FCA3: C4 21	197	CPY WNDWDTH
FCA5: 90 F9	198	BCC CLEOL2
FCA7: 60	199	RTS
FCAB: 3B	200	WAIT SEC
FCA9: 4B	201	WAIT2 PHA
FCAA: E9 01	202	WAIT3 SBC #\$01
FCAC: D0 FC	203	BNE WAIT3
FCAE: 6B	204	PLA
FCAF: E9 01	205	SBC #\$01
FCB1: D0 F6	206	BNE WAIT2
FCB3: 60	207	RTS
FCB4: E6 42	208	NXTA4 INC A4L
FCB6: D0 02	209	BNE NXTA1
FCB8: E6 43	210	INC A4H
FCBA: A5 3C	211	NXTA1 LDA A1L
FCBC: C5 3E	212	CMP A2L
FCBE: A5 3D	213	LDA A1H
FCC0: E5 3F	214	SBC A2H
FCC2: E6 3C	215	INC A1L

FCC4: D0 02	216	BNE RTS4B
FCC6: E6 3D	217	INC A1H
FCCB: 60	218	RTS
FCC9:	219	PAGE
FCC9: A0 4B	220	HEADR
FCCB: 20 DB FC	221	LDY #\$4B
FCCE: D0 F9	222	JSR ZERDLY
FCD0: 69 FE	223	BNE HEADR
FCD2: B0 F5	224	ADC #\$FE
FCD4: A0 21	225	BCS HEADR
FCD6: 20 DB FC	226	LDY #\$21
FCDA: CB	227	JSR ZERDLY
FCDB: CB	228	INY
FCDB: BB	229	INY
FCD9: D0 FD	230	ZERDLY
FCDE: 90 05	231	DEY
FCEO: A0 32	232	BNE ONEDLY
FCE2: BB	233	LDY #\$32
FCE3: D0 FD	234	ONEDLY
FCE5: AC 20 CO	235	WRTAPE
FCEB: A0 2C	236	LDY TAPEOUT
FCEA: CA	237	LDY #\$2C
FCEB: 60	238	DEX
FCEC: A2 08	239	RTS
FCEE: 4B	240	RDBYT2
FCEF: 20 FA FC	241	PHA
FCF2: 6B	242	JSR RD2BIT
FCF3: 2A	243	PLA
FCF4: A0 3A	244	ROL A
FCF6: CA	245	LDY #\$3A
FCF7: D0 F5	246	DEX
FCF9: 60	247	RDBYT2
FCFA: 20 FD FC	248	RTS
FCFD: BB	249	RD2BIT
FCFE: AD 60 CO	250	DEY
FD01: 45 2F	251	LDA TAPEIN
FD03: 10 FB	252	EOR LASTIN
FD05: 45 2F	253	BPL RD2BIT
FD07: B5 2F	254	EOR LASTIN
FD09: C0 80	255	STA LASTIN
FD0B: 60	256	CPY #\$80
FD0C: A4 24	257	RTS
FD0E: B1 2B	258	RDKEY
FD10: 4B	259	LDY CH
FD11: 29 3F	260	LDA (BASL), Y
FD13: 09 40	261	PHA
FD15: 91 2B	262	AND #\$3F
FD17: 6B	263	ORA #\$40
FD18: 6C 38 00	264	STA (BASL), Y
FD1B: E6 4E	265	PLA
FD1D: D0 02	266	JMP (KSWL)
FD1F: E6 4F	267	INC RNDL
FD21: 2C 00 CO	268	INC RNDH
FD21: 2C 00 CO	268	KEYIN2
FD24: 10 F5	269	BIT KBD / READ KEYBOARD
FD26: 91 2B	270	BPL KEYIN
FD28: AD 00 CO	271	STA (BASL), Y
FD2B: 2C 10 CO	272	LDA KBD
FD2E: 60	273	BIT KBDSTRB
FD2F: 20 0C FD	274	RTS
FD2F: 20 0C FD	274	ESC
FD32: 20 A5 FB	275	JSR RDKEY
FD35: 20 0C FD	276	JSR ESCNEW
FD38: C9 9B	277	JSR RDKEY
FD3A: F0 F3	278	CMP #\$9B
FD3C: 60	279	BEG ESC
FD3D:	280	RTS
FD3D: A5 32	281	PAGE
FD3F: 4B	282	NOTCR
FD40: A9 FF	283	LDA INVFLG
FD42: B5 32	284	PHA
FD44: BD 00 02	285	LDA #\$FF
FD47: 20 ED FD	286	STA INVFLG
FD4A: 6B	287	JSR IN, X
FD4B: B5 32	288	JSR COUT
FD4B: B5 32	288	PLA
FD4B: B5 32	288	STA INVFLG

```

FD4D: BD 00 02      289      LDA IN, X
FD50: C9 BB          290      CMP #$BB
FD52: F0 1D          291      BEQ BCKSPC
FD54: C9 9B          292      CMP #$9B
FD56: F0 0A          293      BEG CANCEL
FD58: E0 FB          294      CPX #$FB
FD5A: 90 03          295      BCC NOTCR1
FD5C: 20 3A FF       296      JSR BELL
FD5F: E6              297      NOTCR1 INX
FD60: D0 13          298      BNE NXTCHAR
FD62: A9 DC          299      CANCEL LDA #$DC
FD64: 20 ED FD       300      JSR COUT
FD67: 20 BE FD       301      GETLNZ JSR CROUT
FD6A: A5 33          302      GETLN LDA PROMPT
FD6C: 20 ED FD       303      JSR COUT
FD6F: A2 01          304      LDX #$01
FD71: BA              305      BCKSPC TXA
FD72: F0 F3          306      BEQ GETLNZ
FD74: CA              307      DEX
FD75: 20 35 FD       308      NXTCHAR JSR RDCHAR
FD78: C9 95          309      CMP #$95
FD7A: D0 02          310      BNE CAPTST
FD7C: B1 2B          311      LDA (BASL), Y
FD7E: C9 E0          312      CAPTST CMP #$E0
FD80: 90 02          313      BCC ADDINP
FD82: 29 DF          314      AND #$DF ; SHIFT TO UPPER CASE
FD84: 9D 00 02       315      ADDINP STA IN, X
FD87: C9 BD          316      CMP #$BD
FD89: D0 B2          317      BNE NOTCR
FD8B: 20 9C FC       318      JSR CLREOL
FD8E: A9 BD          319      CROUT LDA #$BD
FD90: D0 5B          320      BNE COUT
FD92: A4 3D          321      PRA1 LDY A1H
FD94: A6 3C          322      LDX A1L
FD96: 20 BE FD       323      PRYX2 JSR CROUT
FD99: 20 40 F9       324      JSR PRNTYX
FD9C: A0 00          325      LDY #$00
FD9E: A9 AD          326      LDA #$AD
FDA0: 4C ED FD       327      JMP COUT
FDA3:                 328      PAGE
FDA3: A5 3C          329      XAMB LDA A1L
FDA5: 09 07          330      DRA #$07
FDA7: 85 3E          331      STA A2L
FDA9: A5 3D          332      LDA A1H
FDAB: 85 3F          333      STA A2H
FDAF: A5 3C          334      MODBCHK LDA A1L
FDAF: 29 07          335      AND #$07
FDB1: D0 03          336      BNE DATAOUT
FDB3: 20 92 FD       337      XAM JSR PRA1
FDB6: A9 A0          338      DATAOUT LDA #$A0
FDBB: 20 ED FD       339      JSR COUT
FDBB: B1 3C          340      LDA (A1L), Y
FDBD: 20 DA FD       341      JSR PRBYTE
FDCC: 20 BA FC       342      JSR NXTA1
FDC3: 90 EB          343      BCC MODBCHK
FDC5: 60              344      RTS4C RTS
FDC6: 4A              345      XAMPM LSR A
FDC7: 90 EA          346      BCC XAM
FDC9: 4A              347      LSR A
FDCA: 4A              348      LSR A
FDCA: A5 3E          349      LDA A2L
FDCC: 90 02          350      BCC ADD
FDCE: 49 FF          351      EDR #$FF
FDD1: 65 3C          352      ADD ADC A1L
FDD3: 48              353      PHA
FDD4: A9 BD          354      LDA #$BD
FDD6: 20 ED FD       355      JSR COUT
FDD9: 6B              356      PLA
FDDA: 4B              357      PRBYTE PHA
FDDB: 4A              358      LSR A
FDCC: 4A              359      LSR A
FDDD: 4A              360      LSR A
FDDE: 4A              361      LSR A

```

FDDF:	20 E5 FD	362	JSR PRHEXZ
FDE2:	68	363	PLA
FDE3:	29 0F	364	PRHEX AND #\$0F
FDE5:	09 B0	365	PRHEXZ ORA #\$B0
FDE7:	C9 BA	366	CMP #\$BA
FDE9:	90 02	367	BCC COUT
FDEB:	69 06	368	ADC #\$06
FDED:	6C 36 00	369	COUT JMP (CSWL)
FDF0:	C9 A0	370	COUT1 CMP #\$A0
FDF2:	90 02	371	BCC COUTZ
FDF4:	25 32	372	AND INVFLG
FDF6:	84 35	373	COUTZ STY YSAV1
FDFB:	4B	374	PHA
FDF9:	20 7B FB	375	JSR VIDWAIT ; GD CHECK FOR PAUSE
FDFA:	6B	376	PLA
FDFD:	A4 35	377	LDY YSAV1
FDFE:	60	378	RTS
FE00:		379	PAGE
FE00:	C6 34	380	BL1 DEC YSAV
FE02:	F0 9F	381	DEG XAMB
FE04:	CA	382	BLANK DEX
FE05:	D0 16	383	BNE SETMDZ
FE07:	C9 BA	384	CMP #\$BA
FE09:	D0 BB	385	BNE XAMPM
FE0B:	85 31	386	STO MODE
FE0D:	A5 3E	387	LDA A2L
FE0F:	91 40	388	STA (A3L),Y
FE11:	E6 40	389	INC A3L
FE13:	D0 02	390	BNE RTS5
FE15:	E6 41	391	INC A3H
FE17:	60	392	RTS5 RTS
FE18:	A4 34	393	SETMODE LDY YSAV
FE1A:	89 FF 01	394	LDA IN-1,Y
FE1D:	85 31	395	SETMDZ STA MODE
FE1F:	60	396	RTS
FE20:	A2 01	397	LT LDX #\$01
FE22:	B5 3E	398	LT2 LDA A2L,X
FE24:	95 42	399	STA A4L,X
FE26:	95 44	400	STA A5L,X
FE28:	CA	401	DEX
FE29:	10 F7	402	BPL LT2
FE2B:	60	403	RTS
FE2C:	B1 3C	404	MOVE LDA (A1L),Y
FE2E:	91 42	405	STA (A4L),Y
FE30:	20 B4 FC	406	JSR NXTA4
FE33:	90 F7	407	BCC MOVE
FE35:	60	408	RTS
FE36:	B1 3C	409	VFY LDA (A1L),Y
FE38:	D1 42	410	CMP (A4L),Y
FE3A:	F0 1C	411	BEQ VFYOK
FE3C:	20 92 FD	412	JSR PRA1
FE3F:	B1 3C	413	LDA (A1L),Y
FE41:	20 DA FD	414	JSR PRBYTE
FE44:	A9 A0	415	LDA #\$A0
FE46:	20 ED FD	416	JSR COUT
FE49:	A9 AB	417	LDA #\$AB
FE4B:	20 ED FD	418	JSR COUT
FE4E:	B1 42	419	LDA (A4L),Y
FE50:	20 DA FD	420	JSR PRBYTE
FE53:	A9 A9	421	LDA #\$A9
FE55:	20 ED FD	422	JSR COUT
FE58:	20 B4 FC	423	VFYOK JSR NXTA4
FE5B:	90 D9	424	BCC VFY
FE5D:	60	425	RTS
FE5E:	20 75 FE	426	LIST JSR AIPC
FE61:	A9 14	427	LDA #\$14
FE63:	48	428	LIST2 PHA
FE64:	20 D0 FB	429	JSR INSTDSP
FE67:	20 53 F9	430	JSR PCADJ
FE6A:	85 3A	431	STA PCL
FE6C:	84 3B	432	STY PCH
FE6E:	6B	433	PLA
FE6F:	3B	434	SEC

FE70:	E9 01	435	SBC #\$01
FE72:	D0 EF	436	BNE LIST2
FE74:	60	437	RTS
FE75:		438	PAGE
FE75:	BA	439	A1PC TXA
FE76:	F0 07	440	BEG A1PCRTS
FE78:	B5 3C	441	A1PCLP LDA A1L,X
FE7A:	95 3A	442	STA PCL,X
FE7C:	CA	443	DEX
FE7D:	10 F9	444	BPL A1PCLP RTS
FE7F:	60	445	A1PCRTS
FE80:	A0 3F	446	SETINV LDY #\$3F
FEB2:	D0 02	447	BNE SETIFLG
FEB4:	A0 FF	448	SETNORM LDY #\$FF
FEB6:	B4 32	449	SETIFLG STY INVFLG
FEB8:	60	450	RTS
FEB9:	A9 00	451	SETKBD LDA #\$00
FEBB:	B5 3E	452	IMPORT STA A2L
FEBD:	A2 38	453	INPRT LDX #CSWL
FEBF:	A0 18	454	LDY #KEYIN
FE91:	D0 08	455	BNE IDPRT
FE93:	A9 00	456	SETVID LDA #\$00
FE95:	B5 3E	457	OUTPORT STA A2L
FE97:	A2 36	458	OUTPRT LDX #CSWL
FE99:	A0 F0	459	LDY #COUT1
FE9B:	A5 3E	460	IDPRT LDA A2L
FE9D:	29 OF	461	AND #\$0F
FE9F:	F0 06	462	BEG IDPRT1
FEA1:	09 C0	463	ORA #IADDR/256
FEA3:	A0 00	464	LDY #\$00
FEA5:	F0 02	465	BEG IDPRT2
FEA7:	A9 FD	466	IDPRT1 LDA #COUT1/256
FEA9:		467	IDPRT2 EQU *
FEA9:	94 00	468	STY LOCO,X ; \$94,\$00
FEAB:	95 01	469	STA LOC1,X ; \$95,\$01
FEAD:	60	470	RTS
FEAE:	EA	471	NOP
FEAF:	EA	472	NOP
FEB0:	4C 00 EO	473	XBASIC JMP BASIC
FEB3:	4C 03 EO	474	BASCONT JMP BASIC2
FEB6:	20 75 FE	475	Q0 JSR A1PC
FEB9:	20 3F FF	476	JSR RESTORE
FEBC:	6C 3A 00	477	JMP (PCL)
FEBF:	4C D7 FA	478	REGZ JMP REGDSP
FEC2:	60	479	TRACE RTS
FEC3:		480	* TRACE IS GONE
FEC3:	EA	481	NOP
FEC4:	60	482	STEPZ RTS ; STEP IS GONE
FEC5:	EA	483	NOP
FEC6:	EA	484	NOP
FEC7:	EA	485	NOP
FEC8:	EA	486	NOP
FEC9:	EA	487	NOP
FECA:	4C FB 03	488	USR JMP USRADR
FECD:		489	PAGE
FECD:	A9 40	490	WRITE LDA #\$40
FECF:	20 C9 FC	491	JSR HEADER
FED2:	A0 27	492	LDY #\$27
FED4:	A2 00	493	WR1 LDX #\$00
FED6:	41 3C	494	EDR (A1L,X)
FED8:	4B	495	PHA
FED9:	A1 3C	496	LDA (A1L,X)
FEDB:	20 ED FE	497	JSR WRBYTE
FEDF:	20 BA FC	498	JSR NXTA1
FEE1:	A0 1D	499	LDY #\$1D
FEE3:	6B	500	PLA
FEE4:	90 EE	501	BCC WR1
FEE6:	A0 22	502	LDY #\$22
FEE8:	20 ED FE	503	JSR WRBYTE
FEEB:	F0 4D	504	BEG BELL
FEED:	A2 10	505	WRBYTE LDX #\$10
FEEF:	0A	506	WRBYT2 ASL A
FEFO:	20 D6 FC	507	JSR WRBIT

FEF3: D0 FA	508	BNE WRBYT2
FEF5: 60	509	RTS
FEF6: 20 00 FE	510	CRMON JSR BL1
FEF9: 68	511	PLA
FEFA: 6B	512	PLA
FEFB: D0 6C	513	BNE MONZ
FEFD: 20 FA FC	514	READ JSR RD2BIT
FF00: A9 16	515	LDA #\$16
FF02: 20 C9 FC	516	JSR HEADR
FF05: B5 2E	517	STA CHKSUM
FF07: 20 FA FC	518	JSR RD2BIT
FF0A: A0 24	519	LDY #\$24
FF0C: 20 FD FC	520	JSR RD8BIT
FF0F: B0 F9	521	BCS RD2
FF11: 20 FD FC	522	JSR RD8BIT
FF14: A0 3B	523	LDY #\$3B
FF16: 20 EC FC	524	RD3 JSR RD8BYTE
FF19: B1 3C	525	STA (A1L X)
FF1B: 45 2E	526	EOR CHKSUM
FF1D: B5 2E	527	STA CHKSUM
FF1F: 20 BA FC	528	JSR NXTA1
FF22: A0 35	529	LDY #\$35
FF24: 90 FO	530	BCC RD3
FF26: 20 EC FC	531	JSR RD8BYTE
FF29: C5 2E	532	CMP CHKSUM
FF2B: F0 0D	533	BEG BELL
FF2D: A9 C5	534	LDA #\$C5
FF2F: 20 ED FD	535	JSR COUT
FF32: A9 D2	536	LDA #\$D2
FF34: 20 ED FD	537	JSR COUT
FF37: 20 ED FD	538	JSR COUT
FF3A: A9 B7	539	BELL LDA #\$B7
FF3C: 4C ED FD	540	JMP COUT
FF3E:	541	PAGE
FF3F: A5 48	542	RESTORE LDA STATUS
FF41: 4B	543	PHA
FF42: A5 45	544	LDA A5H
FF44: A6 46	545	RESTR1 LDX XREG
FF46: A4 47	546	LDY YREG
FF48: 28	547	PLP
FF49: 60	548	RTS
FF4A: B5 45	549	SAVE STA A5H
FF4C: B6 46	550	STX XREG
FF4E: B4 47	551	STY YREG
FF50: 0B	552	PHP
FF51: 6B	553	PLA
FF52: B5 48	554	STA STATUS
FF54: BA	555	TSX
FF55: B6 49	556	STX SPNT
FF57: D8	557	CLD
FF58: 60	558	RTS
FF59: 20 B4 FE	559	OLDRST JSR SETNORM
FF5C: 20 2F FB	560	JSR INIT
FF5F: 20 93 FE	561	JSR SETVID
FF62: 20 89 FE	562	JSR SETKBD
FF65:	563	PAGE
FF65: D8	564	CLD
FF66: 20 3A FF	565	JSR BELL
FF69: A9 AA	566	MONZ LDA #8AA
FF6B: B5 33	567	STA PROMPT
FF6D: 20 67 FD	568	JSR GETLNZ
FF70: 20 C7 FF	569	JSR ZMODE
FF73: 20 A7 FF	570	NXTITM JSR GETNUM
FF76: B4 34	571	STY YSAV
FF78: A0 17	572	LDY #\$17
FF7A: B8	573	CHRSRCH DEY
FF7B: 30 EB	574	BMI MON
FF7D: D9 CC FF	575	CMP CHRTBL, Y
FF80: D0 FB	576	BNE CHRSRCH
FF82: 20 BE FF	577	JSR TOSUB
FF85: A4 34	578	LDY YSAV
FF87: 4C 73 FF	579	JMP NXTITM
FF8A: A2 03	580	DIG LDX #03

FFBC: 0A	581	ASL A	
FFBD: 0A	582	ASL A	
FFBE: 0A	583	ASL A	
FFBF: 0A	584	ASL A	
FF90: 0A	585 NXTBIT	ASL A	
FF91: 26 3E	586	ROL A2L	
FF93: 26 3F	587	ROL A2H	
FF95: CA	588	DEX	
FF96: 10 FB	589	BPL NXTBIT	
FF9B: A5 31	590 NXTBAS	LDA MODE	
FF9A: D0 06	591	BNE NXTBS2	
FF9C:	592 *		
FF9C: B5 3F	593	LDA A2H, X	
FF9E:	594 *		
FF9E: 95 3D	595	STA A1H, X	
FFAO:	596 *		
FFAO: 95 41	597	STA A3H, X	
FFA2: E8	598 NXTBS2	INX	
FFA3: F0 F3	599	BEQ NXTBAS	
FFA5: D0 06	600	BNE NXTCHR	
FFA7: A2 00	601 GETNUM	LDX #\$00	
FFA9: B6 3E	602	STX A2L	
FFAB: B6 3F	603	STX A2H	
FFAD: B9 00 02	604 NXTCHR	LDA IN, Y	
FFB0: C8	605	INY	
FFB1: 49 B0	606	EOR #\$B0	
FFB3: C9 0A	607	CMP #\$0A	
FFB5: 90 D3	608	BCC DIG	
FFB7: 69 BB	609	ADC #\$BB	
FFB9: C9 FA	610	CMP #\$FA	
FFBB: B0 CD	611	BCS DIG	
FFBD: 60	612	RTS	
FFBE: A9 FE	613 TOSUB	LDA #GD/256	
FFC0: 48	614	PHA	
FFC1: B9 E3 FF	615	LDA SUBTBL, Y	
FFC4: 48	616	PHA	
FFC5: A5 31	617	LDA MODE	
FFC7: A0 00	618 ZMODE	LDY #\$00	
FFC9: B4 31	619	STY MODE	
FFCB: 60	620	RTS	
FFCC:	621	PAGE	
FFCC: BC	622 CHRTBL	DFB \$BC	
FFCD: B2	623	DFB \$B2	
FFCE: BE	624	DFB \$BE	
FFCF: B2	625	DFB \$B2	; T CMD NOW LIKE USR
FFD0: EF	626	DFB \$EF	
FFD1: C4	627	DFB \$C4	
FFD2: B2	628	DFB \$B2	; S CMD NOW LIKE USR
FFD3: A9	629	DFB \$A9	
FFD4: BB	630	DFB \$BB	
FFD5: A6	631	DFB \$A6	
FFD6: A4	632	DFB \$A4	
FFD7: 06	633	DFB \$06	
FFD8: 95	634	DFB \$95	
FFD9: 07	635	DFB \$07	
FFDA: 02	636	DFB \$02	
FFDB: 05	637	DFB \$05	
FFDC: F0	638	DFB \$F0	
FFDD: 00	639	DFB \$00	
FFDE: E8	640	DFB \$EB	
FFDF: 93	641	DFB \$93	
FFE0: A7	642	DFB \$A7	
FFE1: C6	643	DFB \$C6	
FFE2: 99	644	DFB \$99	
FFE3: B2	645 SUBTBL	DFB \$B2	
FFE4: C9	646	DFB \$C9	
FFE5: BE	647	DFB \$BE	
FFE6: C1	648	DFB \$C1	
FFE7: 35	649	DFB \$35	
FFE8: BC	650	DFB \$BC	
FFE9: C4	651	DFB \$C4	
FFEA: 96	652	DFB \$96	
FFEB: AF	653	DFB \$AF	

FFEC: 17	654	DFB \$17
FFED: 17	655	DFB \$17
FFEE: 28	656	DFB \$28
FFEF: 1F	657	DFB \$1F
FFFO: 83	658	DFB \$83
FFF1: 7F	659	DFB \$7F
FFF2: 5D	660	DFB \$5D
FFF3: CC	661	DFB \$CC
FFF4: 85	662	DFB \$85
FFF5: FC	663	DFB \$FC
FFF6: 17	664	DFB \$17
FFF7: 17	665	DFB \$17
FFF8: F5	666	DFB \$F5
FFF9: 03	667	DFB \$03
FFFA: FB 03	668	DW NMI
FFFC: 62 FA	669	DW RESET
FFFE: 40 FA	670	DW IRG

ENDASM

MONITOR ROM LISTING

```
1 *****  
2 *  
3 *      APPLE II  
4 *      SYSTEM MONITOR  
5 *  
6 *      COPYRIGHT 1977 BY  
7 *      APPLE COMPUTER, INC.  
8 *  
9 *      ALL RIGHTS RESERVED  
10 *  
11 *          S. WOZNIAK  
12 *          A. BAUM  
13 *  
14 *****  
15      TITLE      "APPLE II SYSTEM MONITOR"  
16 LCOU    EPZ $00  
17 LCC1    EPZ $01  
18 WNDLFT   EPZ $20  
19 WNDWDIN  EPZ $21  
20 WNDTOP   EPZ $22  
21 WNDBTM   EPZ $23  
22 CH     EPZ $24  
23 CV     EPZ $25  
24 GBASL   EPZ $26  
25 GBASH   EPZ $27  
26 BASL    EPZ $28  
27 BASH    EPZ $29  
28 BAS2L   EPZ $2A  
29 BAS2H   EPZ $2B  
30 H2     EPZ $2C  
31 LMNEM   EPZ $2C  
32 RTNL   EPZ $2C  
33 V2     EPZ $2D  
34 RMNEM   EPZ $2D  
35 RTNH   EPZ $2D  
36 MASK    EPZ $2E  
37 CKSUM   EPZ $2E  
38 FORMAT  EPZ $2E  
39 LASTIN  EPZ $2F  
40 LENGTH  EPZ $2F  
41 SIGN    EPZ $2F  
42 COLOR   EPZ $30  
43 MODE    EPZ $31  
44 INVFLG  EPZ $32  
45 PROMPT  EPZ $33  
46 YSAV   EPZ $34  
47 YSAVI  EPZ $35  
48 CSWL   EPZ $36  
49 CSWR   EPZ $37  
50 KSWL   EPZ $38  
51 KSWH   EPZ $39  
52 PCL    EPZ $3A  
53 PCH    EPZ $3B  
54 XQT    EPZ $3C  
55 AIL    EPZ $3C  
56 A1H    EPZ $3D  
57 A2L    EPZ $3E  
58 A2H    EPZ $3F  
59 A3L    EPZ $40  
60 A3H    EPZ $41  
61 A4L    EPZ $42  
62 A4H    EPZ $43  
63 A5L    EPZ $44  
64 A5H    EPZ $45  
65 ACC    EPZ $45  
66 XREG   EPZ $46  
67 YREG   EPZ $47  
68 STATUS  EPZ $48
```

69	SPNT	EPZ	\$49		
70	RNDL	EPZ	\$4E		
71	RNDH	EPZ	\$4F		
72	ACL	EPZ	\$50		
73	ACH	EPZ	\$51		
74	XTNDL	EPZ	\$52		
75	XTNDH	EPZ	\$53		
76	AUXL	EPZ	\$54		
77	AUXH	EPZ	\$55		
78	PICK	EPZ	\$95		
79	IN	EQU	\$0200		
80	USRADR	EQU	\$03F8		
81	NMI	EQU	\$03FB		
82	IRQLOC	EQU	\$03FE		
83	IOADR	EQU	\$C000		
84	KBD	EQU	\$C000		
85	KBDSTRB	EQU	\$C010		
86	TAPEOUT	EQU	\$C020		
87	SPKR	EQU	\$C030		
88	TXTCLR	EQU	\$C050		
89	TXTSET	EQU	\$C051		
90	MIXCLR	EQU	\$C052		
91	MIXSET	EQU	\$C053		
92	LOWSCR	EQU	\$C054		
93	HISCR	EQU	\$C055		
94	LORES	EQU	\$C056		
95	Hires	EQU	\$C057		
96	TAPEIN	EQU	\$C060		
97	PADDLO	EQU	\$C064		
98	PTRIG	EQU	\$C070		
99	BASIC	EQU	\$E000		
100	BASIC2	EQU	\$E003		
101		ORG	\$F890	ROM START ADDRESS	
F800:	4A	102	PLOT	Y-COORD/2	
F801:	0d	103	PHP	SAVE LSB IN CARRY	
F802:	20 47 F8	104	JSR GBASCALC	CALC BASE ADR IN GBASL,H	
F805:	28	105	PLP	RESTORE LSB FROM CARRY	
F806:	A9 0F	106	LDA #\$0F	MASK \$0F IF EVEN	
F808:	90 02	107	BCC RTMASK		
F80A:	69 E0	108	ADC #\$E0	MASK \$F0 IF ODD	
F80C:	85 2E	109	RTMASK		
F80E:	B1 26	110	RTPL1	STA MASK	
F810:	45 30	111	LDA (GBASL),Y	DATA	
F812:	25 2E	112	EOR COLOR	XOR COLOR	
F814:	51 26	113	AND MASK	AND MASK	
F816:	91 26	114	EOR (GBASL),Y	XOR DATA	
F818:	60	115	STA (GBASL),Y	TO DATA	
F819:	20 00 F8	116	RTS		
F81C:	C4 2C	117	HLINE1	RTS1	PLOT SQUARE
F81E:	B0 11	118	BCS RTSL	DONE?	
F820:	C8	119	INY	YES, RETURN	
F821:	20 0E F8	120	JSR PLOT1	NO, INCR INDEX (X-COORD)	
F824:	90 F6	121	BCC HLINE1	PLOT NEXT SQUARE	
F826:	69 01	122	VLINEZ	ALWAYS TAKEN	
F828:	48	123	ADC #\$01	NEXT Y-COORD	
F829:	20 00 F8	124	PHA	SAVE ON STACK	
F82C:	68	125	JSR PLOT	PLOT SQUARE	
F82D:	C5 2D	126	PLA	DONE?	
F82F:	90 F5	127	CMP V2	NO,LOOP.	
F831:	60	128	RTS1		
F832:	A0 2F	129	CLRSCR	RTS	MAX Y, FULL SCR CLR
F834:	D0 02	130	LDY #\$2F	ALWAYS TAKEN	
F836:	A0 27	131	BNE CLRSC2	MAX Y, TOP SCR CLR	
F838:	84 2D	132	CLRTOP	STORE AS BOTTOM COORD	
		133 *	STY V2	FOR VLINE CALLS	
F83A:	A0 27	134	LDY #\$27	RIGHTMOST X-COORD (COLUMN)	
F83C:	A9 00	135	CLRSC3	LDA #\$0	TOP COORD FOR VLINE CALLS
F83E:	85 30	136	STA COLOR	CLEAR COLOR (BLACK)	
F840:	20 28 F8	137	JSR VLINE	DRAW VLINE	
F843:	88	138	DEY	NEXT LEFTMOST X-COORD	
F844:	10 F6	139	BPL CLRSC3	LOOP UNTIL DONE.	
F846:	60	140	RTS		
F847:	48	141	GBASCALC	RTS	FOR INPUT 000DEFGH
F848:	4A	142	LSR A		

F849:	29	03	143		AND #\$03	
F84B:	09	04	144		ORA #\$04	GENERATE GBASH=000001FG
F84D:	85	27	145		STA GBASH	
F84F:	68		146		PLA	AND GBASL=HDEDE000
F850:	29	18	147		AND #\$18	
F852:	90	02	148		BCC GBCALC	
F854:	69	7F	149		ADC #\$7F	
F856:	85	26	150	GBCALC	STA GBASL	
F858:	0A		151		ASL A	
F859:	0A		152		ASL A	
F85A:	05	26	153		ORA GBASL	
F85C:	85	26	154		STA GBASL	
F85E:	60		155		RTS	
F85F:	A5	30	156	NXTCOL	LDA COLOR	INCREMENT COLOR BY 3
F861:	18		157		CLC	
F862:	69	03	158		ADC #\$03	
F864:	29	0F	159	SETCOL	AND #\$0F	SETS COLOR=17*A MOD 16
F866:	85	30	160		STA COLOR	
F868:	0A		161		ASL A	BOTH HALF BYTES OF COLOR EQUAL
F869:	0A		162		ASL A	
F86A:	0A		163		ASL A	
F86B:	0A		164		ASL A	
F86C:	05	30	165		CRA COLOR	
F86E:	85	30	166		STA COLOR	
F870:	60		167		RTS	
F871:	4A		168	SCRN	LSR A	READ SCREEN Y-COORD/2
F872:	08		169		PHP	SAVE LSB (CARRY)
F873:	20	47	F8	170	JSR GBASCALC	CALC BASE ADDRESS
F876:	B1	26	171		LDA (GBASL),Y	GET BYTE
F878:	28		172		PLP	RESTORE LSB FROM CARRY
F879:	90	04	173	SCRN2	BCC RTMSKZ	IF EVEN, USE LO H
F87B:	4A		174		LSR A	
F87C:	4A		175		LSR A	
F87D:	4A		176		LSR A	SHIFT HIGH HALF BYTE DOWN
F87E:	4A		177		LSR A	
F87F:	29	0F	178	RTMSKZ	AND #\$0F	MASK 4-BITS
F881:	60		179		RTS	
F882:	A6	3A	180	INSDS1	LDX PCL	PRINT PCL,H
F884:	A4	3B	181		LDY PCH	
F886:	20	96	FD	182	JSR PRYX2	
F889:	20	48	F9	183	JSR PRBLNK	FOLLOWED BY A BLANK
F88C:	A1	3A	184		LDA (PCL,X)	GET OP CODE
F88E:	A8		185	INSDS2	TAY	
F88F:	4A		186		LSR A	EVEN/ODD TEST
F890:	90	09	187		BCC IEVEN	
F892:	6A		188		ROR A	BIT 1 TEST
F893:	B0	10	189		BCS ERR	XXXXXXXX11 INVALID OP
F895:	C9	A2	190		CMP #\$A2	
F897:	F0	0C	191		BEQ ERR	OPCODE \$99 INVALID
F899:	29	37	192		AND #\$87	MASK BITS
F89B:	4A		193	IEVEN	LSR A	LSB INTO CARRY FOR L/R TEST
F89C:	AA		194		TAX	
F89D:	BD	62	F9	195	LDA FMT1,X	GET FORMAT INDEX BYTE
F8A0:	20	79	F8	196	JSR SCRN2	R/L H-BYTE ON CARRY
F8A3:	D0	04	197		BNE GETFMT	
F8A5:	A0	80	198	ERR	LDY #\$80	SUBSTITUTE \$80 FOR INVALID OPS
F8A7:	A9	00	199		LDA \$80	SET PRINT FORMAT INDEX TO 0
F8A9:	AA		200	GETFMT	TAX	
F8AA:	BD	A6	F9	201	LDA FMT2,X	INDEX INTO PRINT FORMAT TABLE
F8AD:	85	2E		202	STA FORMAT	SAVE FOR ADR FIELD FORMATTING
F8AF:	29	03	203		AND #\$03	MASK FOR 2-BIT LENGTH
			204	*	(P=1 BYTE, 1=2 BYTE, 2=3 BYTE)	
F8B1:	85	2F	205		STA LENGTH	
F8B3:	98		206		TYA	OPCODE
F8B4:	29	8F	207		AND #\$8F	MASK FOR XXXX1010 TEST
F8B6:	AA		208		TAX	SAVE IT
F8B7:	98		209		TYA	OPCODE TO A AGAIN
F8B8:	A0	03	210		LDY #\$03	
F8B9:	E0	8A	211		CIX #\$8A	
F8BC:	F0	0B	212		BEQ MNNDX3	
F8BE:	4A		213	MNNDX1	LSR A	
F8BF:	90	08	214		BCC MNNDX3	FORM INDEX INTO MNEMONIC TABLE
F8C1:	4A		215		LSR A	

F8C2:	4A	216	MNNDX2	LSR A		1) XXXX1010=>00101XXX
F8C3:	09 20	217		ORA #\$20		2) XXXYY01=>00111XXX
F8C5:	88	218		DEY		3) XXXYYY10=>00110XXX
F8C6:	D0 FA	219		BNE MNNDX2		4) XXXYY100=>00100XXX
F8C8:	Cd	220		INY		5) XXXXX000=>000XXXXX
F8C9:	88	221	MNNDX3	DEY		
F8CA:	D0 F2	222		BNE MNNDX1		
F8CC:	60	223		RTS		
F8CD:	FF FF FF	224		DFB \$FF,\$FF,\$FF		
F8D0:	20 82 F8	225	INSTDSP	JSR INSDSL	GEN FMT, LEN BYTES	
F8D3:	48	226		PHA	SAVE MNEMONIC TABLE INDEX	
F8D4:	B1 3A	227	PRNTOP	LDA (PCL),Y		
F8D6:	20 DA FD	228		JSR PRBYTE		
F8D9:	A2 01	229		LDX #\$01	PRINT 2 BLANKS	
F8DB:	20 4A F9	230	PRNTBL	JSR PRBL2		
F8DE:	C4 2F	231		CPY LENGTH	PRINT INST (1-3 BYTES)	
F8E0:	C8	232		INY	IN A 12 CHR FIELD	
F8E1:	90 F1	233		BCC PRNTOP		
F8E3:	A2 03	234		LDX #\$03	CHAR COUNT FOR MNEMONIC PRINT	
F8E5:	C0 04	235		CPY #\$04		
F8E7:	90 F2	236		BCC PRNTBL		
F8E9:	68	237		PLA	RECOVER MNEMONIC INDEX	
F8EA:	A8	238		TAY		
F8EB:	B9 C0 F9	239		LDA MNEML,Y		
F8EE:	85 2C	240		STA LMNEM		
F8F0:	B9 00 FA	241		LDA MNEMR,Y	FETCH 3-CHAR MNEMONIC	
F8F3:	85 2D	242		STA RMNEM	(PACKED IN 2-BYTES)	
F8F5:	A9 J0	243	PRMN1	LDA #\$00		
F8F7:	A0 05	244		LDY #\$05		
F8F9:	J6 2D	245	PRMN2	ASL RMNEM	SHIFT 5 BITS OF	
F8FB:	26 2C	246		ROL LMNEM	CHARACTER INTO A	
F8FD:	2A	247		ROL A	(CLEAR CARRY)	
F8FE:	88	248		DEY		
F8FF:	D0 F6	249		BNE PRMN2		
F901:	69 BF	250		ADC #\$BF	ADD "?" OFFSET	
F903:	20 ED FD	251		JSR COUT	OUTPUT A CHAR OF MNEM	
F906:	CA	252		DEX		
F907:	D0 EC	253		BNE PRMN1		
F909:	20 48 F9	254		JSR PRBLNK	OUTPUT 3 BLANKS	
F90C:	A4 2F	255		LDY LENGTH		
F90E:	A2 06	256		LDX #\$06	CNT FOR 6 FORMAT BITS	
F910:	E0 03	257	PRADR1	CPX #\$03		
F912:	F0 1C	258		BEQ PRADRS5	IF X=3 THEN ADDR.	
F914:	06 2E	259	PRADR2	ASL FORMAT		
F916:	90 0E	260		BCC PRADR3		
F918:	BD B3 F5	261		LDA CHAR1-1,X		
F91B:	20 ED FD	262		JSR COUT		
F91E:	BD B9 F9	263		LDA CHAR2-1,X		
F921:	F0 03	264		BEQ PRADRS3		
F923:	20 ED FD	265		JSR COUT		
F926:	CA	266	PRADRS3	DEX		
F927:	D0 E7	267		BNE PRADR1		
F929:	60	268		RTS		
F92A:	88	269	PRADRS4	DEY		
F92B:	30 C7	270		BMI PRADRS2		
F92D:	20 DA FD	271		JSR PRBYTE		
F930:	A5 2E	272	PRADRS5	LDA FORMAT		
F932:	C9 E8	273		CMP #SE8	HANDLE REL ADR MODE	
F934:	B1 3A	274		LDA (PCL),Y	SPECIAL (PRINT TARGET,	
F936:	90 F2	275		BCC PRADRS4	NOT OFFSET)	
F938:	20 56 F9	276	RELADR	JSR FCADJ3		
F93B:	AA	277		TAX	PCL,PCH+OFFSET+1 TO A,Y	
F93C:	E8	278		INX		
F93D:	D0 01	279		BNE PRNTYX	+1 TO Y,X	
F93F:	C8	280		INY		
F940:	98	281	PRNTYX	TYA		
F941:	20 DA FD	282	PRNTAX	JSR PRBYTE	OUTPUT TARGET ADR	
F944:	8A	283	PRNTX	TXA	OF BRANCH AND RETURN	
F945:	4C DA FD	284		JMP PRBYTE		
F946:	A2 03	285	PRBLNK	LDX #\$03	BLANK COUNT	
F94A:	A9 A0	286	PRBL2	LDA #\$A0	LOAD A SPACE	
F94C:	20 ED FD	287	PRBL3	JSR COUT	OUTPUT A BLANK	
F94F:	CA	288		DEX		

```

F950: DU P8    289      BNE PRBL2      LOOP UNTIL COUNT=0
F952: 60        290      RTS
F953: 3d        291      PCADJ       SEC          U=1-BYTE, l=2-BYTE,
F954: A5 2F    292      PCADJ2     LDA LENGTH   2=3-BYTE
F956: A4 3B    293      PCADJ3     LDY PCH
F958: AA        294      TAX
F959: 10 01    295      BPL PCADJ4    TEST DISPLACEMENT SIGN
F95B: 88        296      DEY          (FOR REL BRANCH)
F95C: 65 3A    297      PCADJ4    ADC PCL
F95E: 90 01    298      BCC RTS2    EXTEND NEG BY DECR PCH
F960: C6        299      INY          PCL+LENGTH(OR DISPL)+1 TO A
F961: 60        300      RTS2       CARRY INTO Y (PCH)
F962: 04 20 54 301 *    FMT1 BYTES: XXXXXXXY0 INSTRS
F965: 30 UD    305      FMT1       DFB $04,$20,$54,$
F967: 80 04 90 306      DFB $80,$04,$30,$
F96A: 03 22    307      DFB $54,$33,$0D,$
F96C: 54 33  UD 307      DFB $54,$33,$0D,$
F96F: 80 04    307      DFB $90,$04,$20,$
F971: 90 04 20 308      DFB $90,$04,$20,$
F974: 54 33    308      DFB $90,$04,$20,$
F976: UD 80 04 308      DFB $90,$04,$20,$
F979: 90 04    309      DFB $90,$04,$20,$
F97B: 20 54 3B 309      DFB $20,$54,$3B,$
F97E: UD 80    310      DFB $20,$54,$3B,$
F980: 04 90 00 310      DFB $04,$90,$00,$
F983: 22 44    311      DFB $04,$90,$00,$
F985: 33 UD  C8 311      DFB $33,$0D,$C8,$
F988: 44 00    312      DFB $44,$00,$04,$
F98A: 11 22 44 312      DFB $11,$22,$44,$
F98D: 33 UD    313      DFB $11,$22,$44,$
F98F: C8 44  A9 313      DFB $C8,$44,$A9,$
F992: 01 22    314      DFB $01,$22,$44,$
F994: 44 33  UD 314      DFB $44,$33,$0D,$
F997: 80 04    315      DFB $44,$33,$0D,$
F999: 90 01 22 315      DFB $90,$01,$22,$
F99C: 44 33    316      DFB $90,$01,$22,$
F99E: UD 80 04 316      DFB $0D,$80,$04,$
F9A1: 90        317      DFB $0D,$80,$04,$
F9A2: 26 31 87 317      DFB $26,$31,$87,$ZZXXXXY01 INSTR'S
F9A5: 9A        318      DFB $26,$31,$87,$ZZXXXXY01 INSTR'S
F9A6: 00        319      FMT2       DFB $00      ERR
F9A7: 21        320      DFB $21      IMM
F9A8: 81        321      DFB $81      Z-PAGE
F9A9: 82        322      DFB $82      ABS
F9AA: 00        323      DFB $00      IMPLIED
F9AB: 00        324      DFB $00      ACCUMULATOR
F9AC: 59        325      DFB $59      (ZPAG,X)
F9AD: 4D        326      DFB $4D      (ZPAG),Y
F9AE: 91        327      DFB $91      ZPAG,X
F9AF: 92        328      DFB $92      ABS,X
F9B0: 86        329      DFB $86      ABS,Y
F9B1: 4A        330      DFB $4A      (ABS)
F9B2: 85        331      DFB $85      ZPAG,Y
F9B3: 9D        332      DFB $9D      RELATIVE
F9B4: AC A9 AC
F9B7: A3 Ab A4 333 CHAR1   ASC ",),#($"
F9BA: D9 UD  D8
F9BD: A4 A4 00 334 CHAR2   DFB $D9,$00,$0D,$
F9BD: 335 *CHAR2: "Y",0,"XSS",0 MNEML IS OF FORM:
F9BD: 336 *      (A) XXXXX000
F9BD: 337 *      (B) XXXYY100
F9BD: 338 *      (C) 1XXX1010
F9BD: 339 *      (D) XXXYYY10
F9BD: 340 *      (E) XXXYYY01
F9BD: 341 *      (X=INDEX)
F9BD: 342 *
F9C0: 1C 8A 1C
F9C3: 23 5D 8B 343 MNEML   DFB $1C,$8A,$1C,$
F9C6: 1B A1 9D

```

F9C9:	BA 1D 23 344	DFB \$1B,S1A,\$9D,S
F9CC:	9D 68 1D	DFB \$9D,\$8B,\$1D,S
F9CF:	A1 00 29 345	DFB \$19,SAE,\$69,S
F9D2:	19 AE 69	DFB \$19,SAE,\$69,S
F9D5:	A8 19 23 346	DFB \$24,\$53,\$1B,S
F9D8:	24 53 1B	DFB \$19,SA1 (A) FORMAT ABOVE
F9DB:	23 24 53 347	DFB \$24,\$53,\$1B,S
F9DE:	19 A1 348	DFB \$19,SA1 (B) FORMAT
F9E0:	00 1A 5B	DFB \$00,S1A,\$5B,S
F9E3:	5B A5 69 349	DFB \$24,S24 (C) FORMAT
F9E6:	24 24 350	DFB \$7C,\$00 (D) FORMAT
F9E8:	AE AE A8	DFB \$AE,SAE,\$A8,S
F9EB:	AD 29 00 351	DFB \$AE,SAE,\$A8,S
F9EE:	7C 00 352	DFB \$7C,\$00 (E) FORMAT
F9F0:	15 9C 6D	DFB \$15,\$9C,\$6D,S
F9F3:	9C A5 69 353	DFB \$29,\$53 (F) FORMAT
F9F6:	29 53 354	DFB \$84,S13,\$34,S
F9F8:	84 13 34	DFB \$23,SA0 (G) FORMAT
F9FB:	11 A5 69 355	DFB \$84,S13,\$34,S
F9FE:	23 A0 356	DFB \$23,SA0 (H) FORMAT
FA00:	D8 62 5A	DFB \$D8,\$62,\$5A,S
FA03:	48 26 62 357 MNEMR	DFB \$94,\$88,\$54,S
FA06:	94 88 54	DFB \$68,S44,SE6,S
FA09:	44 Cd 54 358	DFB \$68,S44,SE6,S
FA0C:	68 44 E8	DFB \$68,S44,SE6,S
FA0F:	94 00 B4 359	DFB \$08,S84,\$74,S
FA12:	08 84 74	DFB \$74,S72 (B) FORMAT
FA15:	B4 28 6E 360	DFB \$74,S72 (C) FORMAT
FA18:	74 F4 CC	DFB \$74,S72 (D) FORMAT
FA1B:	4A 72 F2 361	DFB \$74,S72 (E) FORMAT
FA1E:	A4 8A 362	DFB \$A4,S8A (A) FORMAT
FA20:	00 AA A2	DFB \$00,SAA,\$A2,S
FA23:	A2 74 74 363	DFB \$74,S72 (B) FORMAT
FA26:	74 72 364	DFB \$74,S72 (C) FORMAT
FA28:	44 68 B2	DFB \$44,\$68,\$B2,S
FA2B:	32 B2 00 365	DFB \$22,\$00 (C) FORMAT
FA2E:	22 00 366	DFB \$1A,1A 26
FA30:	1A 1A 26	DFB \$1A,S1A,\$26,S
FA33:	26 72 72 367	DFB \$88,SC8 (D) FORMAT
FA36:	88 C8 368	DFB SC4,SCA,\$26,S
FA38:	C4 CA 26	DFB SA2,SCB (E) FORMAT
FA3B:	48 44 44 369	DFB SFF,SFF,SFF
FA3E:	A2 C8 370	JSR INSTDSP DISASSEMBLE ONE INST
FA40:	FF FF FF 371	AT (PCL,H)
FA43:	20 00 F8 372 STEP	STA RTNL ADJUST TO USER
FA46:	68 373	PLA STACK. SAVE
FA47:	B5 2C 374	STA RTNH RTN ADR.
FA49:	b6 375	LDX #\$08
FA4A:	85 2D 376	LDA INITBL-1,X INIT XEQ AREA
FA4C:	A2 08 377	STA XQT,X
FA4E:	BD 10 FB 378 XQINIT	DEX
FA51:	95 3C 379	BNE XQINIT
FA53:	CA 380	LDA (PCL,X) USER OPCODE BYTE
FA54:	D0 F8 381	BEQ XBRK SPECIAL IF BREAK
FA56:	A1 3A 382	LDY LENGTH LEN FROM DISASSEMBLY
FA58:	F0 42 383	CMP #\$20
FA5A:	A4 2F 384	BEQ XJSR HANDLE JSR, RTS, JMP,
FA5C:	C9 20 385	CMP #\$60 JMP (), RTI SPECIAL
FA5E:	F0 59 386	BEQ XRTS
FA60:	C9 60 387	CMP #\$4C
FA62:	F0 45 388	BEQ XJMP
FA64:	C9 4C 389	CMP #\$6C
FA66:	F0 5C 390	BEQ XJMPAT
FA68:	C9 6C 391	CMP #\$40
FA6A:	F0 59 392	BEQ XRTI
FA6C:	C9 4U 393	AND #\$1F
FA6E:	F0 35 394	EOR #\$14
FA70:	29 1P 395	CMP #\$04 COPY USER INST TO XEQ AREA
FA72:	49 14 396	BEQ XQ2 WITH TRAILING NOPs
FA74:	C9 04 397	LDA (PCL),Y CHANGE REL BRANCH
FA76:	F0 02 398	STA XQTNZ,Y DISP TO 4 FOR
FA78:	B1 3A 399 XQ1	
FA7A:	99 3C 00 400 XQ2	

FA7D: 88	401		DEY	JMP TO BRANCH OR
FA7E: 10 F8	402		BPL XQ1	NBRANCH FROM XEQ.
FA80: 20 3F FF	403		JSR RESTORE	RESTORE USER REG CONTENTS.
FA83: 4C 3C 00	404		JMP XQTNZ	XEQ USER OP FROM RAM
FA86: 85 45	405	IRQ	STA ACC	(RETURN TO NBRANCH)
FA88: 68	406		PLA	
FA89: 48	407		PHA	**IRQ HANDLER
FA8A: 0A	408		ASL A	
FA8B: 0A	409		ASL A	
FA8C: 0A	410		ASL A	
FA8D: 30 03	411		BMI BREAK	TEST FOR BREAK
FA8F: 6C FE 03	412		JMP (IRQLOC)	USER ROUTINE VECTOR IN RAM
FA92: 28	413	BREAK	PLP	
FA93: 20 4C FF	414		JSR SAV1	SAVE REG'S ON BREAK
FA96: 68	415		PLA	INCLUDING PC
FA97: 85 3A	416		STA PCL	
FA99: 68	417		PLA	
FA9A: 85 3B	418		STA PCH	
FA9C: 20 82 F8 419	XBRK		JSR INSDS1	PRINT USER PC.
FA9F: 20 DA FA	420		JSR RGDSPL	AND REG'S
FAA2: 4C 65 FF	421		JMP MON	GO TO MCNITOR
FAA5: 18	422	XRTI	CLC	
FAA6: 68	423		PLA	SIMULATE RTI BY EXPECTING
FAA7: 85 48	424		STA STATUS	STATUS FROM STACK, THEN RTS
FAA9: 68	425	XRTS	PLA	RTS SIMULATION
FAAA: 85 3A	426		STA PCL	EXTRACT PC FROM STACK
FAAC: 68	427		PLA	AND UPDATE PC BY 1 (LEN=0)
FAAD: 85 3B	428	PCINC2	STA PCH	
FAAF: A5 2F	429	PCINC3	LDA LENGTH	UPDATE PC BY LEN
FAB1: 20 56 F9	430		JSR PCADJ3	
FAB4: 84 3B	431		STY PCH	
FAB6: 18	432		CLC	
FAB7: 90 14	433		BCC NEWPCL	
FAB9: 18	434	XJSR	CLC	
FABA: 20 54 F9	435		JSR PCADJ2	UPDATE PC AND PUSH
FABD: AA	436		TAX	INTO STACK FOR
FABE: 98	437		TYA	JSR SIMULATE
FABF: 48	438		PHA	
FAC0: 8A	439		TXA	
FAC1: 48	440		PHA	
FAC2: A0 02	441		LDY #\$02	
FAC4: 18	442	XJMP	CLC	
FAC5: B1 3A	443	XJMPAT	LDA (PCL),Y	
FAC7: AA	444		TAX	LOAD PC FOR JMP,
FAC8: 88	445		DEY	(JMP) SIMULATE.
FAC9: B1 3A	446		LDA (PCL),Y	
FACB: 86 3B	447		STX PCH	
FACD: 85 3A	448	NEWPCL	STA PCL	
FACF: B0 F3	449		BCS XJMP	
FAD1: A5 2D	450	RTNJMP	LDA RTNH	
FAD3: 48	451		PHA	
FAD4: A5 2C	452		LDA RTNL	
FAD6: 48	453		PHA	
FAD7: 20 BE FD	454	RGDSP	JSR CROUT	DISPLAY USER REG
FADA: A9 45	455	RGDSP1	LDA #ACC	CONTENTS WITH
FADC: 85 40	456		STA A3L	LABELS
FADE: A9 00	457		LDA #ACC/256	
FAE0: 85 41	458		STA A3H	
FAE2: A2 FB	459		LDX #\$FB	
FAE4: A9 A0	460	RDSP1	LDA #\$A0	
FAE6: 20 ED FD	461		JSR COUT	
FAE9: BD 1E FA	462		LDA RTBL-\$FB,X	
FAEC: 20 ED FD	463		JSR COUT	
FAEF: A9 BD	464		LDA #\$BD	
FAF1: 20 ED FD	465		JSR COUT	
FAF4: B5 4A	466		LDA ACC+5,X	
FAF6: 20 DA FD	467		JSR PRBYTE	
FAF9: E8	468		INX	
FAFA: 30 E8	469		BMI RDSP1	
FAFC: 60	470		RTS	
FAFD: 18	471	BRANCH	CLC	BRANCH TAKEN,
FAFE: A0 01	472		LDY #\$01	ADD LEN+2 TO PC
FB00: B1 3A	473		LDA (PCL),Y	

FB02:	20 56 F9	474		JSR PCADJ3
FB05:	85 3A	475		STA PCL
FB07:	98	476		TYA
FB08:	38	477		SEC
FB09:	B0 A2	478		BCS PCINC2
FB0B:	20 4A FF	479	NBRNCH	JSR SAVE
FB0E:	38	480		SEC
FB0F:	B0 9E	481		BCS PCINC3
FB11:	EA	482	INITBL	NOP
FB12:	EA	483		NOP
FB13:	4C 0B FB	484		JMP NBRNCH
FB16:	4C FD FA	485		JMP BRANCH
FB19:	C1	486	RTBL	DFB SC1
FB1A:	D8	487		DFB SD8
FB1B:	D9	488		DFB SD9
FB1C:	DU	489		DFB SD0
FB1D:	D3	490		DFB SD3
FB1E:	AD 70 C0	491	PREAD	LDA PTRIG
FB21:	A0 00	492		LDY #\$00
FB23:	EA	493		NOP
FB24:	EA	494		NOP
FB25:	BD 64 C0	495	PREAD2	LDA PADDL0,X
FB28:	10 04	496		BPL RTS2D
FB2A:	Cd	497		INY
FB2B:	DU F8	498		BNE PREAD2
FB2D:	88	499		DEY
FB2E:	60	500	RTS2D	RTS
FB2F:	A9 00	501	INIT	LDA #\$00
FB31:	85 48	502		STA STATUS
FB33:	AD 56 C0	503		LDA LORES
FB36:	AD 54 C0	504		LDA LOWSCR
FB39:	AD 51 C0	505	SETTXT	LDA TXTSET
FB3C:	A9 00	506		LDA #\$500
FB3E:	F0 0B	507		BEQ SETWND
FB40:	AD 50 C0	508	SETGR	LDA TXTCLR
FB43:	AD 53 C0	509		LDA MIXSET
FB46:	20 36 F8	510		JSR CLRTOP
FB49:	A9 14	511		LDA #\$14
FB4B:	85 22	512	SETWND	STA WNDTOP
FB4D:	A9 00	513		LDA #\$00
FB4F:	85 20	514		STA WNDLFT
FB51:	A9 28	515		LDA #\$28
FB53:	65 21	516		STA WNDWDTH
FB55:	A9 18	517		LDA #\$18
FB57:	85 23	518		STA WNDBTM
FB59:	A9 17	519		LDA #\$17
FB5B:	65 25	520	TABV	STA CV
FB5D:	4C 22 FC	521		JMP VTAB
FB60:	20 A4 FB	522	MULPM	JSR MD1
FB63:	A0 10	523	MUL	LDY #\$10
FB65:	A5 50	524	MUL2	LDA ACL
FB67:	4A	525		LSR A
FB68:	90 0C	526		BCC MUL4
FB6A:	18	527		CLC
FB6B:	A2 FE	528		LDX #SFE
FB6D:	B5 54	529	MUL3	LDA XTNDL+2,X
FB6F:	75 56	530		ADC AUXL+2,X
FB71:	95 54	531		STA XTNDL+2,X
FB73:	E8	532		INX
FB74:	DU F7	533		BNE MUL3
FB76:	A2 03	534	MUL4	LDX #\$03
FB78:	76	535	MUL5	DFB #\$76
FB79:	50	536		DFB #\$50
FB7A:	CA	537		DEX
FB7B:	10 FB	538		BPL MUL5
FB7D:	88	539		DEY
FB7E:	DU E5	540		BNE MUL2
FB80:	60	541		RTS
FB81:	20 A4 FB	542	DIVPM	JSR MD1
FB84:	A0 10	543	DIV	LDY #\$10
FB86:	06 50	544	DIV2	ASL ACL
FB88:	26 51	545		ROL ACH
FB8A:	26 52	546		ROL XTND
				XTND/AUX

FBB8C:	26	53	547	ROL	XTNDH	TO AC.
FBB8E:	38		546	SEC		
FBB8F:	A5	52	549	LDA	XTNDL	
FB91:	E5	54	550	SBC	AUXL	MOD TO XTND.
FB93:	AA		551	TAX		
FB94:	A5	53	552	LDA	XTNDH	
FB96:	E5	55	553	SBC	AUXH	
FB98:	90	00	554	BCC	DIV3	
FB9A:	E6	52	555	STX	XTNDL	
FB9C:	85	53	556	STA	XTNDH	
FB9E:	E6	50	557	INC	ACL	
FBA0:	dd		558	DEY		
FBA1:	DU	E3	559	BNE	DIV2	
FBA3:	60		560	RTS		
FBA4:	AU	00	561	MD1	LDY #\$00	ABS VAL OF AC, AUX
FBA6:	84	2F	562	STY	SIGN	WITH RESULT SIGN
FBA8:	A2	54	563	LDX	#AUXL	IN LSB OF SIGN.
FBAA:	20	AF	FB	JSR	MD2	
FBAD:	A2	50	565	LDX	#ACL	
FBAF:	B5	01	566	MD2	LDA LOC1,X	X SPECIFIES AC OR AUX
FBB1:	10	0D	567	BPL	MDRTS	
FBB3:	38		568	SEC		
FBB4:	98		569	MD3	TYA	
FBB5:	F5	00	570	SBC	LOC0,X	COMPL SPECIFIED REG
FBB7:	95	00	571	STA	LOC0,X	IF NEG.
FBB9:	98		572	TYA		
FBBB:	F5	01	573	SBC	LOC1,X	
FBBC:	95	01	574	STA	LOC1,X	
FBBE:	E6	2F	575	INC	SIGN	
FBC0:	60		576	MDRTS	RTS	
FBC1:	48		577	BASCALC	PHA	CALC BASE ADR IN BASL,H
FBC2:	4A		578	LSR	A	FOR GIVEN LINE NO.
FBC3:	29	03	579	AND	#\$03	0<=LINE NO.<=\$17
FBC5:	09	04	580	ORA	#\$04	ARG=000ABCDE, GENERATE
FBC7:	85	29	581	STA	BASH	BASH=000001CD
FBC9:	68		582	PLA		AND
FBCA:	29	18	583	AND	#\$18	BASL=EABAB000
FBCC:	90	02	584	BCC	BSCLC2	
FBCE:	69	7F	585	ADC	#\$7F	
FBD0:	85	28	586	BSCLC2	STA	BASL
FBD2:	0A		587	ASL	A	
PBD3:	0A		588	ASL	A	
FBD4:	05	28	589	ORA	BASL	
FED6:	85	28	590	STA	BASL	
FBD8:	60		591	RTS		
FBD9:	C9	87	592	BELL1	CMP #\$87	BELL CHAR? (CNTRL-G)
FBDB:	DU	12	593	BNE	RTS2B	NO, RETURN
FBDD:	A9	40	594	LDA	#\$40	DELAY .01 SECONDS
FBDF:	20	A8	FC	595	JSR	WAIT
FBE2:	A0	C0	596	LDY	#\$C0	
FBE4:	A9	0C	597	BELL2	LDA #\$0C	TOGGLE SPEAKER AT
FBE6:	20	A8	FC	598	JSR	1 KHZ FOR .1 SEC.
FBE9:	AD	30	C0	599	LDA	SPKR
FBEC:	68		600	DEY		
FBED:	DU	F5	601	BNE	BELL2	
FBEF:	60		602	RTS2B	RTS	
FBF0:	A4	24	603	STOADV	LDY CH	CURSER H INDEX TO Y-REG
FBF2:	91	26	604	STA	(BASL),Y	STOR CHAR IN LINE
FBF4:	E6	24	605	ADVANCE	INC CH	INCREMENT CURSER H INDEX
FBF6:	A5	24	606	LDA	CH	(MOVE RIGHT)
FBF8:	C5	21	607	CMP	WNDWDTH	BEYOND WINDOW WIDTH?
FBFA:	B0	66	608	BCS	CR	YES CR TO NEXT LINE
FBFC:	60		609	RTSJ	RTS	NO, RETURN
FBFD:	C9	A0	610	VIDCUT	CMP #\$AU	CONTROL CHAR?
FBFF:	B0	EF	611	BCS	STOADV	NO, OUTPUT IT.
FC01:	A6		612	TAY		INVERSE VIDEO?
FC02:	10	EC	613	BPL	STOADV	YES, OUTPUT IT.
FC04:	C9	8D	614	CMP	#\$8D	CR?
FC06:	F0	5A	615	BEQ	CR	YES.
FC08:	C9	8A	616	CMP	#\$8A	LINE FEED?
FC0A:	F0	5A	617	BEQ	LF	IF SO, DO IT.
FC0C:	C9	88	618	CMP	#\$88	BACK SPACE? (CNTRL-H)
FC0E:	DU	C9	619	BNE	BELL1	NO, CHECK FOR BELL.

FC10:	C6 24	620	BS	DEC CH	DECREMENT CURSER H INDEX
FC12:	10 E8	621		BPL RTS3	IF POS, OK. ELSE MOVE UP
FC14:	A5 21	622		LDA WNDWDTH	SET CH TO WNDWDTH-1
FC16:	85 24	623		STA CH	
FC18:	C6 24	624		DEC CH	(RIGHTMOST SCREEN POS)
FC1A:	A5 22	625	UP	LDA WNDTOP	CURSER V INDEX
FC1C:	C5 25	626		CMP CV	
FC1E:	BU JB	627		BCS RTS4	IF TOP LINE THEN RETURN
FC20:	C6 25	628		DEC CV	DECR CURSER V-INDEX
FC22:	A5 25	629	VTAB	LDA CV	GET CURSER V-INDEX
FC24:	20 C1 FB	630	VTABZ	JSR BASCALC	GENERATE BASE ADDR
FC27:	65 20	631		ADC WNDLFT	ADD WINDOW LEFT INDEX
FC29:	85 28	632		STA BASL	TO BASL
FC2B:	60	633	RTS4	RTS	
FC2C:	49 C0	634	ESC1	EGR #SCU	ESC?
FC2E:	F0 28	635		BEQ HOME	IF SO, DO HOME AND CLEAR
FC30:	69 FD	636		ADC #\$FD	ESC-A OR B CHECK
FC32:	90 C0	637		BCC ADVANCE	A, ADVANCE
FC34:	F0 DA	638		BEQ BS	B, BACKSPACE
FC36:	69 FD	639		ADC #\$FD	ESC-C OR D CHECK
FC38:	90 2C	640		BCC LF	C, DOWN
FC3A:	F0 DE	641		BEQ UP	D, GO UP
FC3C:	69 FD	642		ADC #\$FD	ESC-E OR F CHECK
FC3E:	90 5C	643		BCC CLREOL	E, CLEAR TO END OF LINE
FC40:	D0 E9	644		BNE RTS4	NOT F, RETURN
FC42:	A4 24	645	CLREOP	LDY CH	CURSOR H TO Y INDEX
FC44:	A5 25	646		LDA CV	CURSOR V TO A-REGISTER
FC46:	48	647	CLEOP1	PHA	SAVE CURRENT LINE ON STK
FC47:	20 24 FC	648		JSR VTABZ	CALC BASE ADDRESS
FC4A:	20 9E FC	649		JSR CLEOLZ	CLEAR TO EOL, SET CARRY
FC4D:	A0 00	650		LDY #\$00	CLEAR FROM H INDEX=0 FOR REST
FC4F:	68	651		PLA	INCREMENT CURRENT LINE
FC50:	69 00	652		ADC #\$00	(CARRY IS SET)
FC52:	C5 23	653		CMP WNDBTM	DONE TO BOTTOM OF WINDOW?
FC54:	90 FD	654		BCC CLEOP1	NO, KEEP CLEARING LINES
FC56:	B0 CA	655		BCS VTAB	YES, TAB TO CURRENT LINE
FC58:	A5 22	656	HOME	LDA WNDTOP	INIT CURSOR V
FC5A:	85 25	657		STA CV	AND H-INDICES
FC5C:	A0 J0	658		LDY #\$00	
FC5E:	64 24	659		STY CH	THEN CLEAR TO END OF PAGE
FC60:	F0 E4	660		BEQ CLEOP1	
FC62:	A9 J0	661	CR	LDA #\$00	CURSOR TO LEFT OF INDEX
FC64:	85 24	662		STA CH	(RET CURSOR H=0)
FC66:	E6 25	663	LF	INC CV	INCR CURSOR V(DOWN 1 LINE)
FC68:	A5 25	664		LDA CV	
FC6A:	C5 23	665		CMP WNDBTM	OFF SCREEN?
FC6C:	90 B6	666		BCC VTABZ	NO, SET BASE ADDR
FC6E:	C6 25	667		DEC CV	DECR CURSOR V(BACK TO BOTTOM)
FC70:	A5 22	668	SCROLL	LDA WNDTOP	START AT TOP OF SCRL WNDW
FC72:	48	669		PHA	
FC73:	20 24 FC	670		JSR VTABZ	GENERATE BASE ADDRESS
FC76:	A5 28	671	SCRL1	LDA BASL	COPY BASL,H
FC78:	85 2A	672		STA BAS2L	TO BAS2L,H
FC7A:	A5 29	673		LDA BASH	
FC7C:	85 2B	674		STA BAS2H	
FC7E:	A4 21	675		LDY WNDWDTW	INIT Y TO RIGHTMOST INDEX
FC80:	68	676		DEY	OF SCROLLING WINDOW
FC81:	68	677		PLA	
FC82:	69 01	678		ADC #\$01	INCR LINE NUMBER
FC84:	C5 23	679		CMP WNDBTM	DONE?
FC86:	B0 0D	680		BCS SCRL3	YES, FINISH
FC88:	48	681		PHA	
FC89:	20 24 FC	682		JSR VTABZ	FORM BASL,H (BASE ADDR)
FC8C:	B1 28	683	SCRL2	LDA (BASL),Y	MOVE A CHR UP ON LINE
FC8E:	91 2A	684		STA (BAS2L),Y	
FC90:	88	685		DEY	NEXT CHAR OF LINE
FC91:	10 F9	686		BPL SCRL2	
FC93:	30 E1	687		BMI SCRL1	NEXT LINE
FC95:	A0 00	688	SCRL3	LDY #\$00	CLEAR BOTTOM LINE
FC97:	20 9E FC	689		JSR CLEOLZ	GET BASE ADDR FOR BOTTOM LINE
FC9A:	B0 86	690		BCS VTAB	CARRY IS SET
FC9C:	A4 24	691	CLREOL	LDY CH	CURSOR H INDEX
FC9E:	A9 A0	692	CLEOLZ	LDA #SA0	

FCA0:	91 28	693	CLEOL2	STA (BASL),Y	STORE BLANKS FROM 'HERE'
FCA2:	C8	694		INY	TO END OF LINES (WNDWDTH)
FCA3:	C4 21	695		CPY WNDWDTH	
FCA5:	90 F9	696		BCC CLEOL2	
FCA7:	60	697		RTS	
FCA8:	38	698	WAIT	SEC	
FCA9:	40	699	WAIT2	PHA	
FCAA:	E9 01	700	WAIT3	SBC #\$01	
FCAC:	D0 FC	701		BNE WAIT3	1.0204 USEC
FCAE:	68	702		PLA	(13+2712*A+512*A*A)
FCAF:	E9 01	703		SBC #\$01	
FCB1:	D0 F6	704		BNE WAIT2	
FCB3:	60	705		RTS	
FCB4:	E6 42	706	NXTA4	INC A4L	INCR 2-BYTE A4
FCB6:	D0 02	707		BNE NXTA1	AND A1
FCB8:	E6 43	708		INC A4H	
FCBA:	A5 3C	709	NXTA1	LDA A1L	INCR 2-BYTE A1.
FCBC:	C5 3E	710		CMP A2L	
FCBE:	A5 3D	711		LDA A1H	AND COMPARE TO A2
FCC0:	E5 3F	712		SBC A2H	
FCC2:	E6 3C	713		INC A1L	(CARRY SET IF >=)
FCC4:	D0 02	714		BNE RTS4B	
FCC6:	E6 3D	715		INC A1H	
FCC8:	60	716	RTS4B	RTS	
FCC9:	A0 4B	717	HEADR	LDY #\$4B	WRITE A*256 'LONG 1'
FCCB:	20 DB PC	718		JSR ZERDLY	HALF CYCLES
FCCE:	DU F9	719		BNE HEADR	(650 USEC EACH)
FCD0:	69 FE	720		ADC #\$FE	
FCD2:	B0 F5	721		BCS HEADR	THEN A 'SHORT 0'
FCD4:	A0 21	722		LDY #\$21	(400 USEC)
FCD6:	20 DB FC	723	WRBIT	JSR ZERDLY	WRITE TWO HALF CYCLES
FCD9:	C8	724		INY	OF 250 USEC ('0')
FCDA:	C8	725		INY	OR 500 USEC ('0')
FCDB:	88	726	ZERDLY	DEY	
FCDC:	D0 FD	727		BNE ZERDLY	
FCDE:	90 05	728		BCC WRTAPE	Y IS COUNT FOR
FCE0:	A0 32	729		LDY #\$32	TIMING LOOP
FCE2:	88	730	ONEDLY	DEY	
FCE3:	D0 FD	731		BNE ONEDLY	
FCE5:	AC 20 CU	732	WRTAPE	LDY TAPEOUT	
FCE8:	A0 2C	733		LDY #\$2C	
FCEA:	CA	734		DEX	
FCEB:	60	735		RTS	
FCEC:	A2 08	736	RDBYTE	LDX #\$08	8 BITS TO READ
FCEE:	48	737	RDBYT2	PHA	READ TWO TRANSITIONS
FCEF:	20 FA PC	738		JSR RD2BIT	(FIND EDGE)
FCF2:	68	739		PLA	
FCF3:	2A	740		ROL A	NEXT BIT
FCF4:	A0 3A	741		LDY #\$3A	COUNT FOR SAMPLES
FCF6:	CA	742		DEX	
FCF7:	D0 F5	743		BNE RDBYT2	
FCF9:	60	744		RTS	
FCFA:	20 FD PC	745	RD2BIT	JSR RDBIT	
FCFD:	68	746	RDBIT	DEY	DECR Y UNTIL
FCFE:	AD 60 CU	747		LDA TAPEIN	TAPE TRANSITION
FD01:	45 2F	748		EOR LASTIN	
FD03:	10 F8	749		BPL RDBIT	
FD05:	45 2F	750		EOR LASTIN	
FD07:	85 2F	751		STA LASTIN	
FD09:	C0 80	752		CPY #\$80	SET CARRY ON Y-REG.
FD0B:	60	753		RTS	
FD0C:	A4 24	754	RDKEY	LDY CH	
FD0E:	B1 2d	755		LDA (BASL),Y	SET SCREEN TO FLASH
FD10:	48	756		PHA	
FD11:	29 3F	757		AND #\$3F	
FD13:	09 40	758		ORA #\$40	
FD15:	91 20	759		STA (BASL),Y	
FD17:	68	760		PLA	
FD16:	6C 38 00	761		JMP (KSWL)	GO TO USER KEY-IN
FD18:	E6 4E	762	KEYIN	INC RNDL	
FD1D:	D0 02	763		BNE KEYIN2	INCR RND NUMBER
FD1F:	E6 4F	764		INC RNDH	
FD21:	2C 00 CU	765	KEYIN2	BIT KBD	KEY DOWN?

FD24:	10 F5	766	BPL	KEYIN	LOOP
FD26:	91 26	767	STA	(BASL),Y	REFLACE FLASHING SCREEN
FD28:	AD 00	C0 768	LDA	KBD	GET KEYCODE
FD2B:	2C 10	C0 769	BIT	KBDSTRB	CLR KEY STROBE
FD2E:	60	770	RTS		
FD2F:	20 UC	FD 771	JSR	RDKEY	GET KEYCODE
FD32:	20 2C	FC 772	JSR	ESC1	HANDLE ESC FUNC.
FD35:	20 UC	FD 773	JSR	RDKEY	READ KEY
FD38:	C9 98	774	CMP	#\$98	ESC?
FD3A:	F0 F3	775	BEQ	ESC	YES, DON'T RETURN
FD3C:	60	776	RTS		
FD3D:	A5 32	777	NOTCR	LDA	INVFLG
FD3F:	48	778	PHA		
FD40:	A9 FF	779	LDA	#\$FF	
FD42:	85 32	780	STA	INVFLG	ECHO USER LINE
FD44:	BD 00	02 781	LDA	IN,X	NON INVERSE
FD47:	20 ED	FD 782	JSR	COUT	
FD4A:	68	783	PLA		
FD4B:	85 32	784	STA	INVFLG	
FD4D:	BD 00	02 785	LDA	IN,X	
FD4F:	C9 88	786	CMP	#\$88	CHECK FOR EDIT KEYS
FD52:	F0 1D	787	BEQ	BCKSPC	BS, CTRL-X.
FD54:	C9 98	788	CMP	#\$98	
FD56:	F0 JA	789	BEQ	CANCEL	
FD58:	E0 F8	790	CFX	#\$F8	MARGIN?
FD5A:	90 U3	791	BCC	NOTCR1	
FD5C:	20 3A	FF 792	JSR	BELL	YES, SOUND BELL
FD5F:	Ed	793	NOTCR1	INX	ADVANCE INPUT INDEX
FD60:	DU 13	794	BNE	NXTCHAR	
FD62:	A9 DC	795	CANCEL	LDA	#\$DC
FD64:	20 ED	FD 796	JSR	COUT	BACKSLASH AFTER CANCELLED LTN
FD67:	20 BE	FD 797	GETLNZ	JSR	CROUT
FD6A:	A5 33	798	GETLN	LDA	PROMPT
FD6C:	20 ED	FD 799	JSR	COUT	OUTPUT PRMPT CHAR
FD6F:	A2 U1	800	LDX	#\$U1	INIT INPUT INDEX
FD71:	8A	801	BCKSPC	TXA	WILL BACKSPACE TO U
FD72:	F0 F3	802	BEQ	GETLNZ	
FD74:	CA	803	DEX		
FD75:	20 35	FD 804	NXTCHAR	JSR	RDCHAR
FD78:	C9 95	805	CMP	#PICK	USE SCREEN CHAR
FD7A:	DU U2	806	BNE	CAPTST	FOR CTRL-U
FD7C:	B1 28	807	LDA	(BASL),Y	
FD7E:	C9 E0	808	CAPTST	CMP	#\$EU
FD80:	90 U2	809	BCC	ADDINP	CONVERT TO CAPS
FD82:	29 DF	810	AND	#\$DF	
FD84:	90 U0	U2 811	ADDINP	STA	IN,X
FD87:	C9 3D	812	CMP	#\$8D	ADD TO INPUT BUF
FD89:	DU B2	813	BNE	NOTCR	
FD8B:	20 9C	FC 814	JSR	CLREOL	CLR TO EOL IF CR
FD8E:	A9 8D	815	CROUT	LDA	#\$BD
FD90:	DU 5B	816	BNE	COUT	
FD92:	A4 3D	817	PRA1	LDY	A1H
FD94:	A6 3C	818		LDX	A1L
FD96:	20 BE	FD 819	PRYX2	JSR	CROUT
FD99:	20 40	F9 820		JSR	PRNTYX
FD9C:	A0 U0	821		LDY	#\$U0
FD9E:	A9 AD	822		LDA	#\$AD
FDA0:	4C ED	FD 823		JMP	COUT
FDA3:	A5 3C	824	XAM8	LDA	A1L
FDA5:	09 U1	825		ORA	#\$07
FDA7:	85 3E	826		STA	A2L
FDA9:	A5 3D	827		LDA	A1H
FDAB:	85 JF	828		STA	A2H
FDAD:	A5 3C	829	MDSCHK	LDA	A1L
FCAF:	29 U7	830		AND	#\$U7
FDB1:	DJ 03	831		BNE	DATAOUT
FDB3:	20 92	FD 832	XAM	JSR	PRA1
FDB6:	A9 AU	833	DATAOUT	LDA	#\$AD
FDB8:	20 ED	FD 834		JSR	COUT
FDBB:	B1 3C	835		LDA	(A1L),Y
FDBD:	20 DA	FD 836		JSR	FRBYTE
FDC0:	20 BA	FC 837		JSR	NXTAI

FDC3:	90 E6	838		BCC	MOD8CHK	CHECK IF TIME TO,
FDC5:	60	839	RTS 4C	RTS		PRINT ADDR
FDC6:	4A	840	XAMPM	LSR	A	DETERMINE IF MCN
FDC7:	90 EA	841		BCC	XAM	MODE IS XAM
FDC9:	4A	842		LSR	A	ADD, OR SUB
FDCA:	4A	843		LSR	A	
FDCB:	A5 3E	844		LDA	A2L	
FDCD:	90 02	845		BCC	ADD	
FDCF:	49 FF	846		EOR	#\$FF	SUB: FORM 2'S COMPLEMENT
FDD1:	65 3C	847	ADD	ADC	AIL	
FDD3:	48	848		PHA		
FDD4:	A9 BD	849		LDA	#\$BD	
FDD6:	20 ED FD	850		JSR	COUT	PRINT '=', THEN RESULT
FDD9:	68	851		PLA		
FDDA:	48	852	PRBYTE	PHA		PRINT BYTE AS 2 HEX
FDDB:	4A	853		LSR	A	DIGITS, DESTROYS A-REG
FDDC:	4A	854		LSR	A	
FDDD:	4A	855		LSR	A	
FDDE:	4A	856		LSR	A	
FDDF:	20 E5 FD	857		JSR	PRHEXZ	
FDE2:	68	858		PLA		
FDE3:	29 0F	859	PRHEX	AND	#\$0F	PRINT HEX DIG IN A-REG
FDE5:	09 30	860	PRHEXZ	CRA	#\$B0	LSB'S
FCE7:	C9 BA	861		CMP	#\$BA	
FDE9:	90 02	862		BCC	COUT	
FDEB:	69 06	863		ADC	#\$06	
FDED:	6C 58 00	864	COUT	JMP	(CSWL)	VECTOR TO USER OUTPUT ROUTINE
FDF0:	C9 A0	865	COUT1	CMP	#\$A0	
FDF2:	90 J2	866		BCC	COUTZ	DON'T OUTPUT CTRL'S INVERSE
FDF4:	25 32	867		AND	INVFLG	MASK WITH INVERSE FLAG
FDF6:	84 35	868	COUTZ	STY	YSAV1	SAV Y-REG
FDF8:	48	869		PHA		SAV A-REG
FDF9:	2J FD FB	870		JSR	VIDOUT	CUTPUT A-REG AS ASCII
FDFFC:	68	871		PLA		RESTORE A-REG
FCFD:	A4 35	872		LDY	YSAV1	AND Y-REG
FDFF:	60	873		RTS		THEN RETURN
FE00:	C6 34	874	BL1	DEC	YSAV	
FE02:	F0 9F	875		BEQ	XAM8	
FE04:	CA	876	BLANK	DEX		BLANK TO MCN
FE05:	DU 16	877		BNE	SETMDZ	AFTER BLANK
FE07:	C9 BA	878		CMP	#\$BA	DATA STORE MODE?
FE09:	DU BB	879		BNE	XAMPM	NO, XAM, ADD OR SUB
FE0B:	85 31	880	STOR	STA	MODE	KEEP IN STORE MODE
FE0D:	A5 3E	881		LDA	A2L	
FE0F:	91 40	882		STA	(A3L),Y	STORE AS LOW BYTE AS (A3)
FE11:	E6 40	883		INC	A3L	
FE13:	DU 02	884		BNE	RTS5	INCR A3, RETURN
FE15:	E6 41	885		INC	A3H	
FE17:	60	886	RTS5	RTS		
FE18:	A4 34	887	SETMODE	LDY	YSAV	SAVE CONVERTED ':', '+',
FE1A:	B9 FF 01	888		LDA	IN-1,Y	'-', '.' AS MODE.
FE1D:	85 31	889	SETMDZ	STA	MODE	
FE1F:	60	890		RTS		
FE20:	A2 01	891	LT	LDX	#\$01	
FE22:	B5 3E	892	LT2	LDA	A2L,X	COPY A2 (2 BYTES) TO
FE24:	95 42	893		STA	A4L,X	A4 AND A5
FE26:	95 44	894		STA	A5L,X	
FE28:	CA	895		DEX		
FE29:	10 F7	896		BPL	LT2	
FE2B:	60	897		RTS		
FE2C:	B1 3C	898	MOVE	LDA	(A1L),Y	MOVE (A1 TO A2) TO
FE2E:	91 42	899		STA	(A4L),Y	(A4)
FE30:	20 B4 FC	900		JSR	NXTA4	
FE33:	90 F7	901		BCC	MOVE	
FE35:	60	902		RTS		
FE36:	B1 3C	903	VFY	LDA	(A1L),Y	VERIFY (A1 TO A2) WITH
FE38:	D1 42	904		CMP	(A4L),Y	(A4)
FE3A:	F0 1C	905		BEQ	VFYOK	
FE3C:	20 92 FD	906		JSR	PRA1	
FE3F:	B1 3C	907		LDA	(A1L),Y	
FE41:	20 DA FD	908		JSR	PRBYTE	
FE44:	A9 A0	909		LDA	#\$AU	
FE46:	20 ED FD	910		JSR	COUT	

FE49:	A9 A8	911	LDA #SA8	
FE4B:	20 ED FD	912	JSR COUT	
FE4C:	B1 42	913	LDA (A4L),Y	
FE50:	20 DA FD	914	JSR PRBYTE	
FE53:	A9 A9	915	LDA #SA9	
FE55:	20 ED FD	916	JSR COUT	
FE58:	20 34 FC	917	VFYOK JSR NXTA4	
FE5B:	90 D9	918	BCC VFY	
FE5D:	60	919	RTS	
FE5E:	20 75 FE	920	LIST JSR AIPC	MOVE A1 (2 BYTES) TO
FE61:	A9 14	921	LDA #S14	PC IF SPEC'D AND
FE63:	48	922	PHA LIST2	DISSEMBLE 20 INSTRS
FE64:	20 D0 F8	923	JSR INSTDSP	
FE67:	20 53 F9	924	JSR FCADJ	ADJUST PC EACH INSTR
FE6A:	65 3A	925	STA PCL	
FE6C:	64 38	926	STY PCH	
FE6E:	68	927	PLA	
FE6F:	38	928	SEC	
FE70:	E9 01	929	SBC #S01	NEXT OF 20 INSTRS
FE72:	D0 EF	930	BNE LIST2	
FE74:	60	931	RTS	
FE75:	6A	932	AIPC TXA	
FE76:	F0 07	933	BEQ AIPCRTS	IF USER SPEC'D ADR COPY FROM A1 TO PC
FE78:	B5 3C	934	AIPCLP LDA A1L,X	
FE7A:	95 3A	935	STA PCL,X	
FE7C:	CA	936	OEX	
FE7D:	10 F9	937	BPL AIPCLP	
FE7F:	60	938	AIPCRTS RTS	
FE80:	A0 3F	939	SETINV LDY #\$3F	SET FOR INVERSE VID
FE82:	D0 02	940	BNE SETIFLG	VIA COUT1
FE84:	A0 FF	941	SETNORM LDY #\$FF	SET FOR NORMAL VID
FE86:	84 32	942	SETIFLG STY INVFLG	
FE88:	60	943	RTS	
FE89:	A9 00	944	SETKBD LD A#\$00	SIMULATE PORT #0 INPUT
FE8B:	85 3E	945	IMPORT STA A2L	SPECIFIED (KEYIN ROUTINE)
FE8D:	A2 38	946	INPRT LDX #KSWL	
FE8F:	A0 18	947	LDY #KEYIN	
FE91:	D0 08	948	BNE IOPRT	
FE93:	A9 00	949	SETVID LD A#\$00	SIMULATE PORT #0 OUTPUT
FE95:	85 3E	950	OUTPORT STA A2L	SPECIFIED (COUT1 ROUTINE)
FE97:	A2 36	951	OUTPRT LDX #CSWL	
FE99:	A0 F0	952	LDY #COUT1	
FE9B:	A5 3E	953	IOPRT LDA A2L	SET RAM IN/OUT VECTORS
FE9D:	29 0F	954	AND #\$0F	
FE9F:	F0 06	955	BEQ IOPRT1	
FEA1:	09 C0	956	ORA #IOADR/256	
FEA3:	A0 00	957	LDY #S00	
FEA5:	F0 02	958	BEQ IOPRT2	
FEA7:	A9 FD	959	IOPRT1 LDA #COUT1/256	
FEA9:	94 00	960	IOPRT2 STY LCCU,X	
FEAB:	95 01	961	STA LOCI,X	
FEAD:	60	962	RTS	
FEAE:	EA	963	NOP	
FEAF:	EA	964	NOP	
FEB0:	4C 00 E0	965	XBASIC JMP BASIC	TO BASIC WITH SCRATCH
FEB3:	4C 03 E0	966	BASCONT JMP BASIC2	CONTINUE BASIC
FEB6:	20 75 FE	967	GO JSR AIPC	ADR TO PC IF SPEC'D
FEB9:	20 3F FF	968	JSR RESTORE	RESTORE META REGS
FEBC:	6C 3A 00	969	JMP (PCL)	GO TO USER SUBR
FEBF:	4C D7 FA	970	REGZ JMP REGDSP	TO REG DISPLAY
FEC2:	Cb 34	971	TRACE DEC YSAV	
FEC4:	20 75 FE	972	STEP2 JSR AIPC	ADR TO PC IF SPEC'D
FEC7:	4C 43 FA	973	JMP STEP	TAKE ONE STEP
FECA:	4C F8 0J	974	USR JMP USRADR	TO USR SUBR AT USRADR
FECD:	A9 40	975	WRITE LDA #S40	
FECF:	20 C9 FC	976	JSR HEADR	WRITE 10-SEC HEADER
FED2:	A0 27	977	LDY #\$27	
FED4:	A2 00	978	WRI LDX #S00	
FED6:	41 3C	979	EOR (A1L,X)	
FED8:	48	980	PHA	
FED9:	A1 3C	981	LDA (A1L,X)	

FEDB:	20 ED FE 992		JSR WRBYTE	
FEDE:	20 BA FC 983		JSR NXTA1	
FEF1:	A0 1D 984		LDY #\$1D	
FEF3:	68 985		PLA	
FEF4:	90 EE 986		BCC WR1	
FEF6:	A0 22 987		LDY #S22	
FEF8:	20 ED FE 988		JSR WRBYTE	
FEFB:	F0 4D 989		BEQ BELL	
FEFD:	A2 10 990	WRBYTE	LDX #\$10	
FEFF:	JA 991	WRBYT2	ASL A	
FEFO:	20 D6 FC 992		JSR WRBIT	
FEF3:	D0 FA 993		BNE WRBYT2	
FEF5:	60 994		RTS	
FEF6:	20 J0 FE 995	CRMON	JSR BL1	HANDLE CR AS BLANK
FEF9:	68 996		PLA	THEN POP STACK
FEFA:	68 997		PLA	AND RTN TO MON
FEFB:	D0 6C 998		BNE MONZ	
FEFD:	20 FA FC 999	READ	JSR RD2BIT	FIND TAPEIN EDGE
FF00:	A9 16 1000		LDA #\$16	
FF02:	20 C9 FC 1001		JSR HEADR	DELAY 1.5 SECONDS
FF05:	85 2E 1002		STA CHKSUM	INIT CHKSUM=\$FF
FF07:	20 FA FC 1003		JSR RD2BIT	FIND TAPEIN EDGE
FF0A:	A0 24 1004	RD2	LDY #\$24	LOOK FOR SYNC BIT
FF0C:	20 FD FC 1005		JSR RDBIT	(SHORT 0)
FF0F:	B0 F9 1006		BCS RD2	LOOP UNTIL FOUND
FF11:	20 FD FC 1007		JSR RDBIT	SKIP SECOND SYNC H-CYCLE
FF14:	A0 3B 1008		LDY #\$3B	INDEX FOR 0/1 TEST
FF16:	20 EC FC 1009	RD3	JSR RDBYTE	READ A BYTE
FF19:	81 3C 1010		STA (A1,X)	STORE AT (A1)
FF1B:	45 2E 1011		EOR CHKSUM	
FF1D:	85 2E 1012		STA CHKSUM	UPDATE RUNNING CHKSUM
FF1F:	20 BA FC 1013		JSR NXTA1	INCR A1, COMPARE TO A2
FF22:	A0 35 1014		LDY #\$35	COMPENSATE 0/1 INDEX
FF24:	90 F0 1015		BCC RD3	LOOP UNTIL DONE
FF26:	20 EC FC 1016		JSR RDBYTE	READ CHKSUM BYTE
FF29:	C5 2E 1017		CMP CHKSUM	
FF2B:	F0 UD 1018		BEQ BELL	GOOD, SOUND BELL AND RETURN
FF2D:	A9 C5 1019	PRERR	LDA #\$C5	
FF2F:	20 ED FD 1020		JSR COUT	PRINT "ERR", THEN BELL
FF32:	A9 D2 1021		LDA #\$D2	
FF34:	20 ED FD 1022		JSR COUT	
FF37:	20 ED FD 1023		JSR COUT	
FF3A:	A9 37 1024	BELL	LDA #\$87	OUTPUT BELL AND RETURN
FF3C:	4C ED FD 1025		JMP COUT	
FF3F:	A5 48 1026	RESTORE	LDA STATUS	RESTORE 6502 REG CCNTENTS
FF41:	48 1027		PHA	USED BY DEBUG SOFTWARE
FF42:	A5 45 1028		LDA ACC	
FF44:	A6 46 1029	RESTRI	LDX XREG	
FF46:	A4 47 1030		LDY YREG	
FF48:	20 1031		PLP	
FF49:	60 1032		RTS	
FF4A:	85 45 1033	SAVE	STA ACC	SAVE 6502 REG CONTENTS
FF4C:	86 46 1034	SAV1	STX XREG	
FF4E:	84 47 1035		STY YREG	
FF50:	08 1036		PHP	
FF51:	60 1037		PLA	
FF52:	85 48 1038		STA STATUS	
FF54:	BA 1039		TSX	
FF55:	80 49 1040		STX SPNT	
FF57:	D8 1041		CLD	
FF58:	60 1042		RTS	
FF59:	20 84 FE 1043	RESET	JSR SETNORM	SET SCREEN MODE
FF5C:	20 2F FB 1044		JSR INIT	AND INIT KBD/SCREEN
FF5F:	20 93 FE 1045		JSR SETVID	AS I/O DEV'S
FF62:	20 89 FE 1046		JSR SETKBD	
FF65:	D8 1047	MON	CLD	MUST SET HEX MODE!
FF66:	20 JA FF 1048		JSR BELL	
FF69:	A9 AA 1049	MONZ	LDA #\$AA	**' PROMPT FOR MON
FF6B:	85 33 1050		STA PROMPT	
FF6D:	20 57 FD 1051		JSR GETLNZ	READ A LINE
FF70:	20 C7 FF 1052		JSR ZMODE	CLEAR MON MODE, SCAN IDX
FF73:	20 A7 FF 1053	NXTITM	JSR GETNUM	GET ITEM, NON-HEX
FF76:	84 34 1054		STY YSAV	CHAR IN A-REG

FF78:	A0 17	1055	LDY #S17	X-REG=0 IF NO HEX INPUT
FF7A:	BB	1056	DEY	
FF7B:	30 E8	1057	BMI MON	NOT FOUND, GO TO MON
FF7D:	D9 CC FF	1058	CMP CHRTBL,Y	FIND CMND CHAR IN TEL
FF80:	DU F8	1059	BNE CHR5RCH	
FF82:	20 BE FF	1060	JSR TCSUB	FOUND, CALL CORRESPONDING
FF85:	A4 34	1061	LDY YSAV	SUBROUTINE
FF87:	4C 73 FF	1062	JMP NXTITM	
FF8A:	A2 03	1063	DIG	LDX #\$03
FF8C:	QA	1064	ASL A	
FF8D:	QA	1065	ASL A	GOT HEX DIG,
FF8E:	QA	1066	ASL A	SHIFT INTO A2
FF8F:	QA	1067	ASL A	
FF90:	QA	1068	NXTBIT	ASL A
FF91:	26 3E	1069	ROL A2L	
FF93:	26 3F	1070	ROL A2H	
FF95:	CA	1071	DEX	LEAVE X=\$FF IF DIG
FF96:	10 F0	1072	3PL NXTBIT	
FF98:	A5 31	1073	NXTBAS	LDA MODE
FF9A:	DU 06	1074	BNE NXTBS2	IF MODE IS ZERO
FF9C:	B5 3F	1075	LDA A2H,X	THEN COPY A2 TO
FF9E:	95 3D	1076	STA A1H,X	A1 AND A3
FFA0:	95 41	1077	STA A3H,X	
FFA2:	E8	1078	NXTBS2	INX
FFA3:	F0 F3	1079	SEQ NXTBAS	
FFA5:	DU 06	1080	BNE NXTCCHR	
FFA7:	A2 00	1081	GETNUM	LDX #S00
FFA9:	86 3E	1082	STX A2L	CLEAR A2
FFAB:	86 3F	1083	STX A2H	
FFAD:	89 00 J2	1084	NXTCHR	LDA IN,Y
FFB0:	C8	1085	INY	GET CHAR
FFB1:	49 B0	1086	EOR #S00	
FFB3:	C9 CA	1087	CMP #S0A	
FFB5:	90 D3	1088	BCC DIG	IF HEX DIG, THEN
FFB7:	69 0B	1089	ADC #S0B	
FFB9:	C9 FA	1090	CMP #SFA	
FFBB:	BU CD	1091	BCS DIG	
FFBD:	60	1092	RTS	
FFBE:	A9 FE	1093	TOSUB	LDA #GO/256
FFC0:	46	1094	PHA	PUSH HIGH-ORDER
FFC1:	B9 E3 FF	1095	LDA SUBTBL,Y	SUBR ADR ON STK
FFC4:	48	1096	PHA	PUSH LOW ORDER
FFC5:	A5 31	1097	LDA MODE	SUBR ADR ON STK
FFC7:	A0 00	1098	ZMODE	LDY #S00
FFC9:	64 31	1099	STY MODE	CLR MODE, OLD MODE
FFCB:	60	1100	RTS	TO A-REG
FFCC:	BC	1101	CHRTBL	GO TO SUBR VIA RTS
FFCD:	B2	1102	DFB \$B0	F("CTRL-C")
FFCE:	BE	1103	DFB \$B2	F("CTRL-Y")
FFCF:	ED	1104	DFB \$BE	F("CTRL-E")
FFD0:	EF	1105	DFB \$ED	F("T")
FFD1:	C4	1106	DFB \$C4	F("V")
FFD2:	EC	1107	DFB \$EC	F("CTRL-K")
FFD3:	A9	1108	DFB \$A9	F("S")
FFD4:	BB	1109	DFB \$BB	F("CTRL-P")
FFD5:	A6	1110	DFB \$A6	F("CTRL-B")
FFD6:	A4	1111	DFB \$A4	F("-")
FFD7:	06	1112	DFB \$06	F("M") (F=EX-OR \$B0+\$89)
FFD8:	95	1113	DFB \$95	F("<")
FFD9:	07	1114	DFB \$07	F("N")
FFDA:	02	1115	DFB \$02	F("I")
FFDB:	05	1116	DFB \$05	F("L")
FFDC:	F0	1117	DFB \$F0	F("W")
FFDD:	00	1118	DFB \$00	F("G")
FFDE:	EB	1119	DFB \$EB	F("R")
FFDF:	93	1120	DFB \$93	F(":")
FFE0:	A7	1121	DFB \$A7	F(".")
FFE1:	C6	1122	DFB \$C6	F("CR")
FFE2:	99	1123	DFB \$99	F(BLANK)
FFE3:	B2	1124	SUBTBL	DFB #BASCONT-1
FFE4:	C9	1125	DFB #USR-1	
FFE5:	BE	1126	DFB #REGZ-1	

FFE6:	C1	1127	DFB	#TRACE-1
FFE7:	35	1128	DFB	#VFY-1
FFE8:	6C	1129	DFB	#INPRT-1
FFE9:	C3	1130	DFB	#STEPZ-1
FFEA:	96	1131	DFB	#OUTPRT-1
FFEB:	AF	1132	DFB	#XBASIC-1
FFEC:	17	1133	DFB	#SETMODE-1
FFED:	17	1134	DFB	#SETMCODE-1
FFEE:	2B	1135	DFB	#MOVE-1
FFEF:	1F	1136	DFB	#LT-1
FFF0:	83	1137	DFB	#SETNORM-1
FFF1:	7F	1138	DFB	#SETINV-1
FFF2:	5D	1139	DFB	#LIST-1
FFF3:	CC	1140	DFB	#WRITE-1
FFF4:	B5	1141	DFB	#GO-1
FFF5:	FC	1142	DFB	#READ-1
FFF6:	17	1143	DFB	#SETMODE-1
FFF7:	17	1144	DFB	#SETMODE-1
FFF8:	F5	1145	DFB	#CRMON-1
FFF9:	03	1146	DFB	#BLANK-1
FFFA:	FB	1147	DFB	#NMI NMI VECTOR
FFFB:	03	1148	DFB	#NMI/256
FFFC:	59	1149	DFB	#RESET RESET VECTOR
FFFD:	FF	1150	DFB	#RESET/256
FFFE:	86	1151	DFB	#IRQ IRQ VECTOR
FFFF:	FA	1152	DFB	#IRQ/256
		1153 XQTNZ	EQU	\$3C

SYMBOL TABLE (NUMERICAL ORDER)

0000 LOCO	FC76 SCRL1	FB5B TABV
0022 WNDTOP	FC9E CLEOLZ	FB78 VIDWAIT
0026 GBASL	FCAA WAIT3	FB9D ESCNOW
002A BAS2L	FCC9 HEADR	FBD9 BELL1
002D V2	FCE5 WRTAPE	FBF4 ADVANCE
002E FORMAT	FCFD RDBIT	FC1A UP
0030 COLOR	FD2F ESC	FC2C ESC1
0034 YSAV	FD62 CANCEL	FC62 CR
0038 KSWL	0001 LOC1	FCBC SCRL2
003C A1L	0023 WNDBTM	FCA0 CLEOL2
0040 A3L	0027 QBASH	FCB4 NXTA4
0044 A5L	002B BAS2H	FCD6 WRBIT
0047 YREG	002D RMNEM	FCEC RDBYTE
004F RNDH	002F LASTIN	FDOC RDKEY
03F2 SOFTEV	0031 MODE	FD35 RDCHAR
03FB NMI	0035 YSAV1	FD67 GETLNZ
C000 IOADR	0039 KSWH	0020 WNDLFT
C030 SPKR	003D A1H	0024 CH
C053 MIXSET	0041 A3H	0028 BASL
C057 HIRES	0045 A5H	002C H2
C05B CLRAN1	0048 STATUS	002E MASK
C05F CLRAN3	0095 PICK	002F LENGTH
CFFF CLRROM	03F4 PWREDUP	0032 INVFLG
FB0C RTMASK	03FE IRQLOC	0036 CSWL
FB26 VLINEZ	C000 KBD	003A PCL
FB36 CLRTOP	C050 TXTCLR	003E A2L
FB56 GBCALC	C054 LOWSCR	0042 A4L
FB7F RTMSKZ	C058 SETANO	0045 ACC
F8A5 ERR	C05C SETAN2	0049 SPNT
FBC9 MNNDX3	C060 TAPEIN	0200 IN
FBF5 NXTCOL	E000 BASIC	03F5 AMPERV
F926 PRADR3	F80E PLOT1	0400 LINE1
F940 PRNTYX	F828 VLINE	C010 KBDSTRB
F94A PRBL2	F838 CLRSC2	C051 TXTSET
F956 PCADJ3	F864 SETCOL	C055 HISCR
F9A6 FMT2	F882 INSDS1	C059 CLRANO
FA00 MNEMR	F8A9 GETFMT	C05D CLRAN2
FA62 RESET	F8D0 INSTDSP	C064 PADDLO
FAA3 NOFIX	F8F9 PRMN2	E003 BASIC2
FABA SLOOP	F92A PRADR4	F819 HLINE
FAE4 RDSP1	F941 PRNTAX	F831 RTS1
FB11 XLTBL	F94C PRBL3	F83C CLRSC3
FB2E RTS2D	F95C PCADJ4	F871 SCRN
FB4B SETWND	F9B4 CHAR1	F88C INSDS2
FB6F SETPWRC	FA40 IRQ	F8BE MNNDX1
FB97 ESCOLD	FA6F INITAN	F8D4 PRNTOP
FBDO BASCLC2	FAA6 PWRUP	F910 PRADR1
FBFO STORADV	FAC7 NXTBYT	F930 PRADR5
FC10 BS	FAFD PWRCON	F944 PRNTX
FC2B RTS4	FB19 RTBL	F953 PCADJ
FC58 HOME	FB2F INIT	F961 RTS2

F9BA CHAR2	F914 PRADR2	FDF0 COUT1
FA4C BREAK	F938 RELADR	FE0B STOR
FA81 NEWMON	F948 PRBLNK	FE20 LT
FAA9 SETPG3	F954 PCADJ2	FE58 VFYOK
FAD7 REGDSP	F962 FMT1	FE78 A1PCLP
FB02 DISKID	F9C0 MNEML	FE86 SETIFLG
FB1E PREAD	FA59 OLDBRK	FE93 SETVID
FB39 SETTXT	FA9B FIXSEV	FEA7 IOPRT1
FB60 APPLEII	FAAB SETPLP	FEB6 GO
FB88 KBDWAIT	FADA RGDSP1	FECA USR
FBA5 ESCNEW	FB09 TITLE	FEFF WRBYT2
FBE4 BELL2	FB25 PREAD2	FF16 RD3
FBFC RTS3	FB40 SETGR	FF44 RESTR1
FC22 VTAB	FB65 STITLE	FF65 MON
FC42 CLREOP	FB94 NOWAIT	FF8A DIG
FC66 LF	FBC1 BASCALC	FFA7 GETNUM
FC95 SCR3	FBEF RTS2B	FFCC CHRTBL
FCAB WAIT	FBFD VIDOUT	FD84 ADDINP
FCBA NXTA1	FC24 VTABZ	FDA3 XAMB
FCDB ZERDLY	FC46 CLEOP1	FDC5 RTS4C
FCEE RDBYT2	FC70 SCROLL	FDE3 PRHEX
FD1B KEYIN	FC9C CLREOL	FDF6 COUT2
FD3D NOTCR	FCA9 WAIT2	FE17 RTS5
FD6A GETLN	FCC8 RTS4B	FE22 LT2
0021 WNDWDTH	FCE2 ONEDLY	FE5E LIST
0025 CV	FCFA RD2BIT	FE7F A1PCRTS
0029 BASH	FD21 KEYIN2	FE89 SETKBD
002C LMNEM	FD5F NOTCR1	FE95 OUTPORT
002E CHKSUM	FD71 BCKSPC	FEA9 IOPRT2
002F SIGN	FD75 NXTCHAR	FEBF REGZ
0033 PROMPT	FD92 PRA1	FECD WRITE
0037 CSWH	FDB3 XAM	FEF6 CRMON
003B PCH	FDD1 ADD	FF2D PRERR
003F A2H	FDED COUT	FF4A SAVE
0043 A4H	FE04 BLANK	FF69 MONZ
0046 XREG	FE1D SETMDZ	FF90 NXTBIT
004E RNDL	FE36 VFY	FFAD NXTCHR
03F0 BRKV	FE75 A1PC	FFE3 SUBTBL
03F8 USRADR	FE84 SETNORM	FD8E CROUT
07F8 MSLOT	FE8D INPRT	FDAD MOD8CHK
C020 TAPEOUT	FE9B IOPRT	FDC6 XAMP
C052 MIXCLR	FEB3 BASCONT	FDE5 PRHEXZ
C056 LORES	FEC4 STEPZ	FE00 BL1
C05A SETAN1	FEED WRBYTE	FE18 SETMODE
C05E SETAN3	FF0A RD2	FE2C MOVE
C070 PTRIG	FF3F RESTORE	FE63 LIST2
F800 PLOT	FF59 OLDRST	FE80 SETINV
F81C HLINE1	FF7A CHRCSRCH	FE8B IMPORT
F832 CLRSCR	FFA2 NXTBS2	FE97 OUTPRT
F847 GBASCALC	FFC7 ZMODE	FEBO XBASIC
F879 SCR2	FD7E CAPTST	FEC2 TRACE
F89B IEVEN	FD96 PRYX2	FED4 WR1
F8C2 MNNDX2	FDB6 DATAOUT	FEFD READ
F8DB PRNTBL	FDDA PRBYTE	FF3A BELL

FF4C SAV1
FF73 NXTITM
FF98 NXTBAS
FFBE TOSUB

SYMBOL TABLE (ALPHABETICAL ORDER)

003D A1H	F956 PCADJ3	FEA7 IOPRT1
FE7F A1PCRTS	0095 PICK	FA40 IRQ
0040 A3L	F910 PRADR1	FD1B KEYIN
0044 A5L	F930 PRADR5	002F LASTIN
FBF4 ADVANCE	FDDA PRBYTE	FE5E LIST
002A BAS2L	FDE3 PRHEX	0001 LOC1
0029 BASH	F8DB PRNTBL	FE20 LT
FD71 BCKSPC	0033 PROMPT	F9C0 MNEML
FE00 BL1	03F4 PWREDUP	F8C9 MNNDX3
FC10 BS	FF16 RD3	FF65 MON
F9BA CHAR2	FD35 RDCHAR	03FB NMI
0024 CH	FAD7 REGDSP	FB94 NOWAIT
C059 CLRANO	FF3F RESTORE	FF90 NXTBIT
FC9C CLREOL	004F RNDH	FFAD NXTCHR
FB3C CLRSC3	F87F RTMSKZ	FF59 OLDRST
FDED COUT	F961 RTS2	C064 PADDLO
FC62 CR	003C A1L	F95C PCADJ4
0025 CV	003F A2H	F80E PLOT1
FBA5 ERR	0043 A4H	F914 PRADR2
FB97 ESCOLD	0045 ACC	F94A PRBL2
F9A6 FMT2	03F5 AMPERV	FB1E PREAD
0026 GBASL	FBC1 BASCALC	FDE5 PRHEXZ
FD6A GETLN	E000 BASIC	F8D4 PRNTOP
FCC9 HEADR	FBD9 BELL1	FD96 PRYX2
FB19 HLINE	FE04 BLANK	FAA6 PWRUP
0200 IN	FD62 CANCEL	FCFD RDBIT
FBB2 INSDS1	002E CHKSUM	FDOC RDKEY
C000 IOADR	FCA0 CLEOL2	FEBF REGZ
03FE IRQLOC	C05B CLRAN1	FF44 RESTR1
C000 KBD	FC42 CLREOP	004E RNDL
0038 KSWL	F832 CLRSCR	F831 RTS1
0400 LINE1	FDF0 COUT1	FBFC RTS3
0000 LOCO	FEF6 CRMON	FE78 A1PCLP
FE22 LT2	FDB6 DATAOUT	003E A2L
C053 MIXSET	FC2C ESC1	0042 A4L
F8C2 MNNDX2	FD2F ESC	FD84 ADDINP
FF69 MONZ	002E FORMAT	FB60 APPLEII
FA81 NEWMON	F856 GBCALC	FBDO BASCLC2
FD5F NOTCR1	FFA7 GETNUM	E003 BASIC2
FF98 NXTBAS	C057 HIRES	FBE4 BELL2
FD75 NXTCHAR	FC58 HOME	FA4C BREAK
FA59 OLDBRK	FB2F INIT	FD7E CAPTST
FE97 OUTPRT	F88C INSDS2	FF7A CHRSRCH

FC9E CLEOLZ	FF3A BELL	C05C SETAN2
C05D CLRAN2	03F0 BRKV	FE86 SETIFLG
CFFF CLRROM	F9B4 CHAR1	FE18 SETMODE
F836 CLRTOP	FFCC CHRTBL	FB6F SETPWRC
FDF6 COUTZ	FC46 CLEOP1	002F SIGN
0037 CSWH	C05F CLRAN3	0049 SPNT
FFBA DIG	F838 CLRSC2	FE0B STOR
FBA5 ESCNEW	0030 COLOR	C060 TAPEIN
FA9B FIXSEV	FD8E CROUT	FEC2 TRACE
F847 GBASCALC	0036 CSWL	FECA USR
F8A9 GETFMT	FB02 DISKID	FE58 VFYOK
FEB6 GO	FB9B ESCNOW	F828 VLINE
C055 HISCR	F962 FMT1	FCAB WAIT
F89B IEVEN	0027 GBASH	0022 WNDTOP
FE8B INPORT	FD67 GETLNZ	FEFF WRBYT2
F8D0 INSTDSP	002C H2	FDA3 XAMB
FEA9 IOPRT2	FB1C HLINE1	FB11 XLTBL
C010 KBDSTRB	FA6F INITAN	0034 YSAV
FD21 KEYIN2	FEBD INPRT	FC8C SCRL2
002F LENGTH	0032 INVFLG	FC70 SCROLL
FE63 LIST2	FE9B IOPRT	C05E SETAN3
C056 LORES	FB88 KBDWAIT	FE80 SETINV
002E MASK	0039 KSWH	FE84 SETNORM
FA00 MNEMR	FC66 LF	FB39 SETTXT
FDAD MOD8CHK	002C LMNEM	FABA SLOOP
FE2C MOVE	C054 LOWSCR	0048 STATUS
FAA3 NOFIX	C052 MIXCLR	FBFO STORADV
FCBA NXTA1	F8BE MNNDX1	C020 TAPEOUT
FFA2 NXTBS2	0031 MODE	C050 TXTCLR
F8F5 NXTCOL	07FB MSLOT	03F8 USRADR
FCE2 ONEDLY	FD3D NOTCR	FBFD VIDOUT
F954 PCADJ2	FCB4 NXTA4	FC24 VTABZ
003B PCH	FAC7 NXBTBYT	FCAA WAIT3
F800 PLOT	FF73 NXITIM	0021 WNDWDTH
F926 PRADR3	FE95 OUTPORT	FEED WRBYTE
F94C PRBL3	F953 PCADJ	FDC6 XAMPM
FB25 PREAD2	003A PCL	0046 XREQ
F8F9 PRMN2	FD92 PRA1	FCDB ZERDLY
F944 PRNTX	F92A PRADR4	FF4C SAV1
C070 PTRIG	F948 PRBLNK	FC95 SCRL3
FCFA RD2BIT	FF2D PRERR	C058 SETANO
FCEE RDBYT2	F941 PRNTAX	F864 SETCOL
FAE4 RDSP1	F940 PRNTYX	FE89 SETKBD
F938 RELADR	FAFD PWRCON	FAA9 SETPG3
FADA RGDSP1	FF0A RD2	FE93 SETVID
FB19 RTBL	FCEC RDBYTE	03F2 SOFTEV
FBEF RTS2B	FEFD READ	FEC4 STEPZ
FCC8 RTS4B	FA62 RESET	FFE3 SUBTBL
FE75 A1PC	002D RMNEM	FB09 TITLE
0041 A3H	FB0C RTMASK	C051 TXTSET
0045 A5H	FB2E RTS2D	002D V2
FDD1 ADD	FDC5 RTS4C	FB78 VIDWAIT
002B BAS2H	FE17 RTS5	FC22 VTAB
FEB3 BASCONT	FC2B RTS4	0023 WNDBTM
0028 BASL	FC76 SCRL1	FED4 WR1
	F879 SCRNN2	

FECD WRITE
FDB3 XAM
0047 YREG
FFC7 ZMODE
FF4A SAVE
FB71 SCRN
C05A SETAN1
FB40 SETGR
FE1D SETMDZ
FAAB SETPLP
FB4B SETWND
C030 SPKR
FB65 STITLE
FB5B TABV
FFBE TOSUB
FC1A UP
FE36 VFY
F826 VLINEZ
FCA9 WAIT2
0020 WNDLFT
FCD6 WRBIT
FCE5 WRTAPE
FEBO XBASIC
0035 YSAV1

SYMBOL TABLE SIZE
2589 BYTES USED
2531 BYTES REMAINING

SLIST 4A

GLOSSARY

6502: The manufacturer's name for the microprocessor at the heart of your Apple.

Address: As a noun: the particular number associated with each memory location. On the Apple, an address is a number between \emptyset and 65535 (or \$0000 and \$FFFF hexadecimal). As a verb: to refer to a particular memory location.

Address Bus: The set of wires, or the signal on those wires, which carry the binary-encoded address from the microprocessor to the rest of the computer.

Addressing mode: The Apple's 6502 microprocessor has thirteen distinct ways of referring to most locations in memory. These thirteen methods of forming addresses are called **addressing modes**.

Analog: Analog measurements, as opposed to digital measurements, use an continuously variable physical quantity (such as length, voltage, or resistance) to represent values. Digital measurements use precise, limited quantities (such as presence or absence of voltages or magnetic fields) to represent values.

AND: A binary function which is "on" if and only if all of its inputs are "on".

Apple: 1. The round fleshy fruit of a Rosaceous tree (*Pyrus Malus*). 2. A brand of personal computer. 3) Apple Computer, Inc., manufacturer of home and personal computers.

ASCII: An acronym for the American Standard Code for Information Interchange (often called "USASCII" or misinterpreted as "ASC-II"). This standard *code* assigns a unique value from \emptyset to 127 to each of 128 numbers, letters, special characters, and control characters.

Assembler: 1) One who assembles electronic or mechanical equipment. 2) A program which converts the *mnemonics* and *symbols* of assembly language into the *opcodes* and *operands* of machine language.

Assembly language: A language similar in structure to machine language, but made up of *mnemonics* and *symbols*. Programs written in assembly language are slightly less difficult to write and understand than programs in machine language.

BASIC: Acronym for "Beginner's All-Purpose Symbolic Instruction Code". BASIC is a *higher-level language*, similar in structure to FORTRAN but somewhat easier to learn. It was invented by Kemney and Kurtz at Dartmouth College in 1963 and has proved to be the most popular language for personal computers.

Binary: A number system with two digits, "0" and "1", with each digit in a binary number representing a power of two. Most digital computers are binary, deep down inside. A binary signal is easily expressed by the presence or absence of something, such as an electrical potential or a magnetic field.

Binary Function: An operation performed by an electronic circuit which has one or more inputs and only one output. All inputs and outputs are binary signals. See *AND OR*, and *Exclusive-OR*.

Bit: A *Binary digit*. The smallest amount of information which a computer can hold. A single bit specifies a single value: "0" or "1". Bits can be grouped to form larger values (see *Byte* and *Nibble*).

Board: See *Printed Circuit Board*.

Bootstrap ("boot"): To get a system running from a *cold-start*. The name comes from the machine's attempts to "pull itself off the ground by tugging on its own bootstraps."

Buffer: A device or area of memory which is used to hold something temporarily. The "picture buffer" contains graphic information to be displayed on the video screen; the "input buffer" holds a partially formed input line.

Bug: An error. A *hardware bug* is a physical or electrical malfunction or design error. A *software bug* is an error in programming, either in the logic of the program or typographical in nature. See "feature".

Bus: A set of wires or *traces* in a computer which carry a related set of data from one place to another, or the data which is on such a bus.

Byte: A basic unit of measure of a computer's memory. A byte usually comprises eight *bits*. Thus, it can have a value from 0 to 255. Each character in the *ASCII* can be represented in one byte. The Apple's memory locations are all one byte, and the Apple's addresses of these locations consist of two bytes.

Call: As a verb: to leave the program or subroutine which is currently executing and to begin another, usually with the intent to return to the original program or subroutine. As a noun: an instruction which calls a subroutine.

Character: Any *graphic symbol* which has a specific meaning to people. Letters (both upper- and lower-case), numbers, and various symbols (such as punctuation marks) are all characters.

Chip: See *Integrated Circuit*.

Code: A method of representing something in terms of something else. The ASCII code represents characters as binary numbers, the BASIC language represents algorithms in terms of program statements. **Code** is also used to refer to programs, usually in *low-level languages*.

Cold-start: To begin to operate a computer which has just been turned on.

Color burst: A signal which color television sets recognize and convert to the colored dots you see on a color TV screen. Without the color burst signal, all pictures would be black-and-white.

Computer: Any device which can receive and store a set of *instructions*, and then act upon those instructions in a predetermined and predictable fashion. The definition implies that both the instruction and the *data* upon which the instructions act can be changed. A device whose instructions cannot be changed is not a computer.

Control (CTRL) character: Characters in the *ASCII* character set which usually have no graphic representation, but are used to control various functions. For example, the RETURN control character is a signal to the Apple that you have finished typing an *input line* and you wish the computer to act upon it.

CRT: Acronym for "Cathode-Ray Tube", meaning any television screen, or a device containing such a screen.

Cursor: A special symbol which reminds you of a certain position on something. The cursor on a slide rule lets you line up numbers; the cursor on the Apple's screen reminds you of where you are when you are typing.

Data (datum): Information of any type.

Debug: To find *bugs* and eliminate them.

DIP: Acronym for "Dual In-line Package", the most common container for an Integrated Circuit. DIPs have two parallel rows of *pins*, spaced on one-tenth of an inch centers. DIPs usually come in 14-, 16-, 18-, 20-, 24-, and 40-pin configurations.

Disassembler: A program which converts the *opcodes* of *machine language* to the *mnemonics* of *assembly language*. The opposite of an *assembler*.

Display: As a noun: any sort of output device for a computer, usually a *video* screen. As a noun: to place information on such a screen.

Edge connector: A socket which mates with the edge of a *printed circuit board* in order to exchange electrical signals.

Entry point: The location used by a machine-language subroutine which contains the first executable instruction in that subroutine; consequently, often the beginning of the subroutine.

Exclusive-OR: A binary function whose value is "off" only if all of its inputs are "off", or all of its inputs are "on".

Execute: To perform the intention of a command or instruction. Also, to run a program or a portion of a program.

Feature: A *bug* as described by the marketing department.

Format: As a noun: the physical form in which something appears. As a verb: to specify such a form.

Graphic: Visible as a distinct, recognizable shape or color.

Graphics: A system to display graphic items or a collection of such items.

Hardware: The physical parts of a computer.

Hexadecimal: A number system which uses the ten digits 0 through 9 and the six letters A through F to represent values in base 16. Each hexadecimal digit in a hexadecimal number represents a power of 16. In this manual, all hexadecimal numbers are preceded by a dollar sign (\$).

High-level Language: A *language* which is more intelligible to humans than it is to machines.

High-order: The most important, or item with the highest value, of a set of similar items. The high-order bit of a byte is that which has the highest place value.

High part: The *high-order* byte of a two-byte address. In decimal, the high part of an address is the quotient of the address divided by 256. In the 6502, as in many other microprocessors, the high part of an address comes last when that address is stored in memory.

Hz (Hertz): Cycles per second. A bicycle wheel which makes two revolutions in one second is running at 2Hz. The Apple's microprocessor runs at 1,023,000Hz.

I/O: See *Input/Output*.

IC: See *Integrated Circuit*.

Input: As a noun: data which flows from the outside world into the computer. As a verb: to obtain data from the outside world.

Input/Output (I/O): The software or hardware which exchanges data with the outside world.

Instruction: The smallest portion of a program that a computer can execute. In 6502 machine language, an instruction comprises one, two, or three bytes; in a higher-level language, instructions may be many characters long.

Integrated circuit: A small (less than the size of a fingernail and about as thin) wafer of a glassy material (usually silicon) into which has been etched an electronic circuit. A single IC can contain from ten to ten thousand discrete electronic components. ICs are usually housed in *DIPs* (see above), and the term IC is sometimes used to refer to both the circuit and its package.

Interface: An exchange of information between one thing and another, or the mechanisms which make such an exchange possible.

Interpreter: A program, usually written in machine language, which understands and executes a higher-level language.

Interrupt: A physical effect which causes the computer to jump to a special interrupt-handling subroutine. When the interrupt has been taken care of, the computer resumes execution of the interrupted program with no noticeable change. Interrupts are used to signal the computer that a particular device wants attention.

K: Stands for the greek prefix "Kilo", meaning one thousand. In common computer-related usage, "K" usually represents the quantity 2^{10} , or 1024 (hexadecimal \$400).

Kilobyte: 1,024 bytes.

Language: A computer language is a code which (hopefully!) both a programmer and his computer understand. The programmer expresses what he wants to do in this code, and the computer understands the code and performs the desired actions.

Line: On a video screen, a "line" is a horizontal sequence of graphic symbols extending from one edge of the screen to the other. To the Apple, an *input line* is a sequence of up to 254 characters, terminated by the control character RETURN. In most places which do not have personal computers, a line is something you wait in to use the computer.

Low-level Language: A *language* which is more intelligible to machines than it is to humans.

Low-order: The least important, or item with the least value, of a set of items. The low-order bit in a byte is the bit with the least place value.

Low part: The *low-order* byte of a two-byte address. In decimal, the low part of an address is the remainder of the address divided by 256, also called the "address *modulo* 256." In the 6502, as in many other microprocessors, the low part of an address comes first when that address is stored in memory.

Machine language: The lowest level language which a computer understands. Machine

languages are usually binary in nature. Instructions in machine language are single-byte *opcodes* sometimes followed by various *operands*.

Memory address: A memory address is a two-byte value which selects a single memory location out of the *memory map*. Memory addresses in the Apple are stored with their low-order bytes first, followed by their high-order bytes.

Memory location: The smallest subdivision of the memory map to which the computer can refer. Each memory location has associated with it a unique *address* and a certain *value*. Memory locations on the Apple comprise one byte each.

Memory Map: This term is used to refer to the set of all memory locations which the microprocessor can address directly. It is also used to describe a graphic representation of a system's memory.

Microcomputer: A term used to described a computer which is based upon a microprocessor.

Microprocessor: An integrated circuit which understands and executes machine language programs.

Mnemonic: An acronym (or any other symbol) used in the place of something more difficult to remember. In *Assembly Language*, each machine language opcode is given a three letter mnemonic (for example, the opcode \$60 is given the mnemonic RTS, meaning "ReTurn from Subroutine").

Mode: A condition or set of conditions under which a certain set of rules apply.

Modulo: An arithmetic function with two operands. *Modulo* takes the first operand, divides it by the second, and returns the remainder of the division.

Monitor: 1) A closed-circuit television receiver. 2) A program which allows you to use your computer at a very low level, often with the values and addresses of individual memory locations.

Multiplexer: An electronic circuit which has many data inputs, a few selector inputs, and one output. A multiplexer connects one of its many data inputs to its output. The data input it chooses to connect to the output is determined by the selector inputs.

Mux: See *Multiplexer*.

Nibble: Colloquial term for half of a byte, or four bits.

Opcode: A machine language instruction, numerical (often binary) in nature.

OR: A binary function whose value is "on" if at least one of its inputs are "on".

Output: As a noun, data generated by the computer whose destination is the real world. As a verb, the process of generating or transmitting such data.

Page: 1) A screenfull of information on a video display. 2) A quantity of memory locations, addressable with one byte. On the Apple, a "page" of memory contains 256 locations.

Pascal: A noted French scientist.

PC board: See *Printed Circuit Board*.

Peripheral: Something attached to the computer which is not part of the computer itself. Most peripherals are input and/or output devices.

Personal Computer: A computer with *memory*, *languages*, and *peripherals* which are well-suited for use in a home, office, or school.

Pinout: A description of the function of each pin on an IC, often presented in the form of a diagram.

Potentiometer: An electronic component whose resistance to the flow of electrons is proportional to the setting of a dial or knob. Also known as a "pot" or "variable resistor".

Printed Circuit Board: A sheet of fiberglass or epoxy onto which a thin layer of metal has been applied, then etched away to form *traces*. Electronic components can then be attached to the board with molten solder, and they can exchange electronic signals via the etched traces on the board. Small printed circuit boards are often called "cards", especially if they are meant to connect with *edge connectors*.

Program: A sequence of instructions which describes a process.

PROM: Acronym for "*Programmable Read-Only Memory*". A PROM is a ROM whose contents can be altered by electrical means. Information in PROMs does not disappear when the power is turned off. Some PROMs can be erased by ultraviolet light and be reprogrammed.

RAM: See *Random-Access Memory*.

Random-Access Memory (RAM): This is the main memory of a computer. The acronym RAM can be used to refer either to the integrated circuits which make up this type of memory or the memory itself. The computer can store values in distinct locations in RAM and recall them again, or alter and re-store them if it wishes. On the Apple, as with most small computers, the values which are in RAM memory are lost when the power to the computer is turned off.

Read-Only Memory (ROM): This type of memory is usually used to hold important programs or data which must be available to the computer when the power is first turned on. Information in ROMs is placed there in the process of manufacturing the ROMs and is unalterable. Information stored in ROMs does not disappear when the power is turned off.

Reference: 1) A source of information, such as this manual. 2) As a verb, the action of examining or altering the contents of a memory location. As a noun, such an action.

Return: To exit a subroutine and go back to the program which called it.

ROM: See *Read-Only Memory*.

Run: To follow the sequence of instructions which comprise a program, and to complete the process outlined by the instructions.

Scan line: A single sweep of a cathode beam across the face of a *cathode-ray tube*.

Schematic: A diagram which represents the electrical interconnections and circuitry of an electronic device.

Scroll: To move all the text on a display (usually upwards) to make room for more (usually at the bottom).

Soft switch: A two-position switch which can be "thrown" either way by the software of a computer.

Software: The *programs* which give the hardware something to do.

Stack: A reserved area in memory which can be used to store information temporarily. The information in a stack is referenced not by address, but in the order in which it was placed on the stack. The last datum which was "pushed" onto the stack will be the first one to be "popped" off it.

Strobe: A momentary signal which indicates the occurrence of a specific event.

Subroutine: A segment of a program which can be executed by a single *call*. Subroutines are used to perform the same sequence of instructions at many different places in one program.

Syntax: The structure of instructions in a given *language*. If you make a mistake in entering an instruction and garble the syntax, the computer sometimes calls this a "SYNTAX ERROR."

Text: Characters, usually letters and numbers. "Text" usually refers to large chunks of English, rather than computer, language.

Toggle switch: A two-position switch which can only flip from one position to the other and back again, and cannot be directly set either way.

Trace: An etched conductive path on a *Printed-Circuit Board* which serves to electronically connect components.

Video: 1) Anything visual. 2) Information presented on the face of a *cathode-ray tube*.

Warm-start: To restart the operation of a computer after you have lost control of its language or operating system.

Window: Something out of which you jump when the power fails and you lose a large program. Really: a reserved area on a *display* which is dedicated to some special purpose.

BIBLIOGRAPHY

Here are some other publications which you might enjoy:

Synertek/MOS Technology 6500 Programming Manual

This manual is an introduction to machine language programming for the MC6502 microprocessor. It describes the machine language operation of the Apple's microprocessor in meticulous detail. However, it contains no specific information about the Apple.

This book is available from Apple. Order part number A2L0003.

Synertek/MOS Technology 6500 Hardware Manual

This manual contains a detailed description of the internal operations of the Apple's 6502 microprocessor. It also has much information regarding interfacing the microprocessor to external devices, some of which is pertinent to the Apple.

This book is also available from Apple. Order part number A2L0002.

The Apple II Monitor Peeled

This book contains a thorough, well-done description of the operating subroutines within the Apple's original Monitor ROM.

This is available from the author:

William E. Dougherty
14349 San Jose Street
Los Angeles, CA 91345

Programming the 6502

This book, written by Rodnay Zaks, is an excellent tutorial manual on machine and assembly-language programming for the Apple's 6502 microprocessor.

This manual is available from Sybex Incorporated, 2020 Milvia, Berkeley, CA 94704. It should also be available at your local computer retailer or bookstore. Order book number C202.

6502 Applications

This book, also written by Rodnay Zaks, describes many applications of the Apple's 6502 microprocessor.

This is also available from Sybex. Order book number D302.

System Description: The Apple II

Written by Steve Wozniak, the designer of the Apple computers, this article describes the basic construction and operation of the Apple II.

This article was originally published in the May, 1977 issue of BYTE magazine, and is available from BYTE Publications, Inc. Peterborough, NH 30458.



SWEET16: The 6502 Dream Machine

Also written by Steve Wozniak, this article describes the SWEET16® interpretive machine language enclosed in the Apple's Integer BASIC ROMs.

This article appeared in the October, 1977 issue of BYTE magazine, and is available from BYTE Publications, Inc. Peterborough, NH 30458.

More Colors for your Apple

This article, written by Allen Watson III, describes in detail the Apple High-Resolution Graphics mode. Also included is a reply by Steve Wozniak, the designer of the Apple, describing a modification you can make to update your Revision 0 Apple to add the two extra colors available on the Revision 1 board.

This article appeared in the June, 1979 issue of BYTE magazine, and is available from BYTE Publications, Inc. Peterborough, NH 30458.

Call APPLE (Apple Puget Sound Program Library Exchange)

This is one of the largest Apple user group newsletters. For information, write:

Apple Puget Sound Program Library Exchange
6708 39th Ave. Southwest
Seattle, Wash., 98136

The Cider Press

This is another large club newsletter. For information, write:

The Cider Press
c/o The Apple Core of San Francisco
Box 4816
San Francisco, CA 94101

INDEX

- 190 GENERAL INDEX
- 194 INDEX OF FIGURES
- 195 INDEX OF PHOTOS
- 195 INDEX OF TABLES
- 195 CAST OF CHARACTERS

GENERAL INDEX

Ø boards, Revision.....3, 26
1 board, Revision.....3, 26
2716 type PROMs.....94
50Hz modification, Eurapple.....10
6502 instruction set.....Appendix A
6502 internal registers.....53, 81
6502 microprocessor.....3, 88

-- A --

Access Memory (RAM), Random.....3
address and data buses.....88, 90
address multiplexer, RAM.....96
addresses and data.....40
addressing modes.....66
analog inputs.....24
annunciator outputs.....23, 36, 100
annunciator special locations.....24
Apple Firmware card.....73
Apple Language card.....3
Apple main board, the.....3, 89
Apple Mini-assembler.....49
Apple, photo of the.....2
Apple power supply, the.....2, 92
Apple, setting up the.....2
Apples, varieties of.....25
ASCII character code.....5, 6, 7, 8, 15
ASCII codes, keys and.....7
Autostart ROM listing.....Appendix C
Autostart ROM Reset.....36
Autostart ROM special locations.....37
Autostart ROM.....25
auxiliary video connector.....9

-- B --

backspace character.....30
backspace key.....34
BASIC, entering.....34, 54
BASIC, reentering.....34, 54
bell character.....31
block pinout, configuration.....71
blocks, RAM configuration.....70
board I/O, peripheral.....79
board, Revision Ø.....3, 26
board, Revision 1.....3, 26
board, the Apple main.....3, 89
board schematic, main.....110
buffer, picture.....12
buffer, input.....33
built-in I/O.....78, 98

buses, address and data.....88, 90
byte, power-up.....37, 65

-- C --

card, Apple Language.....3
card, Apple Firmware.....73
cassette interface jacks.....22, 103
cassette interface.....22
cassette tape, saving to.....46
cassette tape, reading from.....47
changing memory.....43
character code, ASCII.....5, 6, 7, 8, 15
character, backspace.....30
character, line-feed.....30
character, RETURN.....30
character, bell.....31
characters, prompting.....33
characters, keyboard.....7, 8
characters, control.....7
clearing the keyboard strobe.....6
code, ASCII character.....5, 6, 7, 8, 15
codes, escape.....34
codes, keys and ASCII.....7
cold start.....36
colors, Low-Res.....11, 17
colors, High-Res.....11, 19, 26
colors, European High-Res.....20
command loops, Monitor.....56
commands, creating your own.....57
commands, summary of Monitor.....59
comparing memory.....46
configuration block pinout.....71
configuration blocks, RAM.....70
configuration, RAM memory.....70
connector pinout, peripheral.....106
connector, keyboard.....5, 102
connector, power.....104
connector, speaker.....105
connector, Game I/O.....23, 100
connector, auxiliary video.....9
connector, video.....9
connectors, peripheral.....3, 105
connnector pinouts, keyboard.....103
control characters.....7
control values, Normal/Inverse.....32
Controllers, Game.....24, 100
COUT, KEYIN switches.....83
COUT standard output subroutine.....30
creating your own commands.....57
CSW/KSW switches.....83

cursor.....	30
cursor, output.....	30
cycle, the RESET	36
-- D --	
data buses, address and	90
data, addresses and.....	40
debugging programs.....	51
display special locations, video	13
display, video.....	9
-- E --	
editing an input line	33
editing features.....	25
entering BASIC	34, 54
entering the Monitor	40
entry vector, soft	37
escape (ESC) codes	34
Eurapple 50Hz modification	10
European High-Res colors.....	20
examining memory	41
expansion ROM.....	84
-- F --	
feature, the Stop-List	26, 30
features, input/output	20
features, editing.....	25
features, keyboard	5
features, microprocessor.....	88
features, power supply.....	92
Firmware card, Apple	73
("flag") inputs, one-bit.....	24, 78
format, Text screen	16
format, Low-Res screen	18
format, High-Res screen	21
from cassette tape, reading.....	47
-- G --	
Game Controllers	24
Game I/O connector	23, 100
generator, the video	96
GETLN and input lines	33
graphics modes	11
graphics, High-Res	19
graphics, Low-Res	17
-- H --	
hexadecimal notation.....	40
High-Res colors, European.....	20
High-Res graphics.....	19
High-Res screen, the	21
High-Res video mode, the	19
High-Res colors	19, 26
-- I --	
input buffer.....	33
input line, editing an	33
input lines, GETLN and.....	33
input prompting.....	32
input subroutine, RDKEY standard	32
input/output features.....	20
input/output special locations.....	25
input/output	78
inputs, data.....	78
inputs, one-bit ("flag")	24, 78
inputs, analog	24
inputs, single-bit pushbutton	78
instruction set, 6502	Appendix A
instructions, Mini-Assembler	66
interface jacks, cassette	22, 103
interface, cassette	22
internal registers, 6502	53, 81
interrupts	65, 107, 108
inverse text mode	32, 54
I/O connector, Game	23, 100
I/O programming suggestions	80
I/O special locations	79
I/O, built-in.....	78, 98
I/O, peripheral board.....	79
I/O, peripheral slot	79
-- J --	
jacks, cassette interface.....	22, 103
jacks, video output	97
jumper, "USER 1"	99
-- K --	
key, backspace	34
key, retype	34
keyboard characters	7, 8
keyboard connector	5, 102
keyboard connector pinouts	103
keyboard features	5
keyboard schematic	101
keyboard special locations	6
keyboard strobe	6, 78, 79, 98, 102
keyboard strobe, clearing the	6
keyboard, review of the.....	4, 100
keyboard, reading the	6
KEYIN switches, COUT,	83

keys and ASCII codes.....7

-- L --

Language card, Apple.....3, 69
leaving the Mini-Assembler.....50
line, editing an input.....33
line-feed character.....30
lines, GETLN and input.....33
listing, Autostart ROM.....Appendix C
listing, Monitor ROM.....Appendix C
listing machine language programs.....49
list of special locations.....Appendix B
locations, list of special.....Appendix B
locations, annunciator special.....24
locations, video display special.....13
locations, input/output special.....25
locations, text window special.....31
locations, Autostart ROM special.....37
locations, Monitor special.....65
locations, keyboard special.....6
locations, I/O special.....79
loops, Monitor command.....56
Low-Res colors.....17
Low-Res screen, the.....18
Low-Res video mode, the.....17
lukewarm start.....36

-- M --

machine language programs, listing.....49
main board, the Apple.....3, 89
main board schematic.....110
map, system memory.....68
maps, zero page memory.....74
Memory (RAM), Random Access.....3
Memory (ROM), Read-Only.....3
memory configuration, RAM.....70
memory map, system.....68
memory maps, zero page.....74
memory pages.....68
memory, examining.....41
memory, changing.....43
memory, moving.....44
memory, comparing.....46
memory, RAM.....68, 95
memory, ROM.....72, 94
microprocessor features.....88
microprocessor, 6502.....3, 88
Mini-Assembler instructions.....66
Mini-Assembler prompt (!).....50
Mini-Assembler, Apple.....49
Mini-Assembler, leaving the.....50
mode, the text video.....14

mode, the Low-Res video.....17
mode, the High-Res video.....19
mode, inverse text.....32
mode, normal text.....32
modes, addressing.....66
modes, graphics.....11
modification, Eurapple 50Hz.....10
Monitor command loops.....56
Monitor commands, summary of.....59
Monitor prompt (*).....40
Monitor ROM RESET.....38
Monitor ROM listing.....Appendix C
Monitor ROM.....25
Monitor special locations.....65
Monitor subroutines, some useful.....61
Monitor, entering the.....40
moving memory.....44
multiplexer, RAM address.....96

-- N --

normal text mode.....32
Normal/Inverse control values.....32
notation, hexadecimal.....40
number, random.....33

-- O --

one (system stack), page.....69
one-bit ("flag") inputs.....24, 25, 78
output cursor.....30
output jacks, video.....97
output subroutine, COUT standard.....30
output, utility strobe.....25
outputs, annunciator.....23
outputs, strobe.....78
own commands, creating your.....57

-- P --

page memory maps, zero.....74
page one (system stack).....69
page zero.....69, 74
pages, screen.....12
pages, memory.....68
peripheral board I/O.....79
peripheral connector pinout.....106
peripheral connectors.....3, 105
peripheral slot I/O.....79
peripheral slot RAM.....82
peripheral slot ROM.....80
photo of the Apple.....2
picture buffer.....12
pinout, peripheral connector.....106

pinout, configuration block.....	71
pinout, ROM	95
pinout, RAM	96
pinouts, keyboard connector.....	103
power connector	104
power supply features.....	92
power supply schematic	93
power-up byte.....	2, 28, 92
programming suggestions, I/O	80
programs, running machine language	48
programs, listing machine language.....	48
programs, debugging	51
PROM, peripheral card.....	80
PROM, expansion ROM or	84
PROMs, 2716 type	94
prompt (*), Monitor.....	40
prompt (!), Mini-Assembler.....	50
prompting characters	33
prompting, input.....	32
pushbutton inputs, single-bit.....	78
 -- R --	
RAM address multiplexer	96
RAM configuration blocks.....	70
RAM memory configuration	70
RAM memory	68, 95
RAM pinout	96
RAM, peripheral slot.....	82
random access memory (RAM)	3
random number	33
RDKEY standard input subroutine	32
reading from cassette tape	47
reading the keyboard.....	6
read-only memory (ROM)	3
reentering BASIC.....	34, 54
registers, 6502 internal.....	53, 81
relationships, timing signals and.....	91
RESET cycle, the	36
RESET, Autostart ROM.....	36
RESET, Monitor ROM.....	38
return character	30
retype key	34
review of the keyboard.....	4, 100
Revision 0 boards.....	3, 26
Revision 1 board	26
ROM listing, Autostart.....	Appendix C
ROM listing, Monitor.....	Appendix C
ROM memory	72, 94
ROM pinout	95
ROM RESET, Autostart.....	36
ROM RESET, Monitor.....	38
ROM special locations, Autostart	37
ROM, Autostart	25
ROM, Monitor	25
ROM, peripheral slot.....	80
ROM or PROM, expansion	84
running machine language programs	48
 -- S --	
saving to cassette tape	46
schematic, keyboard	101
schematic, power supply.....	93
schematic, main board.....	110
screen format.....	11
screen format, Text	16
screen format, High-Res	21
screen format, Low-Res	18
screen pages.....	12
screen soft switches	12
screen, the text	16
screen, the Low-Res	18
screen, the High-Res	21
set, 6502 instruction	Appendix A
setting up the Apple	2
signals and relationships, timing	91
single-bit pushbutton inputs	78
slot I/O, peripheral	79
slot RAM, peripheral.....	82
slot ROM, peripheral	80
soft entry vector	37
soft switches	12, 79, 98
soft switches, screen	12
speaker connector	105
special locations, list of	Appendix B
special locations, video display	13
special locations, input/output	25
special locations, text window	31
special locations, Autostart ROM	37
special locations, Monitor	65
special locations, keyboard	6
special locations, I/O	79
stack), page one (system	69
standard input subroutine, RDKEY	32
standard output subroutine, COUT	30
start, cold	36
start, lukewarm	36
start, warm	36
STEP and TRACE	26, 51
Stop-List feature, the	26, 30
strobe output, utility	25
strobe outputs	78
strobe, clearing the keyboard	6
subroutine, COUT standard output	30
subroutine, RDKEY standard input	32
subroutines, some useful Monitor	61

suggestions, I/O programming	80
summary of Monitor commands	59
supply features, power.....	92
supply schematic, power.....	93
supply, the Apple power	2, 28, 92
switches, soft.....	12, 79, 98
switches, screen soft.....	12
switches, toggle.....	79
switches, COUT, KEYIN	83
switches, CSW/KSW	83
system memory map.....	68
(system stack), page one	69
system timing	90

-- T --

tape, saving to cassette	46
tape, reading from cassette.....	47
text mode, inverse.....	32
text mode, normal.....	32
text screen, the.....	11, 16
text video mode, the	14
text window special locations.....	31
text window, the.....	31
timing signals and relationships.....	91
timing, system	90
toggle switches.....	79
TRACE, STEP and.....	26, 51

-- U --

"USER 1" jumper.....	99
useful Monitor subroutines, some.....	61
utility strobe output.....	25

-- V --

values, Normal/Inverse control.....	32
varieties of Apples.....	25
vector, soft entry	37
video connector	9
video connector, auxiliary	9
video display	9
video display special locations	13
video generator, the	96
video mode, the text	14
video mode, the Low-Res	17
video mode, the High-Res	19
video output jacks.....	97

-- W --

warm start.....	36
window special locations, text	31

window, the text.....	31
-----------------------	----

-- Y --

your own commands, creating.....	57
----------------------------------	----

-- Z --

zero page memory maps.....	74
zero, page	69, 74

INDEX OF FIGURES

Figure 1. Map of the Text screen.....	16
Figure 2. Map of the Low-Res mode	18
Figure 3. Map of the High-Res screen	21
Figure 4. Cursor-moving escape codes.....	35
Figure 5. System Memory Map.....	68
Figure 6. Memory Configurations	71
Figure 7. Configuration Block Pinouts	71
Figure 8. Expansion ROM Enable circuit.....	85
Figure 9. \$CFXX decoding	85
Figure 10. The Apple Main Board.....	89
Figure 11. Timing Signals.....	91
Figure 12. Power Supply Schematic	93
Figure 13. ROM Pinout.....	95
Figure 14. RAM Pinouts	96
Figure 15. Auxiliary Video Connector	98
Figure 16. Game I/O Connector Pinout.....	100
Figure 17. Keyboard Schematic Drawing	101
Figure 18. Keyboard connector Pinout.....	103
Figure 19. Power Connector	104
Figure 20. Speaker Connector	105
Figure 21. Peripheral Connector Pinout.....	106
Figure 22. Main Board Schematic.....	110-115

INDEX OF PHOTOS

Photo 1. The Apple II	2
Photo 2. The Apple Power Supply	3
Photo 3. The Apple Keyboard.....	6
Photo 4. The Video Connectors	10
Photo 5. Eurapple jumper pads.....	11
Photo 6. The Apple Character Set.....	14
Photo 7. The Game I/O Connector.....	23
Photo 8. The USER 1 Jumper.....	99

INDEX OF TABLES

Table 1. Keyboard Special Locations.....	6
Table 2. Keys and their ASCII codes	7
Table 3. The ASCII Character Set.....	8
Table 4. Video Display Memory Ranges	12
Table 5. Screen Soft Switches.....	13
Table 6. Screen Mode Combinations.....	13
Table 7. ASCII Screen Character Set.....	15
Table 8. Low-Resolution Colors.....	17
Table 9. Announcer Special Locations	24
Table 10. Input/Output Special Locations	25
Table 11. Text Window Special Locations.....	31
Table 12. Normal/Inverse Control Values.....	32
Table 13. Autostart ROM Special Locations....	37
Table 14. Page Three Monitor Locations	65
Table 15. Mini-Assembler Address Formats...	66
Table 16. RAM Organization and Usage	69
Table 17. ROM Organization and Usage	72
Table 18. Monitor Zero Page Usage	74
Table 19. Applesoft II Zero Page Usage	74
Table 20. DOS 3.2 Zero Page Usage.....	75
Table 21. Integer BASIC Zero Page Usage.....	75
Table 22. Built-In I/O Locations	79
Table 23. Peripheral Card I/O Locations	80
Table 24. Peripheral Card PROM Locations....	81
Table 25. I/O Location Base Addresses.....	82
Table 26. I/O Scratchpad RAM Addresses.....	83
Signal Descriptions:	
Table 27. Timing	90
Table 28. Auxiliary Video Output	97
Table 29. Game I/O Connector	100
Table 30. Keyboard Connector	102
Table 31. Power Connector	104
Table 32. Speaker Connector	105
Table 33. Peripheral Connector	107ff

CAST OF CHARACTERS

!	33	
#	66	
\$.50, 66	
&	65	
*	33, 38, 40	
+	55	
-	55	
:	(colon)	43
.	(period)	41
<	45, 46
>	33
?	33
@	35
A	34
B	34
C	35
D	35
E	35
F	35
G	48
L	25, 35, 54
J	25, 35
K	25, 35
L	49
M	25, 35, 45
N54, 55
R	47
S	52
T	52
V	46
W	47
CTRL B	54
CTRL C	30, 40, 54
CTRL E	53
CTRL G (bell)	30
CTRL H (←)	30, 33, 34
CTRL J (line feed)	30
CTRL K	55
CTRL P	54
CTRL S26, 30
CTRL U (→)33, 34
CTRL X	33
CTRL Y57, 58
ESC25, 34
RETURN	30, 33, 43
[.....	8
\8, 33
I	33
^	50
_	8

W) F) F)

