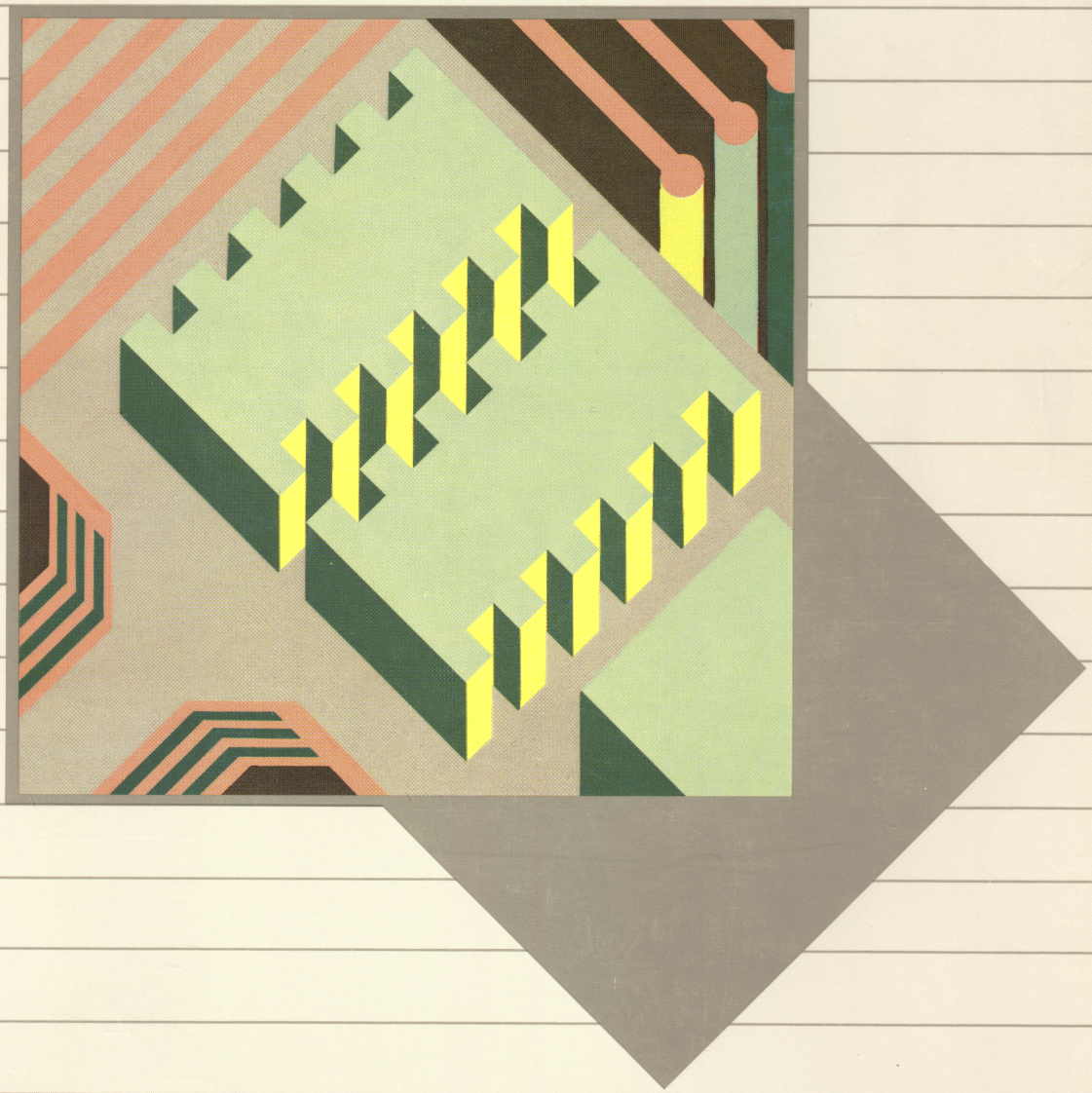
Apple *II*

Reference Manual

For *//e* Only



Notice

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

Disclaimer of All Warranties and Liabilities

Apple Computer, Inc. makes no warranties, either express or implied, with respect to this manual or with respect to the software described in this manual, its quality, performance, merchantability, or fitness for any particular purpose. Apple Computer, Inc. software is sold or licensed "as is." The entire risk as to its quality and performance is with the buyer. Should the programs prove defective following their purchase, the buyer (and not Apple Computer, Inc., its distributor, or its retailer) assumes the entire cost of all necessary servicing, repair, or correction and any incidental or consequential damages. In no event will Apple Computer, Inc. be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software, even if Apple Computer, Inc. has been advised of the possiblity of such damages. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

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

Simultaneously published in the U.S.A and Canada.

Warning

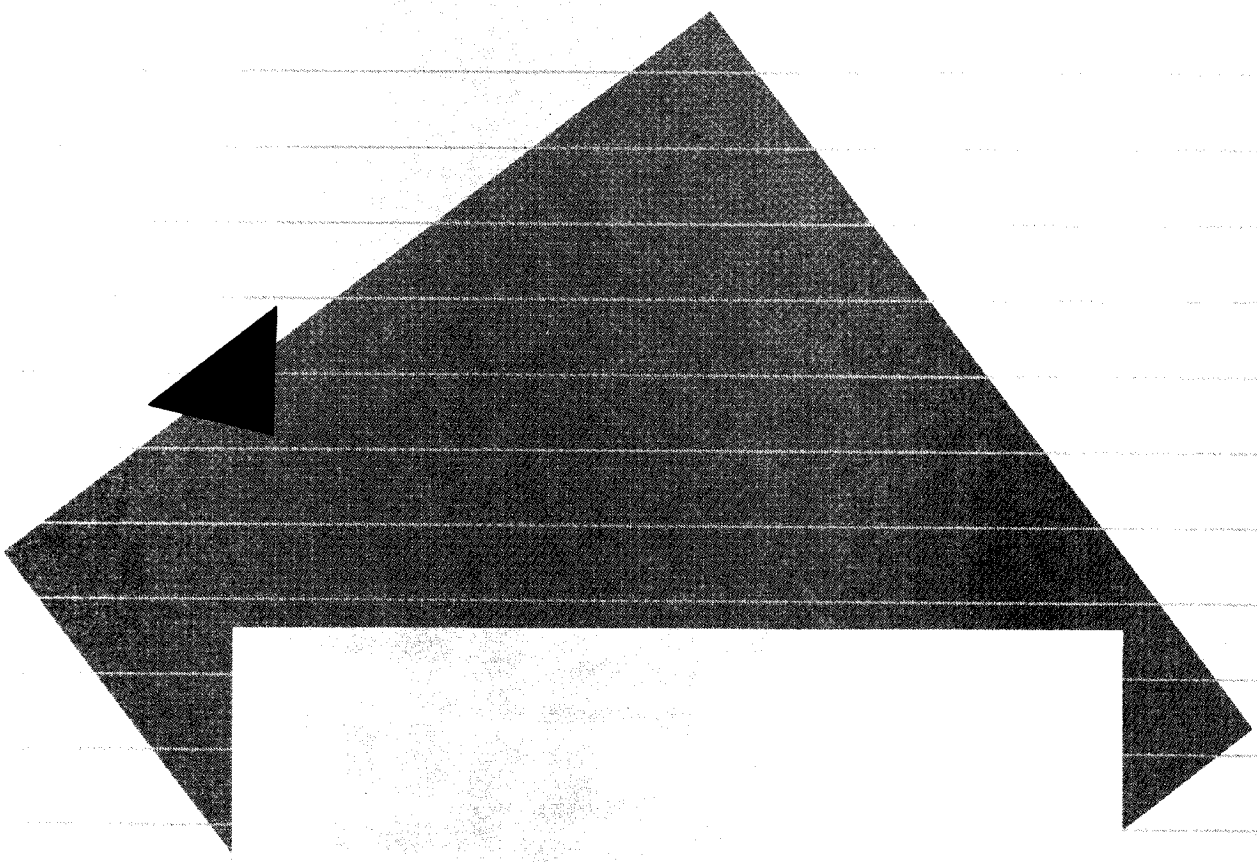
This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation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

Written by Allen Watson of the Apple PCSD Publications Department

Reorder Apple Product A2L2005 030-0357-B 1982

Apple *II*

Reference Manual



030-0357-B 1982

Radio and Television Interference

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

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

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

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

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

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

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

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

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

BLANK PAGE

Table of Contents

List of Figures

List of Tables

Foreword

This is the reference manual for the Apple IIe personal computer. It contains detailed descriptions of all of the hardware and firmware that make up the Apple IIe and provides the technical information that peripheral-card designers and programmers need. There is an Addendum, bound separately, that contains source listings of the built-in firmware.

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

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

This manual is designed to answer the question, What's inside the box? It describes the internal operation of the Apple IIe as completely as possible in a single volume. The criterion for deciding to include an item of information was whether it would help an assembly-language programmer or peripheral designer.

Contents of This Manual

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

- Introduction: Foreword and Chapter 1
- Use of built-in features: Chapters 2 and 3
- How the memory is organized: Chapter 4
- Information for programmers: Chapters 5 and 6

- Hardware implementation: Chapter 7
- Additional information: Appendices and Addendum

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

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

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

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

Chapter 6 describes the programmable features of the peripheral-card connectors and gives guidelines for their use.

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

Additional reference information appears in the appendices. Appendix A is the manufacturer's description of the 6502 instruction set.

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

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

Appendix D lists the differences between the Apple IIe and the earlier Apple II and Apple II Plus models and tells you which sections to look at for more information.

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

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

The Addendum to this manual contains the source listing of the Monitor firmware. You can refer to it to find out more about the operation of the Monitor subroutines listed in Appendix C.

Symbols Used in This Manual

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

```
Information that appears on the display screen is
set off by this screen-shaped outline.
```

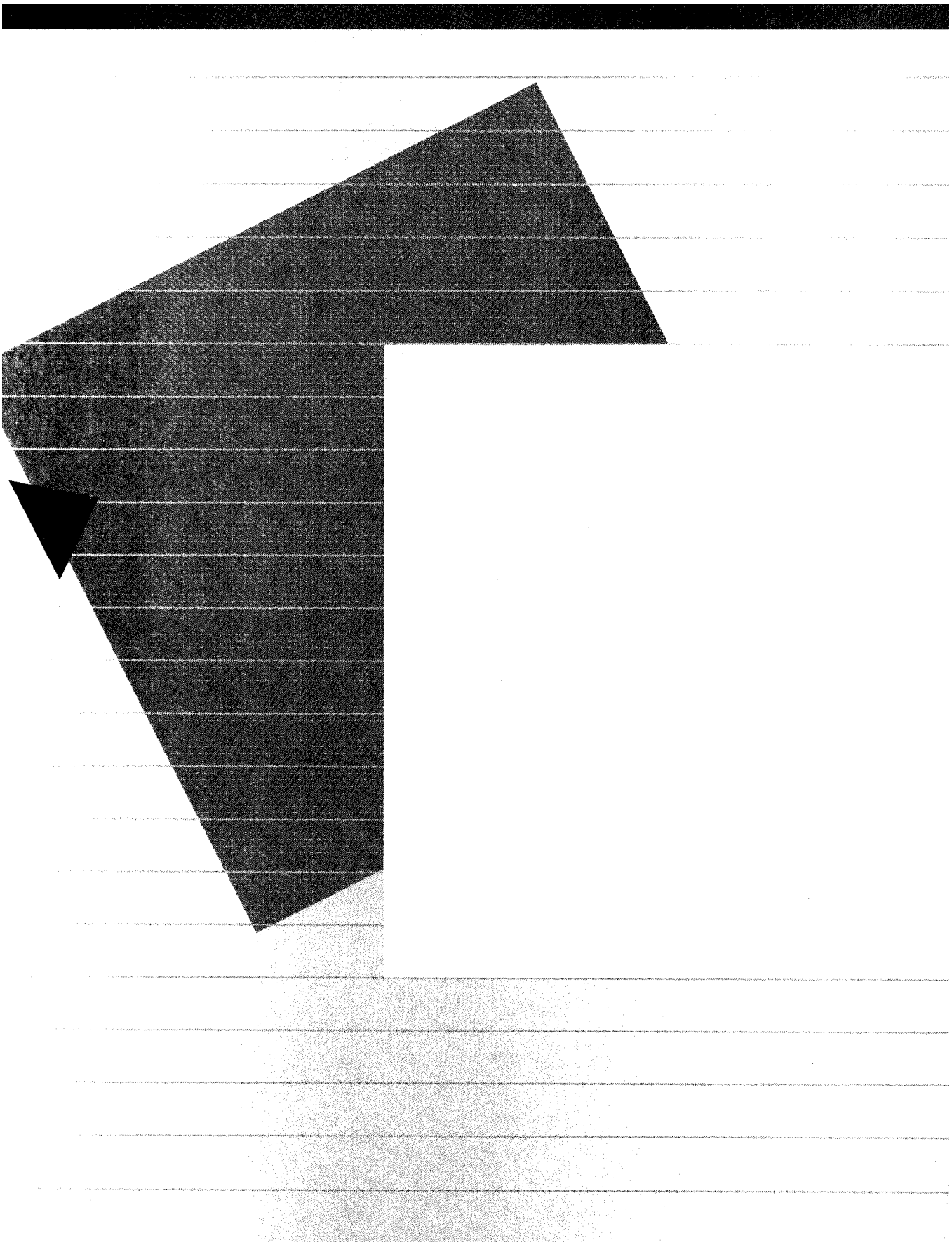
Warning
Important warnings appear in boxes like this.

Captions, definitions, and other short items appear in marginal glosses like this.

Information that is useful but is incidental to the text appears in grey boxes like this. You may want to skip over such boxes and return to them later.

Chapter 1

Introduction



Chapter 1

Introduction

This first chapter introduces you to the Apple IIe itself. It shows you what the inside looks like, identifies the major components that make up the machine, and tells you where to find information about each one. These major components are shown in the diagram in Figure 1.1.

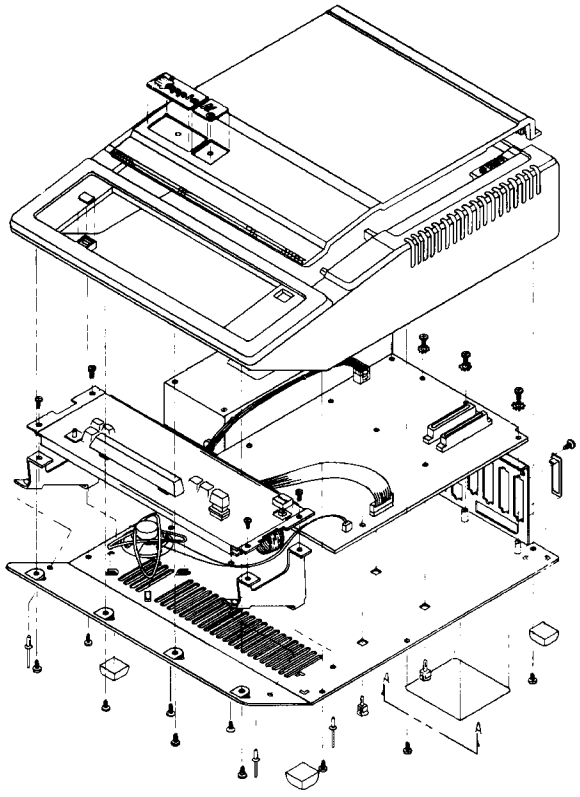
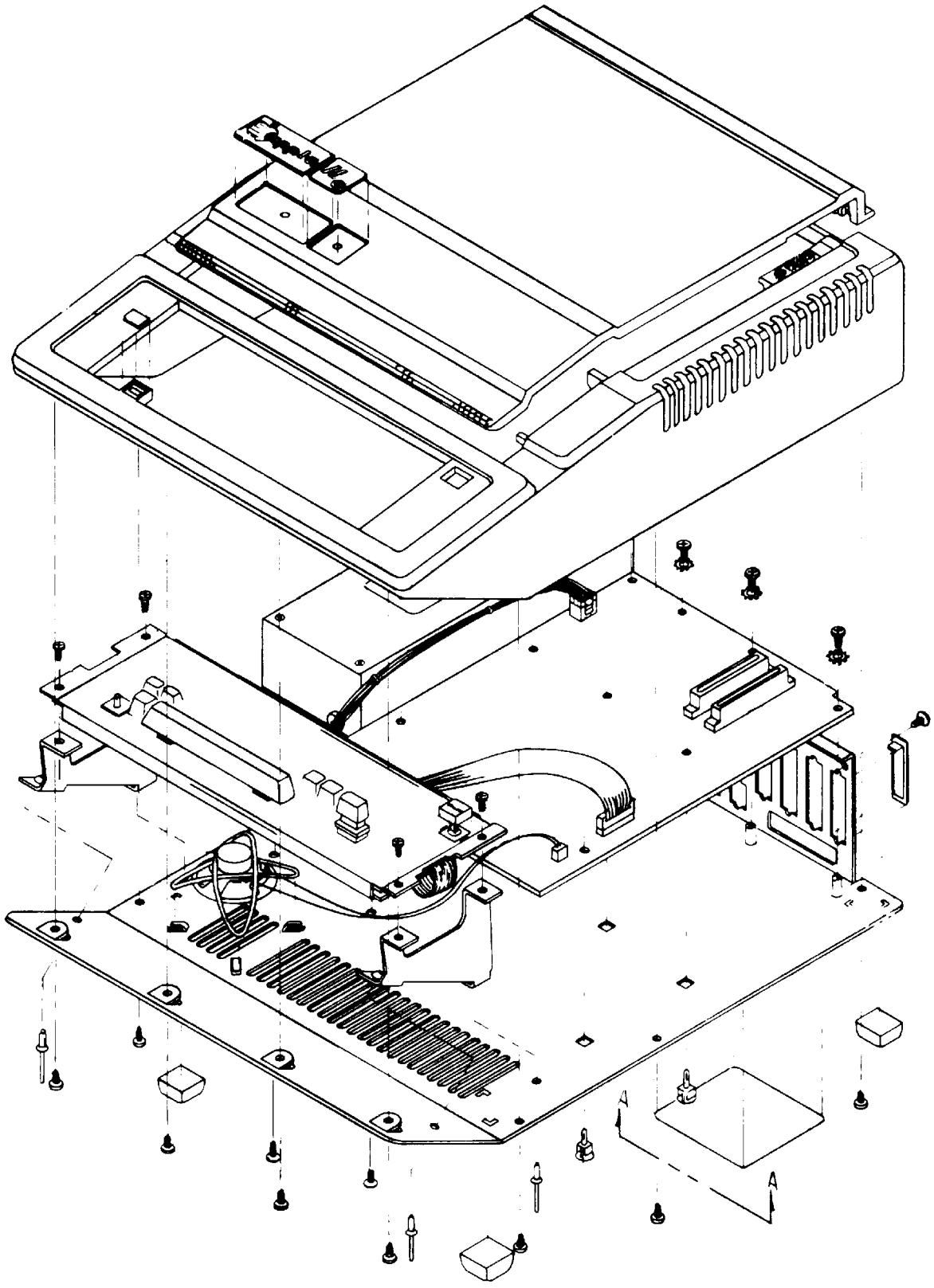


Figure 1-1 Exploded Diagram of the Apple IIe

Figure 1-1 Exploded Diagram of the Apple IIe



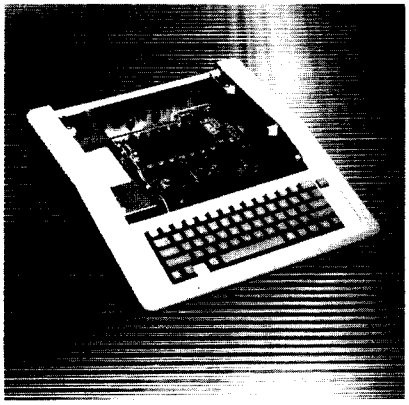
"A2_030-0357-B_1982_1-003a.pict" 326 KB 2002-10-27 dpi: 1000h x 1000v pix: 3009h x 4326v

Removing the Cover

Remove the cover of the Apple IIe by pulling up on the back edge until the fasteners on either side pop loose, then move the cover an inch or so towards the rear of the machine to free the front of the cover, as shown in Figure 1-2. What you will see is shown in Figure 1-3.

Figure 1-2 Removing the Cover

Figure 1-3 The Apple IIe with the Cover Off

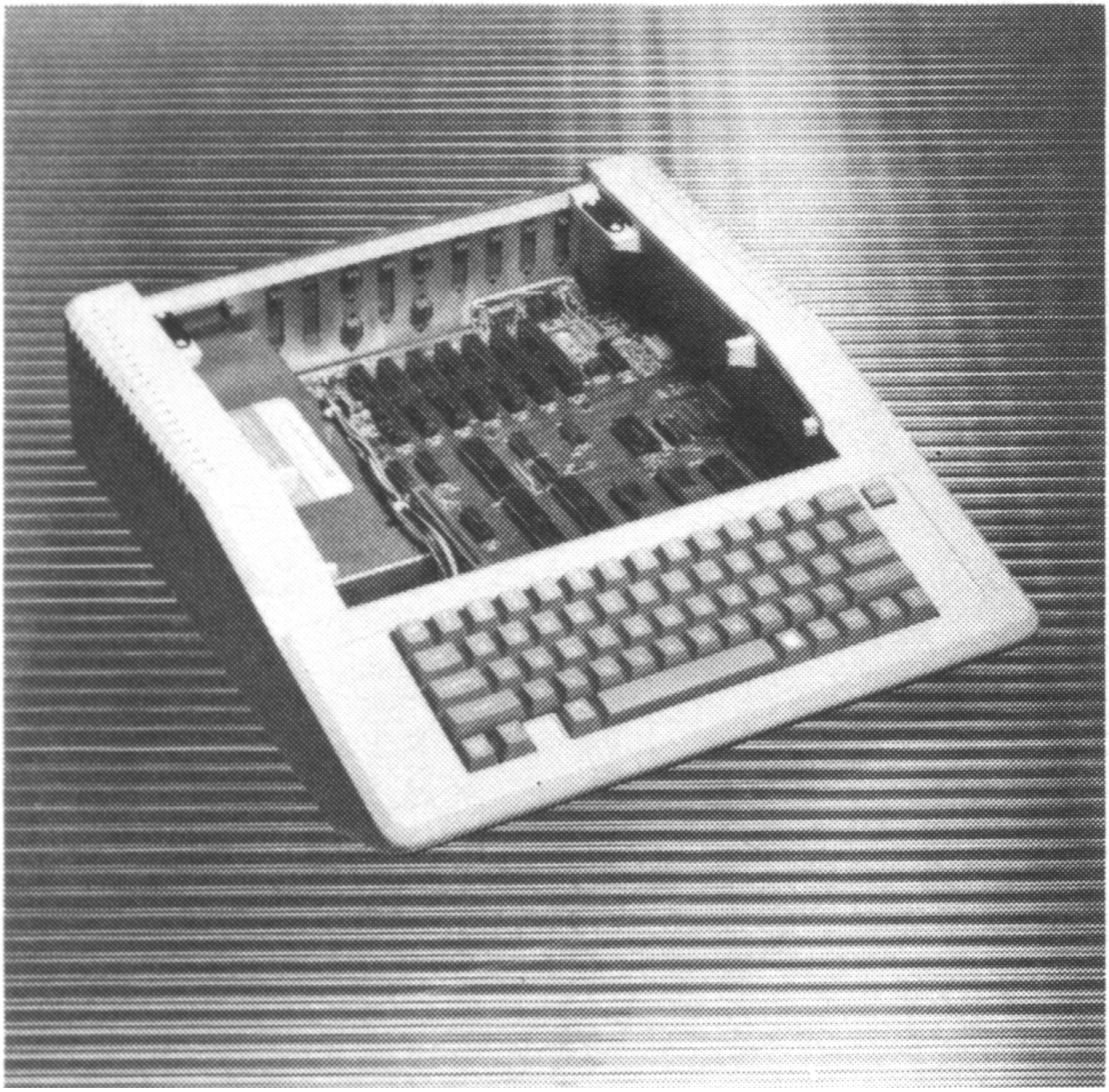


Warning

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

Figure 1-2 Removing the Cover

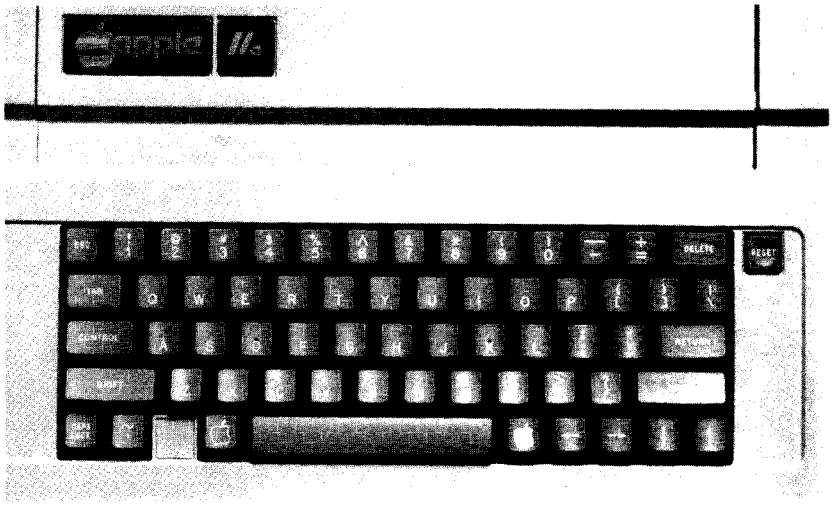
Figure 1-3 The Apple IIe with the Cover Off



The Keyboard

The keyboard is the Apple IIe's primary input device. As shown in Figure 1-4, it has a normal typewriter layout, upper- and lowercase, with all of the special characters in the ASCII character set. (ASCII stands for American Standard Code for Information Interchange.) The keyboard is fully integrated into the machine; its operation is described in the first part of Chapter 2. Firmware subroutines for reading the keyboard are described in Chapter 3.

Figure 1-4 Apple IIe Keyboard



The Speaker

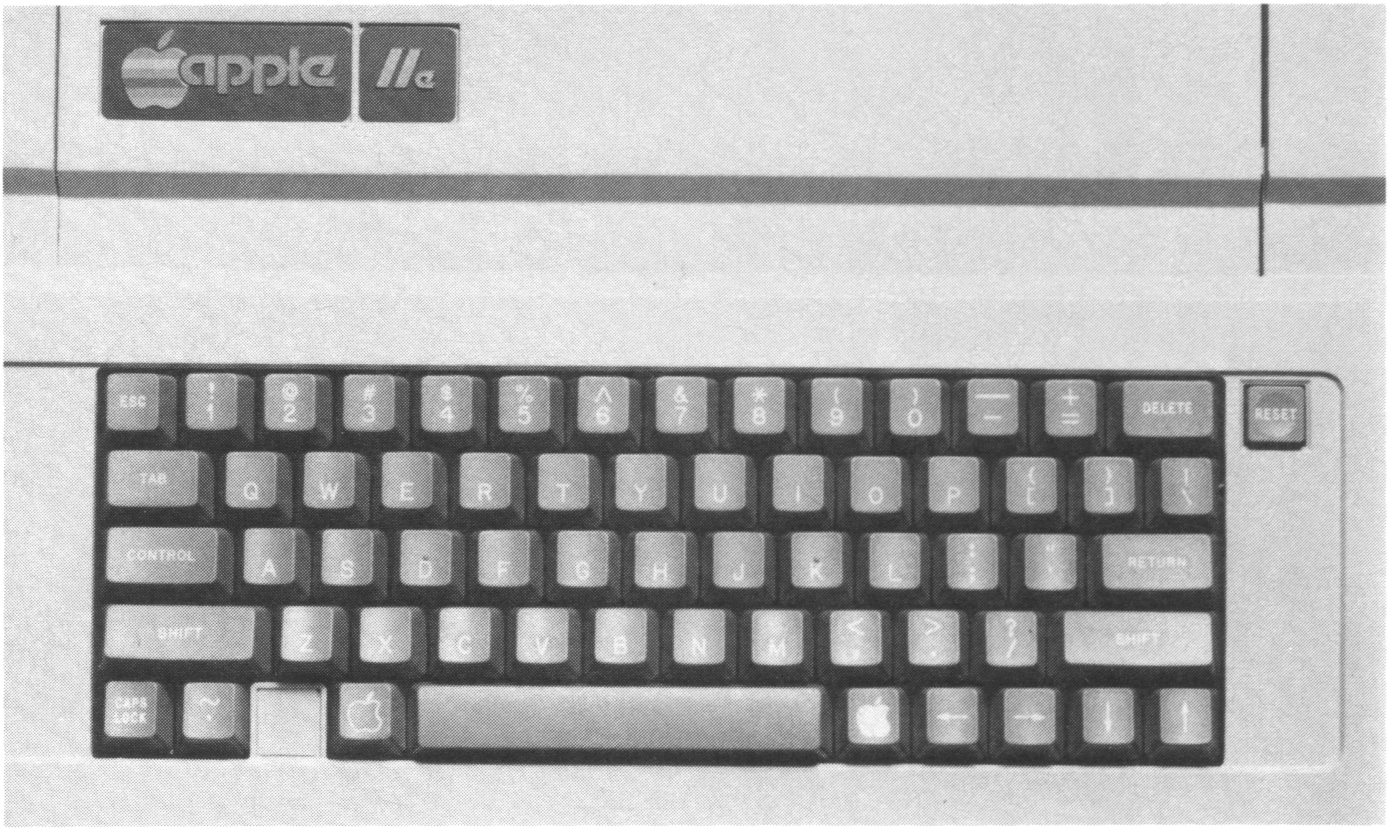
The Apple IIe has a small loudspeaker in the bottom of the case, as shown in Figure 1-1. The speaker enables Apple IIe programs to produce a variety of sounds that make the programs more useful and interesting. The way programs control the speaker is described in Chapter 2.

The Power Supply

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

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

Figure 1-4 Apple IIe Keyboard



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

The Circuit Board

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

Figure 1-5 Circuit Board

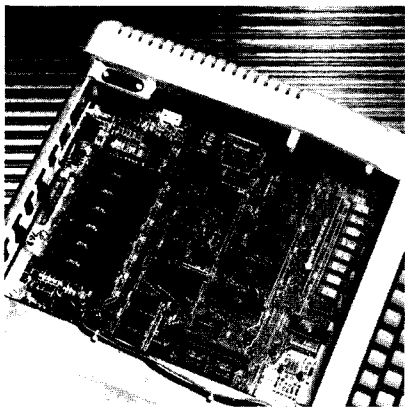


Figure 1-5 shows the main integrated circuits (ICs) in the Apple IIe. They are the central processing unit (CPU), the keyboard encoder and read-only memory (ROM), the two interpreter ROMs, and the custom integrated circuits: the Input Output Unit (IOU) and the Memory Management Unit (MMU).

The CPU is a 6502B microprocessor. The 6502B is a high-speed version of the 6502, which is an eight-bit microprocessor with a sixteen-bit address bus. It uses instruction pipelining for faster processing than comparable microprocessors. In the Apple IIe, the 6502B runs at 1 MHz and performs up to 500,000 eight-bit operations per second. The specifications of the 6502B are given in Chapter 7; the 6502 instruction set is given in Appendix A.

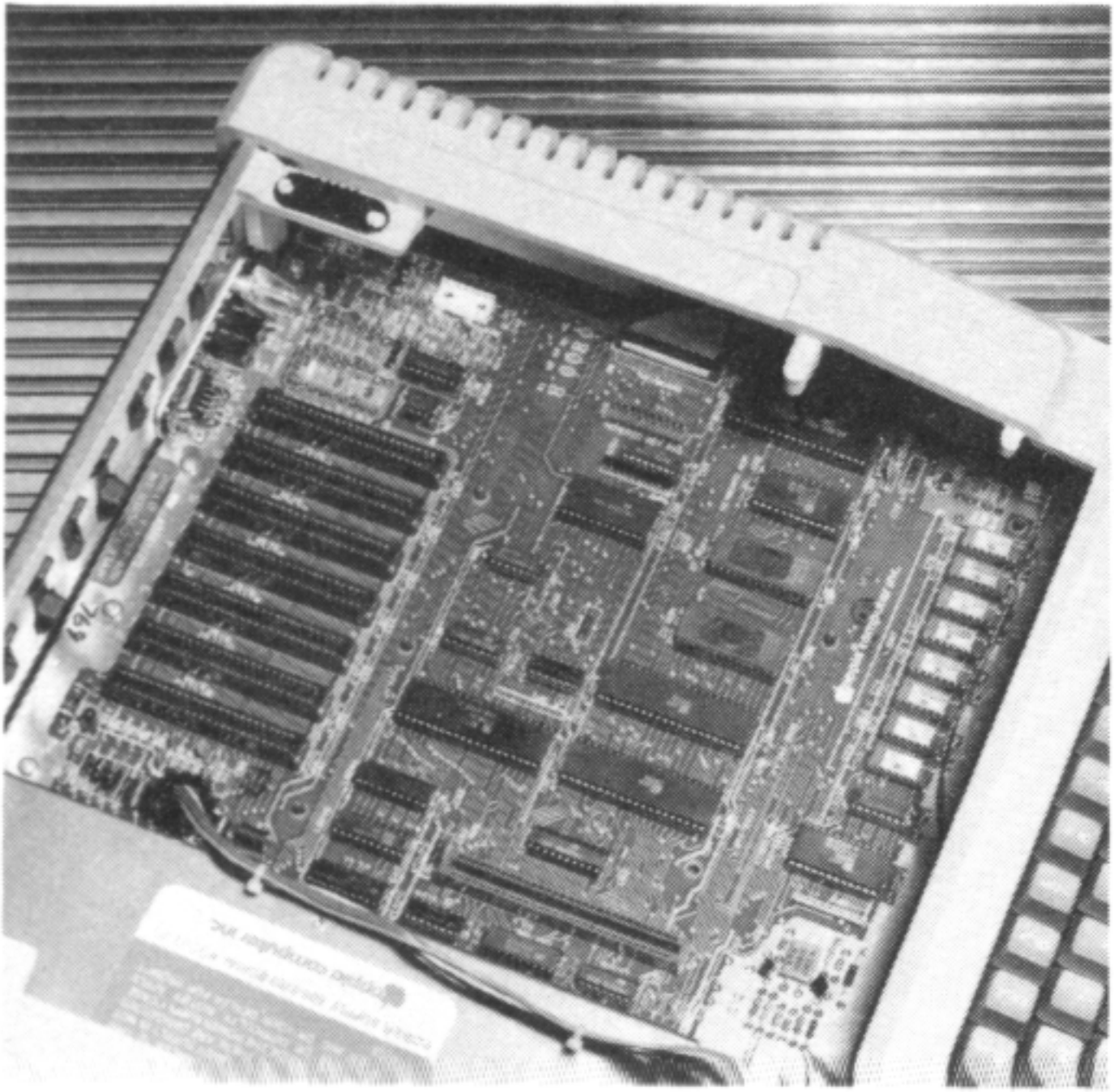
The keyboard is decoded by an AY-3600-type integrated circuit and a read-only memory (ROM). These devices are described in Chapter 7.

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

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

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

Figure 1-5 Circuit Board

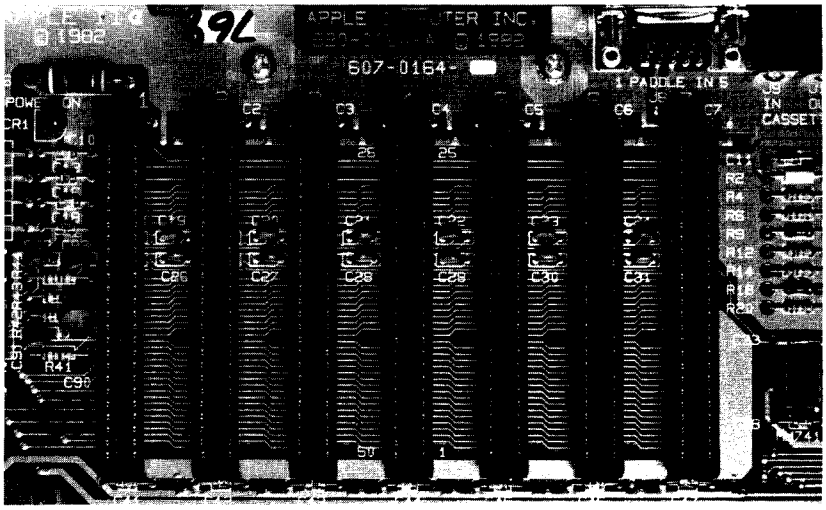


"A2_030-0357-B_1982_1-006a.pict" 4021 KB 2002-10-27 dpi: 1000h x 1000v pix: 2082h x 2401v

Connectors on The Circuit Board

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

Figure 1-6 Expansion Slots



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

Figure 1-6 Expansion Slots

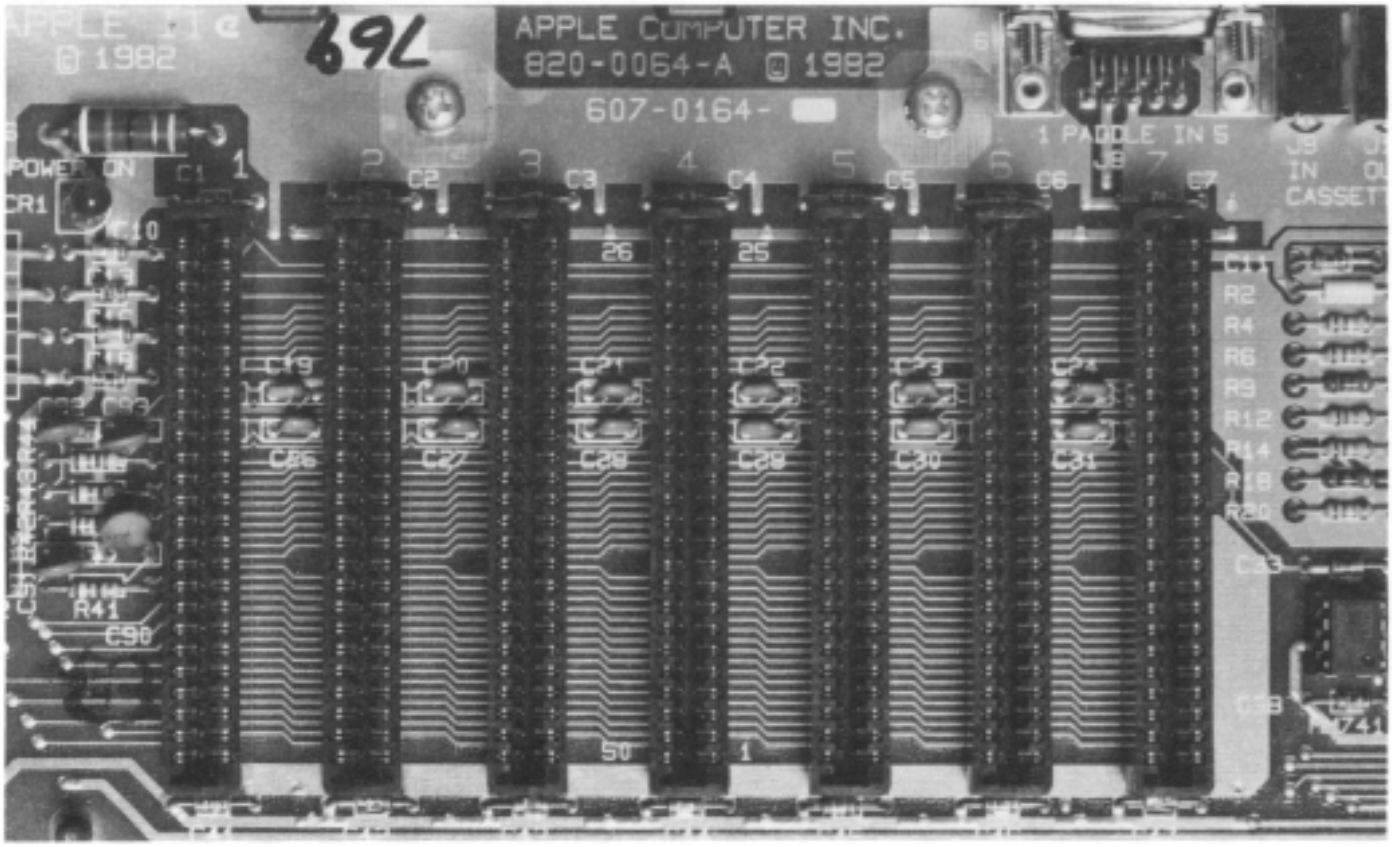
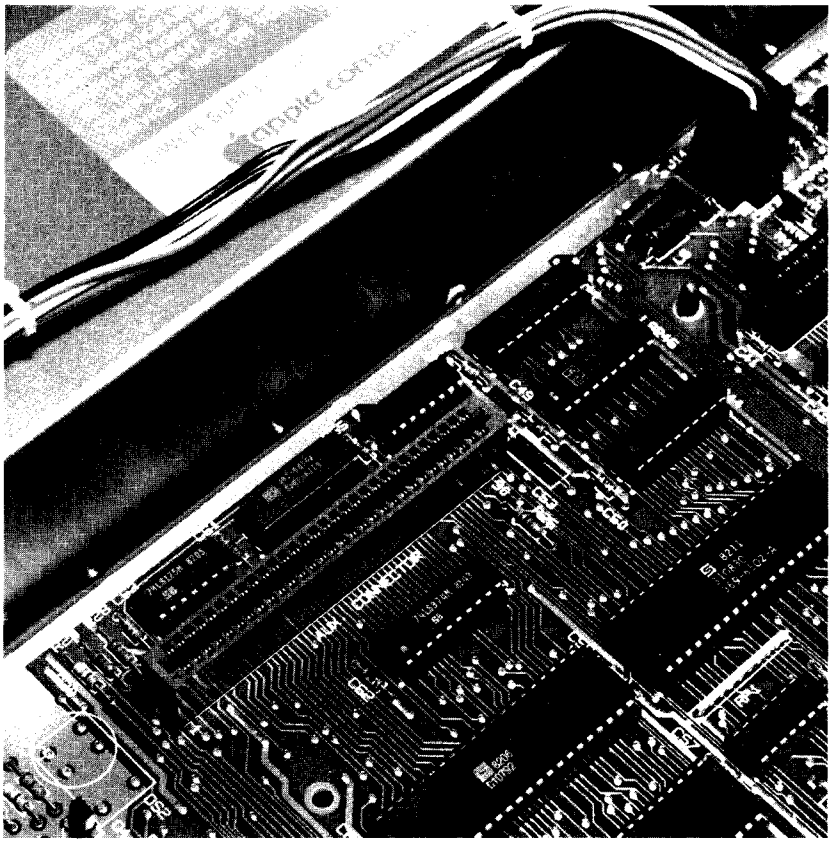


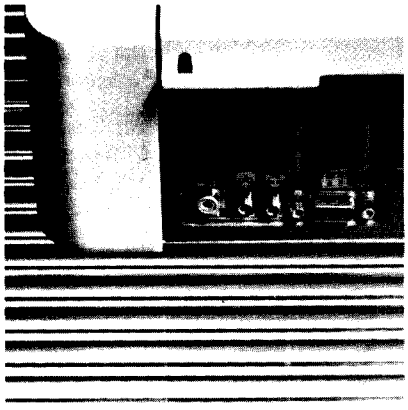
Figure 1-7 Auxiliary Slot



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

Connectors on the Back Panel

Figure 1-8 Back Panel Connectors



The back of the Apple IIe has two miniature phone jacks for connecting a cassette recorder, an RCA-type jack for a video monitor, and a 9-pin D-type miniature connector for the hand controls, as shown in Figure 1-8. In addition to these, there are spaces for additional connectors used with the peripheral cards installed in the Apple IIe. The installation manuals for the peripheral cards contain instructions for installing the peripheral connectors.

Figure 1-7 Auxiliary Slot

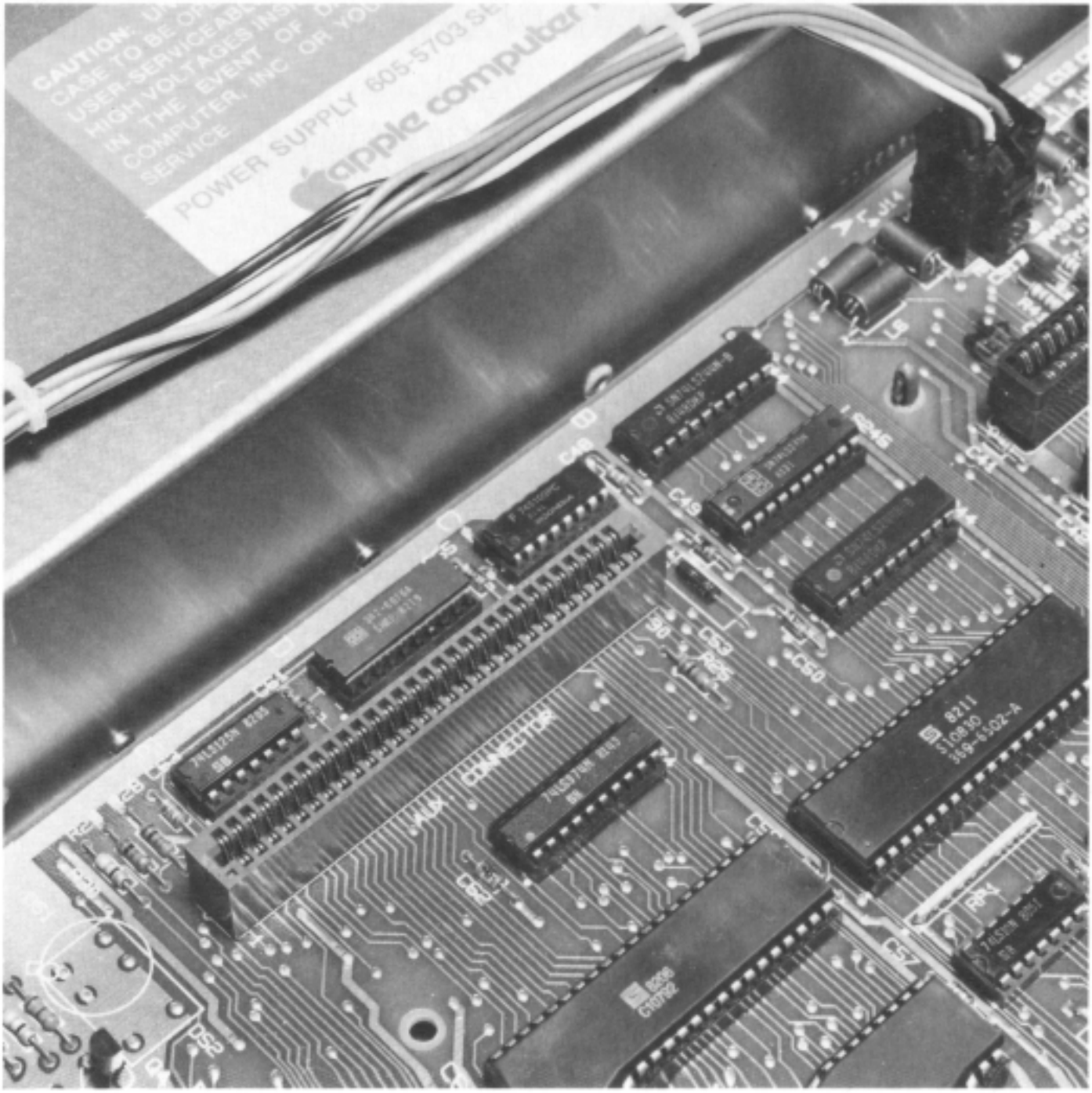
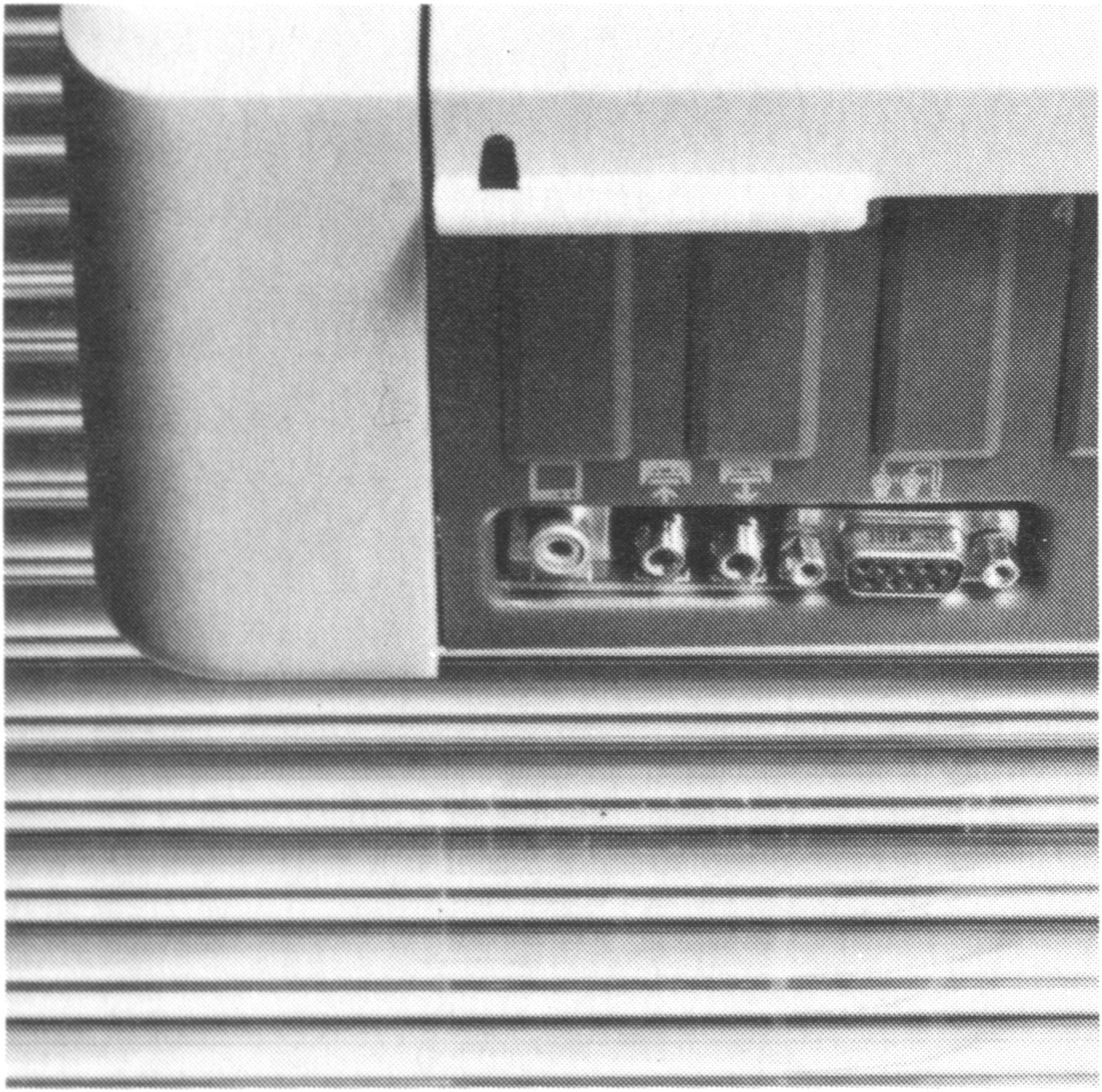


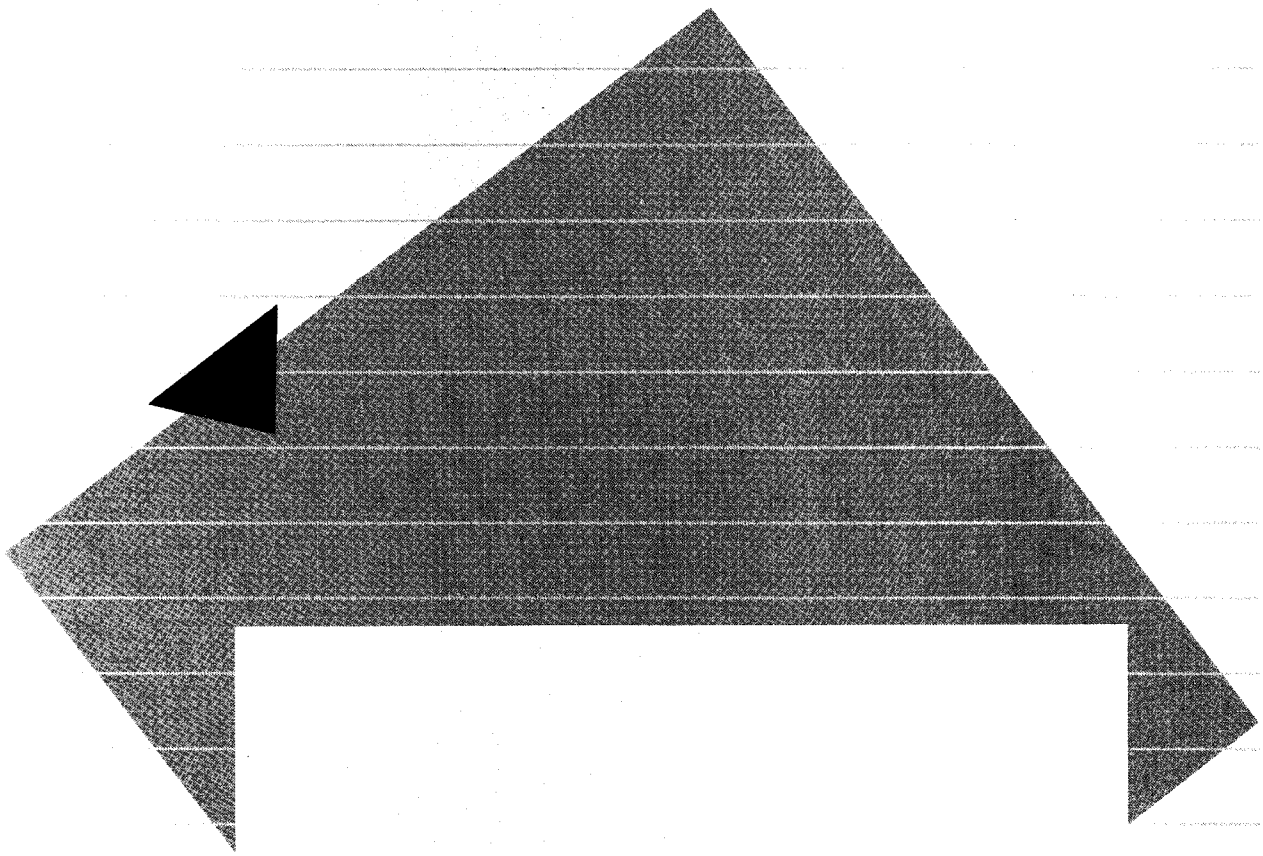
Figure 1-8 Back Panel Connectors



"A2_030-0357-B_1982_1-008b.pict" 5724 KB 2002-10-27 dpi: 1200h x 1200v pix: 2518h x 2861v

Chapter 2

Built-in I/O Devices



"A2_030-0357-B_1982_1-010.pict" 582 KB 2002-10-27 dpi: 600h x 600v pix: 4064h x 5287v

Chapter 2

Built-in I/O Devices

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

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

At the lowest level, programs use the built-in I/O devices by reading and writing to dedicated memory locations. This chapter lists these locations for each I/O device. It also gives the locations of the internal soft-switches that select the different display modes of the Apple IIe. For descriptions of the built-in I/O hardware, refer to Chapter 7.

This method of input and output — loading and storing directly to specific locations in memory — is not the only method you can use. For many of your programs, it may be more convenient to call the built-in I/O routines stored in the Apple IIe's firmware. These firmware routines are described in Chapter 3.

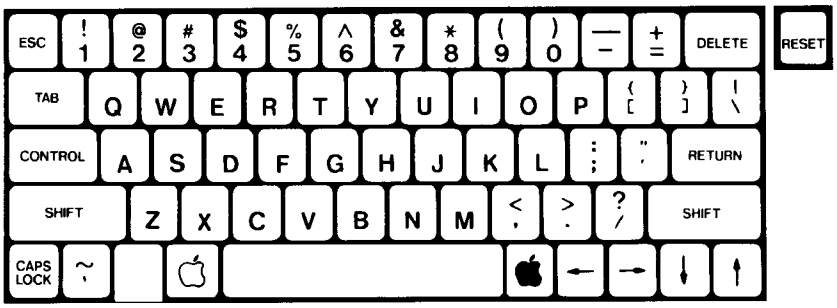
The Keyboard

The primary input device of the Apple IIe is its built-in keyboard. The keyboard has 63 keys and is similar to the keyboard of a typewriter. The Apple IIe keyboard has automatic repeat on all keys: hold the key down to repeat. It also has N-key rollover, which means that you can hold down any number of keys while typing another. Of course, if you hold the keys down much longer

than the length of time you would hold them down during normal typing, the automatic-repeat function will start repeating the last key you pressed.

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

Figure 2-1 The Keyboard



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

Table 2-1 Apple IIe Keyboard Specifications

Number of keys:	63
Character encoding:	ASCII
Number of codes:	128
Special keys:	CONTROL, SHIFT, CAPS LOCK, ESC, RESET, LEFT-ARROW, RIGHT-ARROW, UP-ARROW, DOWN-ARROW, OPEN-APPLE, SOLID-APPLE
Rollover:	N-key
Electrical interface:	AY-5-3600 keyboard encoder

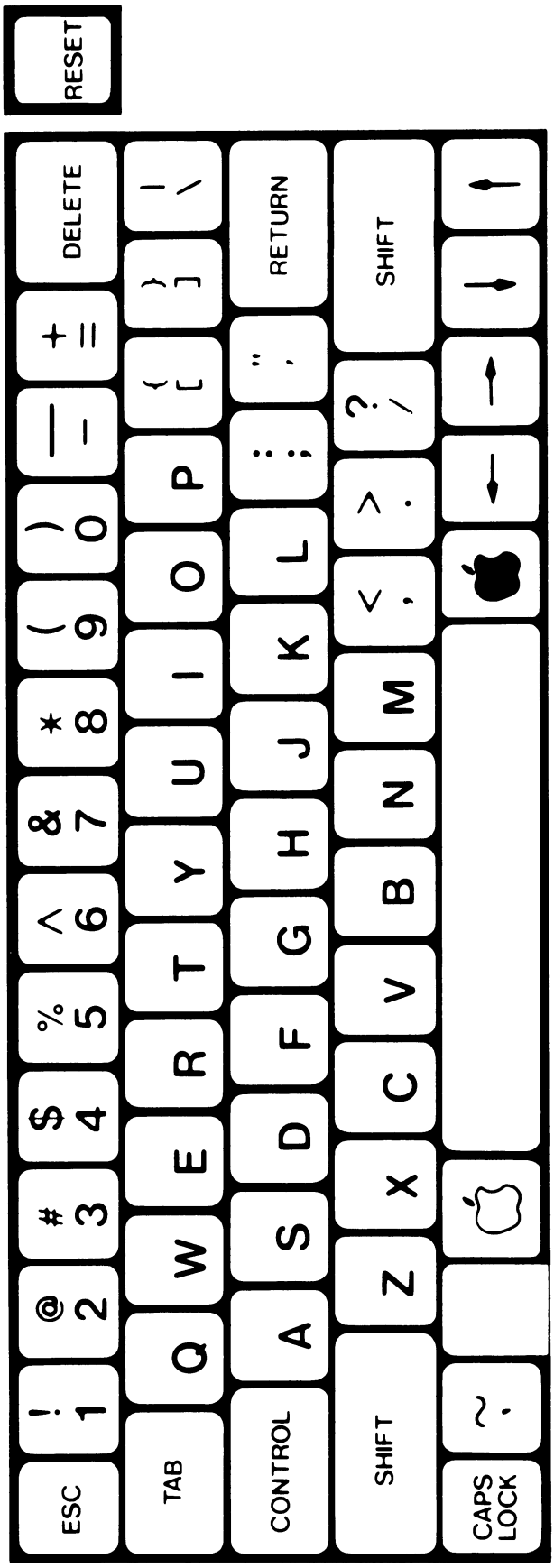


Figure 2-1 The Keyboard

Four special keys, CONTROL, SHIFT, CAPS LOCK, and ESC change the codes generated by the other keys. The CONTROL key is similar to the ASCII CTRL key. Three other keys have special functions: the RESET key, and two keys marked with apples, one outlined, or open, and one solid, or closed. Pressing the RESET key with the CONTROL key depressed resets the Apple IIe, as described in Chapter 4. The Apple keys are connected to the one-bit game inputs, described later in this chapter.

The electrical interface between the Apple IIe and the keyboard is a ribbon cable with a 26-pin connector. This cable carries the keyboard signals to the encoding circuitry on the main board. A complete description of the electrical interface to the keyboard is given in Chapter 7.

Reading the Keyboard

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

Table 2-2 Keyboard Memory Locations

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

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

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

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

The strobe bit is the high-order bit of the keyboard-data byte. After any key has been pressed, the strobe bit is high. It remains high until you reset it by reading or writing at the clear-strobe

Table 2-3a Keys and ASCII Codes

Codes are shown here in hexadecimal; to find the decimal equivalents, use Table 2-4.

Key	Normal	Control	Shift	Both
DELETE	7F	7F	7F	7F
LEFT-ARROW	08	08	08	08
TAB	09	09	09	09
DOWN-ARROW	0A	0A	0A	0A
UP-ARROW	0B	0B	0B	0B
RETURN	0D	0D	0D	0D
RIGHT-ARROW	15	15	15	15
ESC	1B	1B	1B	1B
space	20	20	20	20
' "	27	27	22	22
, <	2C	2C	3C	3C
- _	2D	2D	5F	1F
. >	2E	2E	3E	3E
/?	2F	2F	3F	3F
0)	30	30	29	29
1!	31	31	21	21
2@	32	00	40	00
3#	33	33	23	23
4$	34	34	24	24
5%	35	35	25	25
6 ^	36	1E	5E	1E
7&	37	37	26	26
8*	38	38	2A	2A
9(	39	39	28	28
: ;	3B	3B	3A	3A
= +	3D	3D	2B	2B
[{	5B	1B	7B	1B
\ |	5C	1C	7C	1C

location. This location is a combination flag and switch; the flag tells whether any key is down, and the switch clears the strobe bit. The switch function of this memory location is called a *soft switch* because it is controlled by software. In this case, it doesn't matter whether the program reads or writes, and it doesn't matter what data the program writes: the only action that occurs is the resetting of the keyboard strobe. Similar soft switches, described later, are used for controlling other functions in the Apple IIe.

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

Table 2-3b Keys and ASCII Codes, continued

Codes are shown here in hexadecimal; to find the decimal equivalents, use Table 2-4.

Key	Normal	Control	Shift	Both
]}	5D	1D	7D	1D
`~	60	60	7E	7E
A	61	01	41	01
B	62	02	42	02
C	63	03	43	03
D	64	04	44	04
E	65	05	45	05
F	66	06	46	06
G	67	07	47	07
H	68	08	48	08
I	69	09	49	09
J	6A	0A	4A	0A
K	6B	0B	4B	0B
L	6C	0C	4C	0C
M	6D	0D	4D	0D
N	6E	0E	4E	0E
O	6F	0F	4F	0F
P	70	10	50	10
Q	71	11	51	11
R	72	12	52	12
S	73	13	53	13
T	74	14	54	14
U	75	15	55	15
V	76	16	56	16
W	77	17	57	17
X	78	18	58	18
Y	79	19	59	19
Z	7A	1A	5A	1A

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

Table 2-4 The ASCII Character Set

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

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

Another key that doesn't generate a code is the RESET key, located at the upper-right corner of the keyboard; it is connected directly to the Apple IIe's circuits. Pressing the RESET key with the CONTROL key depressed normally causes the system to stop whatever program it's running and restart itself. This restarting process is called the reset routine, and it is described in Chapter 4.

Two more special keys are marked with two versions of the Apple logo and located on either side of the space bar. The key with the outlined apple is the OPEN-APPLE key; the key with the solid-color apple is the SOLID-APPLE key. These keys are connected to the one-bit game inputs, which are described later in this chapter. Pressing them in combination with the CONTROL and RESET keys causes the built-in firmware to perform special reset and self-test cycles, described with the reset routine in Chapter 4.

The Video Display Generator

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

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

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

The specifications for the video display are summarized in Table 2-5.

Table 2-5 Video Display Specifications

Display modes:	40-column text 80-column text with optional card Low-resolution color graphics High-resolution color graphics
Text capacity:	24 lines by 40 columns 24 lines by 80 columns with optional card
Character set:	96 ASCII characters (uppercase and lowercase)
Display formats:	Normal, Inverse, Flashing
Low-resolution graphics:	16 colors, 40 horizontal by 48 vertical
High-resolution graphics:	6 colors, 280 horizontal by 192 vertical

The video signal produced by the Apple IIe is NTSC-compatible composite color video. It is available at three places: the RCA-type phono jack on the back of the Apple IIe, the single Molex-type pin on the main circuit board near the back on the right side, and one of the group of four Molex-type pins in the same area on the main board. Use the RCA-type phono jack to connect a video monitor or an external video modulator; use the Molex pins to connect the type of video modulator that fits inside the Apple IIe case. For a full description of the video signal and the connections to the Molex-type pins, refer to the section "Video Output Signals" in Chapter 7.

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

- Text, 24 lines of 40 characters
- Text, 24 lines of 80 characters (with optional card)
- Low-resolution graphics, 40 by 48, in 16 colors
- High-resolution graphics, 280 by 192, in 6 colors

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

Either of the graphics displays can have four lines of text, either 40-column or 80-column, at the bottom of the screen. Graphics displays with text at the bottom are called *mixed-mode displays*.

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

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

Text Modes

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

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

Text Character Sets

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

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

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

The alternate character set sacrifices the flashing format for a complete inverse format. With the alternate character set, the Apple IIe can display uppercase letters, lowercase letters, numbers, and special characters in either normal format or inverse format.

You select the character set by means of the alternate-text soft switch, described below in the section "Display Mode Switching". Table 2-6 shows the character codes in decimal and hexadecimal for the Apple IIe primary and alternate character sets in normal, inverse, and flashing formats.

Table 2-6 The Display Character Sets

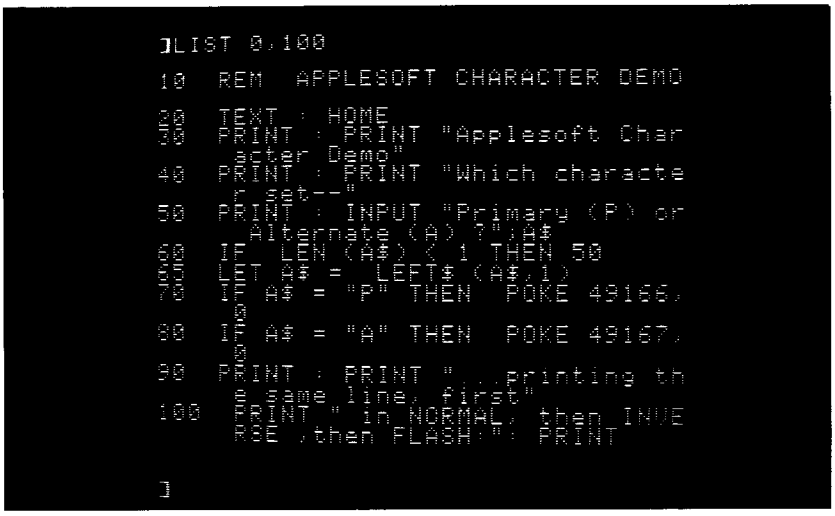
To identify particular characters and values, refer to Table 2-4.

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

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

40-column versus 80-column Text

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



```
]LIST 0,100

10  REM  APPLESOFT CHARACTER DEMO

20  TEXT : HOME
30  PRINT : PRINT "Applesoft Char
    acter Demo"
40  PRINT : PRINT "Which characte
    r set--"
50  PRINT : INPUT "Primary (P) or
    Alternate (A) ?";A$
60  IF LEN (A$) < 1 THEN 50
65  LET A$ = LEFT$ (A$,1)
70  IF A$ = "P" THEN  POKE 49165,
    0
80  IF A$ = "A" THEN  POKE 49167,
    0
90  PRINT : PRINT "...printing th
    e same line, first"
100  PRINT " in NORMAL, then INVE
     RSE ,then FLASH:": PRINT

]
```

Figure 2-2 40-column Text Display

Figure 2-2 40-column Text Display

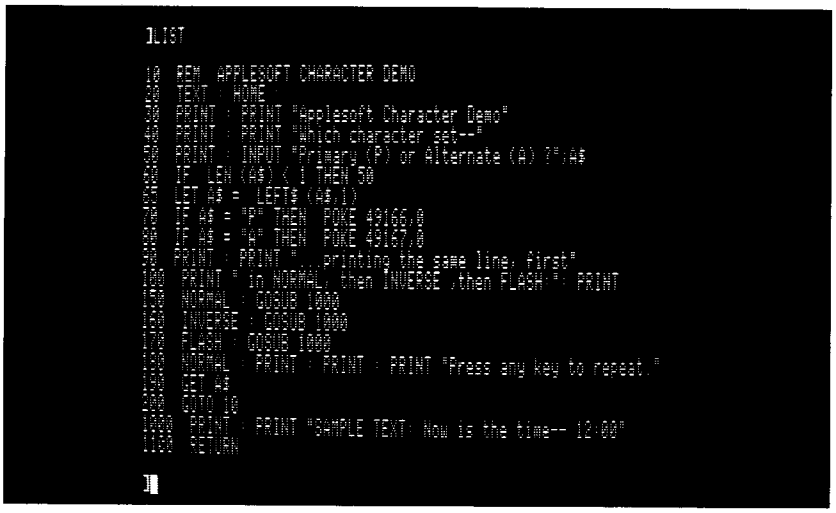


Figure 2-3 80-column Text Display

Graphics Modes

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

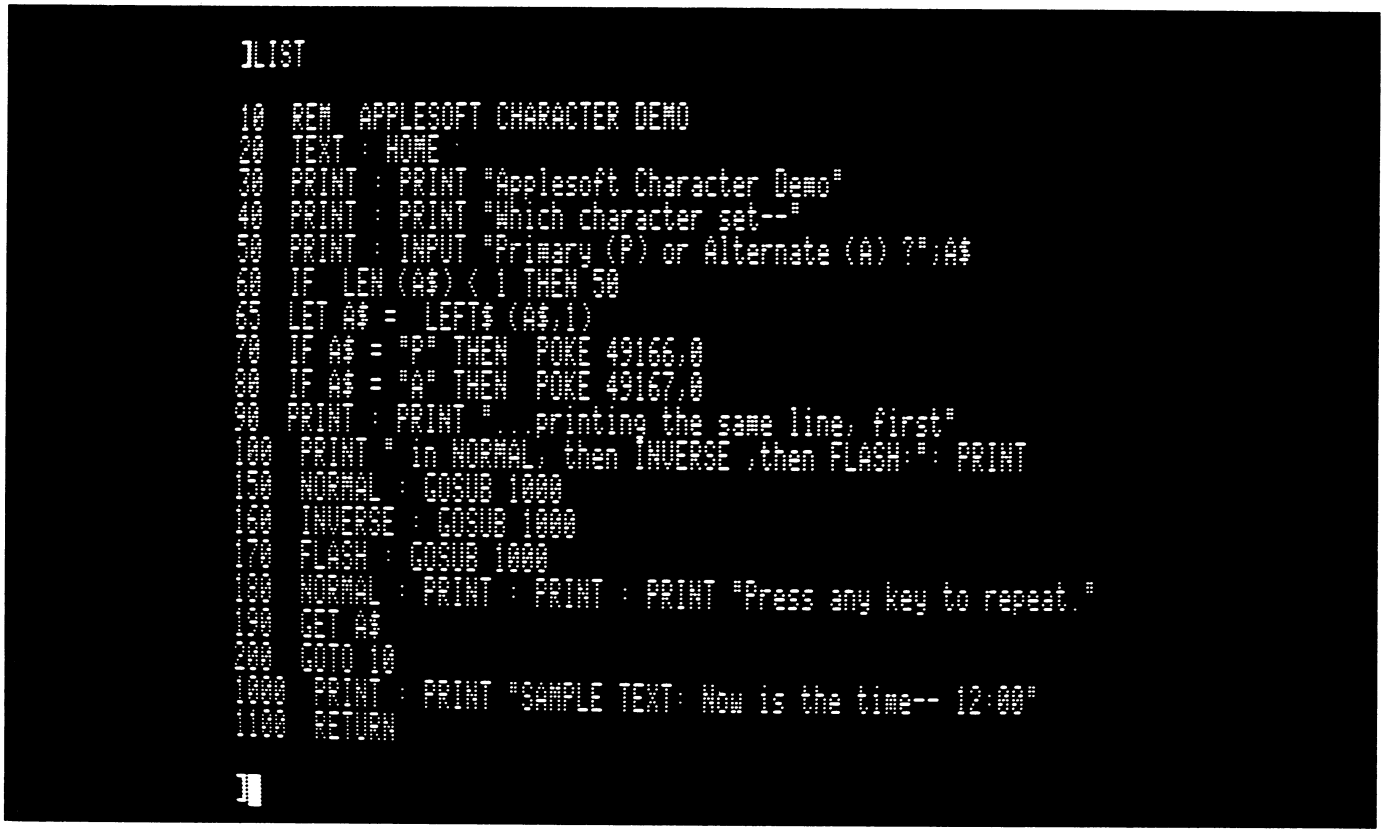
Low-resolution Graphics

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

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

Half a byte — four bits, or one nybble — is assigned to each graphics block. Each nybble can have a value from 0 to 15, and this value determines which one of sixteen colors appears on the

Figure 2-3 80-column Text Display



```
]LIST

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

]
```

screen. The colors and their corresponding nybble values are shown in Table 2-7. In each byte, the low-order nybble sets the color for the top block of the pair, and the high-order nybble sets the color for the bottom block. Thus, a byte containing the hexadecimal value $D8 produces a brown block atop a yellow block on the screen.

Table 2-7 Low-resolution Graphics Colors

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

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

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

High-resolution Graphics

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

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

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

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

On an NTSC color monitor or a color television set, a dot whose controlling bit is off (0) is black. If the bit is on, the dot will be white or a color, depending on its position, the dots on either side, and the setting of the high-order bit of the byte. Call the left-most column of dots column zero, and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control them are on, dots in even-numbered columns, 0, 2, 4, and so forth, are purple, and dots in odd-numbered columns are green — but only if the dots on either side are black. If two adjacent dots are both on, they are both white.

You select the other two colors, blue and orange, by turning the high-order bit (bit 7) of a data byte on (1). The colored dots controlled by a byte with the high-order bit on are either blue or orange: the dots in even-numbered columns are blue, and the dots in odd-numbered columns are orange — again, only if the dots on either side are black. Within each horizontal line of seven dots controlled by a single byte, you can have black, white, and

one pair of colors. To change the color of any dot to one of the other pair of colors, you must change the high-order bit of its byte, which affects the colors of all seven dots controlled by the byte.

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

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

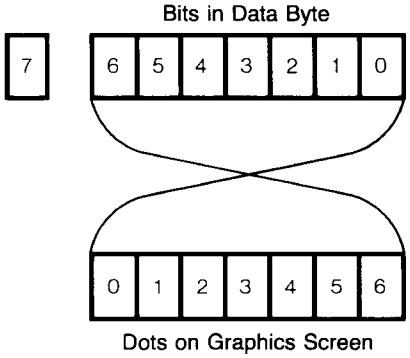
These rules are summarized in Table 2-8. The blacks and whites are numbered to remind you that the high-order bit is different.

Table 2-8 High-resolution Graphics Colors

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

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

Figure 2-4 High-resolution Display Bits



The peculiar behavior of the high-resolution colors reflects the way NTSC color television works. The dots that make up the Apple IIe video signal are spaced to coincide with the frequency of the color subcarrier used in the NTSC system. Alternating black and white dots at this spacing cause a color monitor or TV set to produce color, but two or more white dots together do not. For more details about the way the Apple IIe produces color on a TV set, see Chapter 7. For information about the way NTSC color television works, see the magazine articles listed in the bibliography.

Display Pages

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

The 40-column-text and low-resolution-graphics modes use two display pages of 1024 bytes each. These are called text Page 1 and Text Page 2, and they are located at 1024-2047 (hexadecimal $400-$7FF) and 2048-3071 ($800-$BFF) in main memory. Normally, only Page 1 is used, but you can put text or graphics data into Page 2 and switch displays instantly. Either page can be displayed as 40-column text, low-resolution graphics, or mixed-mode (four rows of text at the bottom of a graphics display).

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

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

Table 2-9 Video Display Page Locations

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

*Note: 80-column mode uses the 1024-byte page-1 locations in both main and auxiliary memory. The PAGE2 switch is used to select one or the other for storing data (see the section "Display Mode Switching").

Display Mode Switching

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

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

```
STA $C052
```

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

```
POKE 49234,0
```

The table gives the switch locations in three forms: hexadecimal, decimal, and negative decimal. You can use the hexadecimal values in your machine-language programs. Use the decimal values in PEEK or POKE commands in Applesoft BASIC; the negative values are for Integer BASIC.

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

Some of the soft switches in Table 2-10 are marked read or write. Those soft switches share their locations with the keyboard data and strobe functions. In the original Apple II, memory locations from $C000 to $C01F (49152 to 49183) were used only for the keyboard data and strobe functions. In the Apple IIe, these locations are used the same way, but only when you *read* to get data and *write* to clear the strobe. To perform the function shown in the table, use the operation listed there. Soft switches that are not marked may be accessed by either a read or a write. When writing to a soft switch, it doesn't matter what value you write; the action occurs when you address the location, and the value is ignored.

Table 2-10 Display Soft Switches

Name	Function	Location Hex	Location Decimal	Notes
ALTCHARSET	Alternate char. set on	$C00F	49167 -16369	Write
	Alternate char. set off	$C00E	49166 -16370	Write
	Read ALTCHARSET switch	$C01E	49182 -16354	Read
TEXT	Text mode on	$C051	49233 -16303	
	Text mode off (graphics)	$C050	49232 -16304	
	Read TEXT switch	$C01A	49178 -16358	Read
MIXED	Mixed-mode on	$C053	49235 -16301	1
	Mixed-mode off	$C052	49234 -16302	1
	Read MIXED switch	$C01B	49179 -16357	Read
PAGE2	Page 2 on	$C055	49237 -16299	2
	Page 2 off (Page 1)	$C054	49236 -16300	2
	Read PAGE2 switch	$C01C	49180 -16356	Read
HIRES	Hi-res mode on	$C057	49239 -16297	1
	Hi-res mode off	$C056	49238 -16298	1
	Read HIRES switch	$C01D	49181 -16355	Read
80COL	80-column display on	$C00D	49165 -16371	Write
	80-column display off	$C00C	49164 -16372	Write
	Read 80COL switch	$C01F	49183 -16353	Read
80STORE	Store in auxiliary memory	$C001	49153 -16383	Write, 3
	Store in main memory	$C000	49152 -16384	Write, 3
	Read 80STORE switch	$C018	49176 -16360	Read
VBL	Read vertical blanking	$C019	49177 -16359	Read, 4

(1) This mode is only effective when graphics-mode switch is ON.
(2) This switch has a different function when the 80-column text card's auxiliary text page is enabled for writing. Refer to the next section, "Addressing Display Pages Directly".
(3) This switch changes the function of the PAGE2 switch for addressing the auxiliary text memory on the extended 80-column text card. The next section describes how to do this.
(4) Reading this location returns the state of the vertical blanking signal VBL . The function of VBL is described in Chapter 7 in the section "Video Output Signals."

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

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

Addressing Display Pages Directly

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

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

The high-resolution graphics display is stored in much the same way as text, but there are eight times as many bytes to store, because eight rows of dots occupy the same space on the display as one row of characters. The subset consisting of all the first rows from the groups of eight is stored in the first 1024 bytes of the high-resolution display page. The subset consisting of all the second rows from the groups of eight is stored in the second 1024 bytes, and so on for a total of 8 times 1024, or

8192 bytes. In other words, each block of 1024 bytes in the high-resolution display page contains one row of dots out of every group of eight rows. The individual rows are stored in sets of three forty-byte rows, the same way as the text display.

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

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

To store display data on the 80-column text card, first turn on the 80STORE soft switch by writing to location 49153 (hexadecimal $C001 or complementary -16383). With 80STORE on, the page-select switch PAGE2 selects between the portion of the 80-column display stored in Page 1 of main memory and the portion stored in the 80-column text card memory. To select the 80-column text card, turn the PAGE2 soft switch on by reading or writing at location 49237. For more details about the way the displays are generated, see Chapter 7.

Figure 2-5 Map of 40-column Text Display

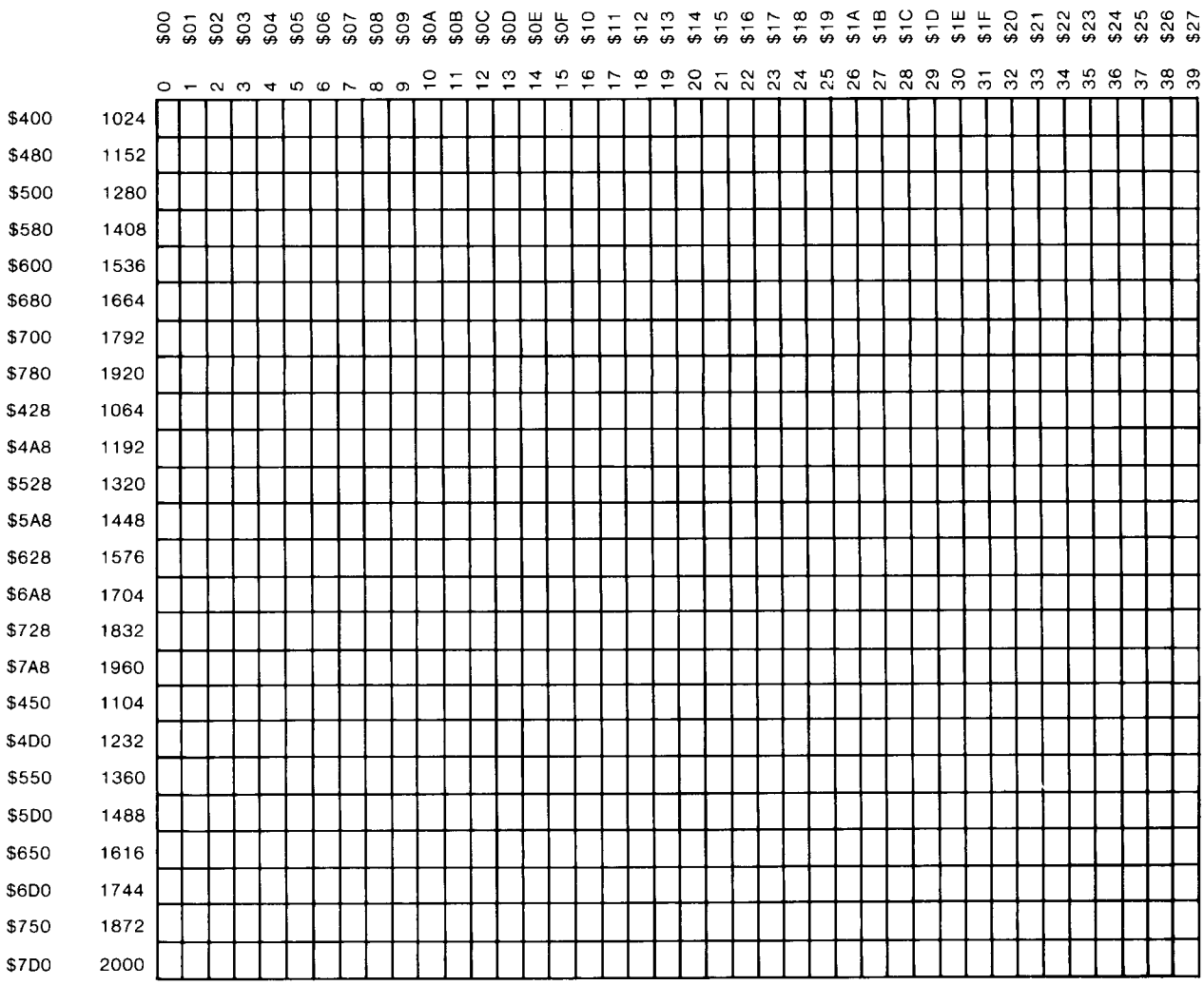


Figure 2-6 Map of 80-column Text Display

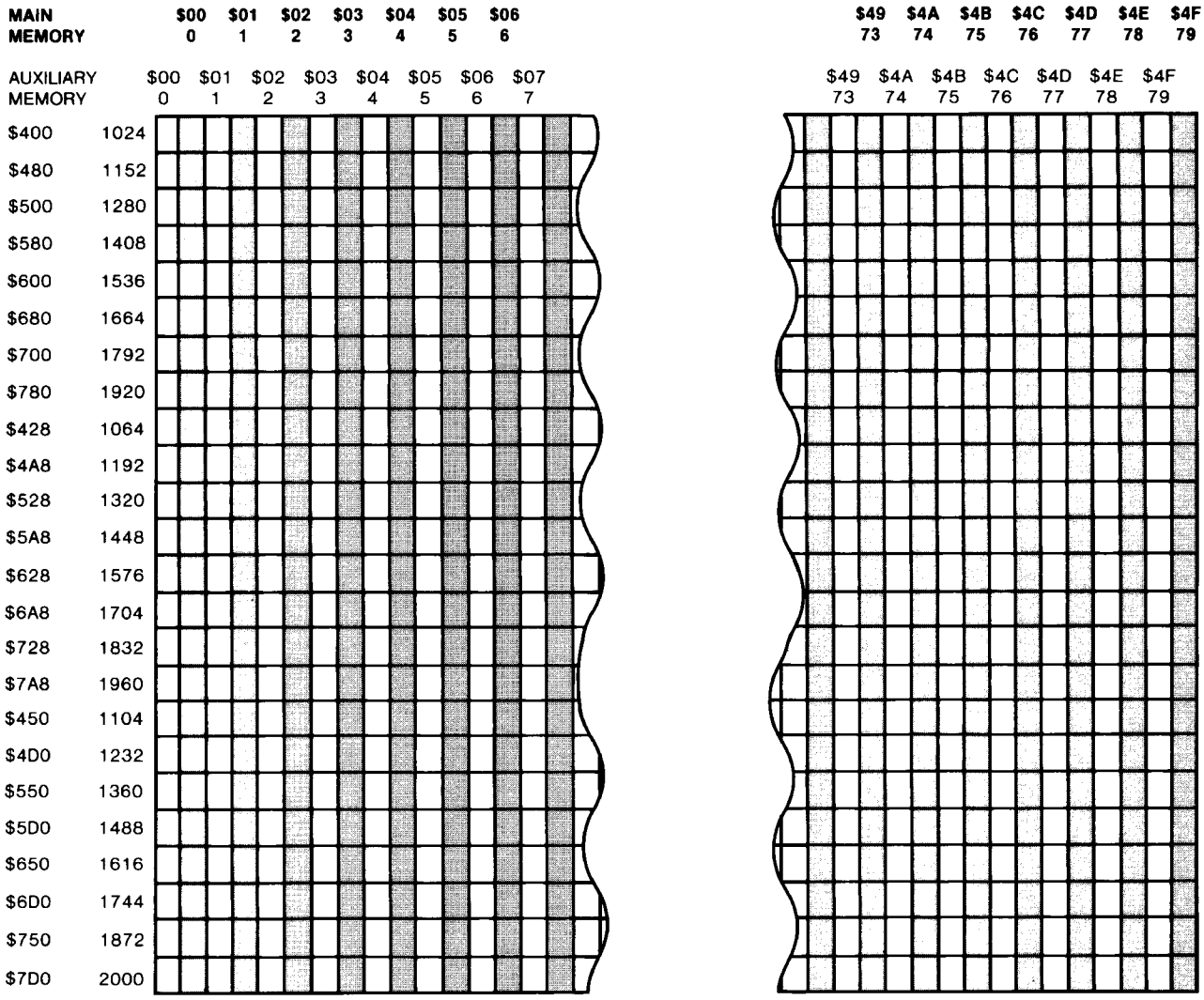


Figure 2-7 Map of Low-resolution Graphics Display

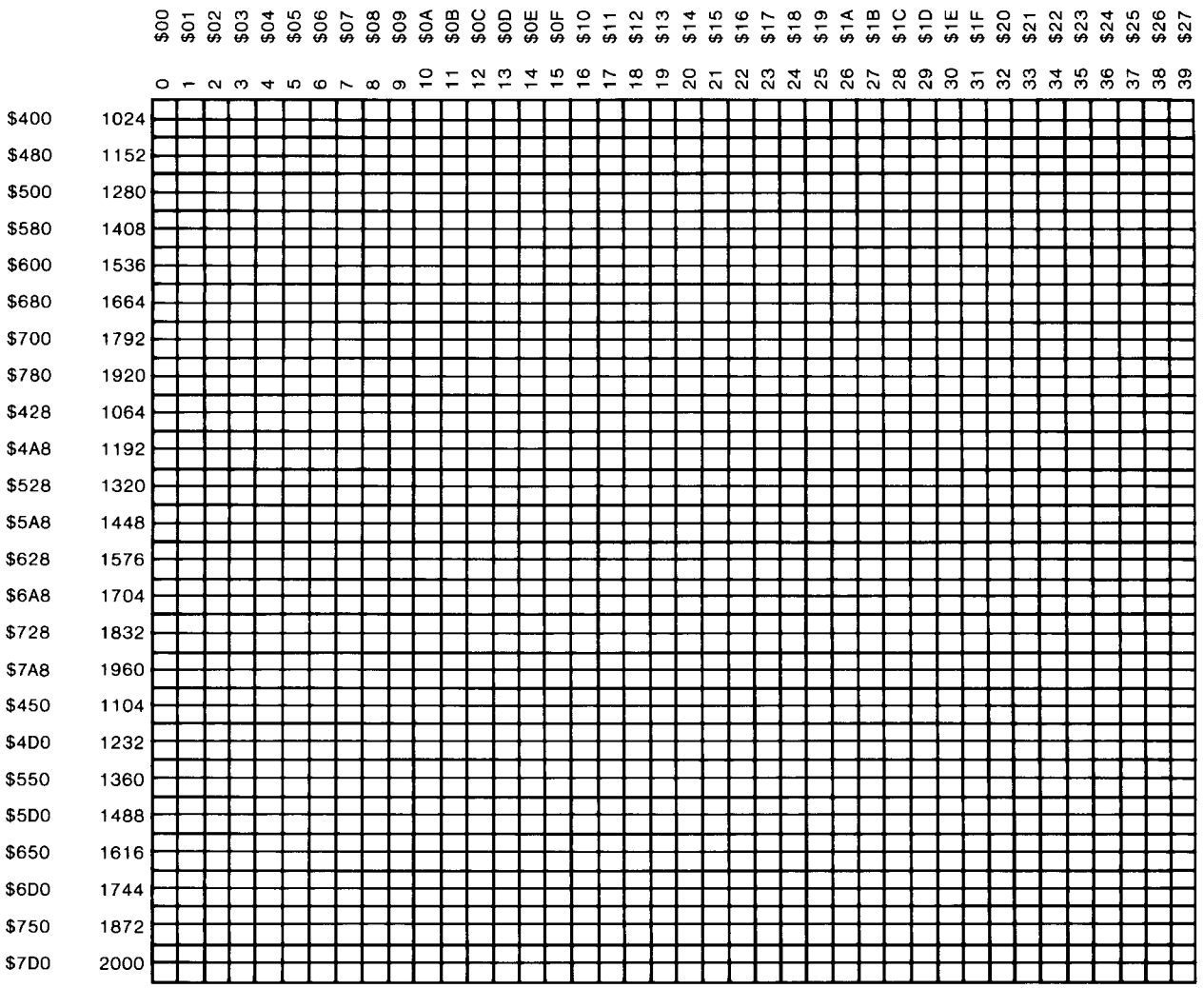
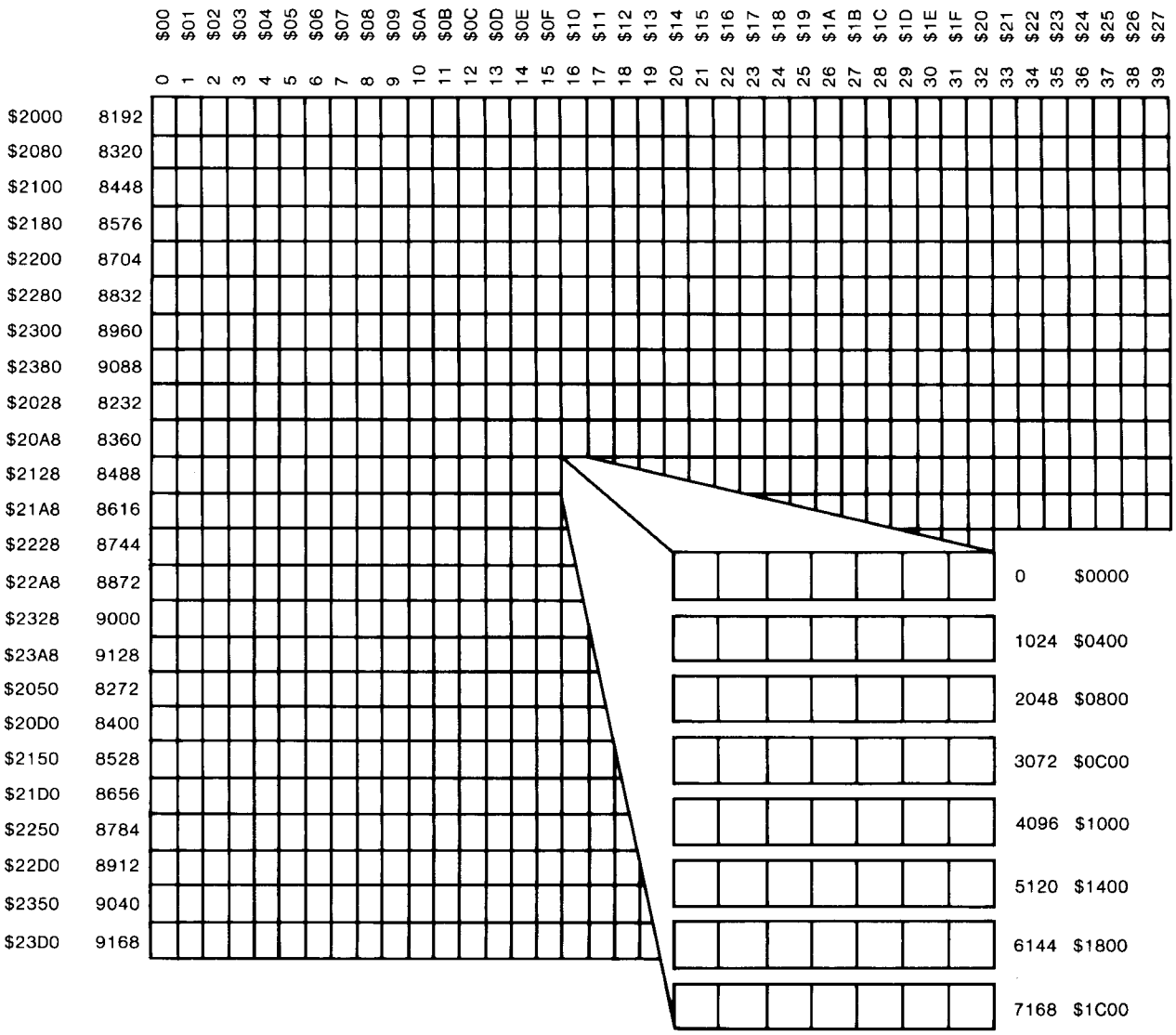


Figure 2-8 Map of High-resolution Graphics Display



Secondary Inputs and Outputs

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

- The speaker (output)
- Cassette input and output
- Annunciator outputs
- Strobe output
- Switch inputs
- Analog (hand control) inputs

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

Some of these devices toggle — change state — each time they are accessed. If you write using an indexed store operation, the Apple IIe's 6502 microprocessor activates the address bus twice during successive clock cycles, causing a device that toggles each time it is addressed to end up back in its original state. For this reason, you should read, rather than write, to such devices.

The Speaker

The Apple IIe has a small speaker mounted toward the front of the bottom plate. The speaker is connected to a soft switch that toggles; it has two states, off and on, and it changes from one to the other each time it is accessed. Electrical specifications of the speaker circuit appear in Chapter 7.

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

The soft switch for the speaker uses memory location `49200` (hexadecimal `$C030`). From Integer BASIC, use the complementary address `-16336`. You can make various tones and buzzes with the

speaker by using combinations of timing loops in your program. There is also a routine in the built-in firmware to make a beep through the speaker. This routine is called BELL1; it is described in Appendix C.

Cassette Input and Output

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

The phone jack marked with a picture of an arrow pointing towards a cassette is the output jack. It is connected to a toggled soft switch, like the speaker switch described above. The signal at the phone jack switches from zero to 25 millivolts or from 25 millivolts to zero each time you access the soft switch. Detailed electrical specifications for the cassette input and output are given in Chapter 7.

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

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

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

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

The Hand Control Connector Signals

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

Ordinarily, you connect a pair of hand controls to the 9-pin connector. The rotary controls use two analog inputs, and the push-buttons use two one-bit inputs. However, you can also use these inputs and outputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick. Complete electrical specifications of these inputs and outputs are given in Chapter 7; Table 7-18 shows the connector pin numbers.

Annunciator Outputs

The four one-bit outputs are called annunciators. Each annunciator can be used to turn a lamp, a relay, or some similar electronic device on and off. For electrical specifications of the annunciator outputs, refer to Chapter 7.

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

Table 2-11 Annunciator Memory Locations

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

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

Strobe Output

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

Switch Inputs

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

The memory locations for these switches are 49249 through 49251 (hexadecimal $C061 through $C063, orcomplementary -16287 through -16285), as shown in Table 2-12. Switch 0 and switch 1 are permanently connected to the OPEN-APPLE and SOLID-APPLE keys on the keyboard; these are the ones normally connected to the buttons on the hand controls. Some software for the older models of the Apple II uses the third switch, switch

2, as a way of detecting the shift key. This technique requires a hardware modification known as the single-wire shift-key mod.

To perform this modification on your Apple IIe, all you have to do is solder across the broken circle labelled X6 on the main circuit board. Early production Apple IIes, identified by a circuit board part number ending in -A, have the shift-key mod active; you can remove it by breaking the circuit at X6. Remember to turn off the power before changing anything inside the Apple IIe. Also remember that changes such as this are at your own risk and may void the warranty.

Warning

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

Analog Inputs

The four analog inputs are designed for use with 150K ohm variable resistors or potentiometers. The variable resistance is connected between the +5V supply and each input, so that it makes up part of a timing circuit (refer to Chapter 7 for details). The circuit changes state when its time constant has elapsed, and the time constant varies as the resistance varies. Your program can measure this time by counting in a loop until the circuit changes state, or times out.

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

To read the analog inputs from machine language, you can use a program loop that resets the timers and then increments a counter until the bit at the appropriate memory location changes

to zero, or you can use the built-in routine. It is called PREAD, and it is described in Appendix C. BASIC and other high-level languages also include convenient means of reading the analog inputs: refer to your language manuals.

Summary of Secondary I/O Locations

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

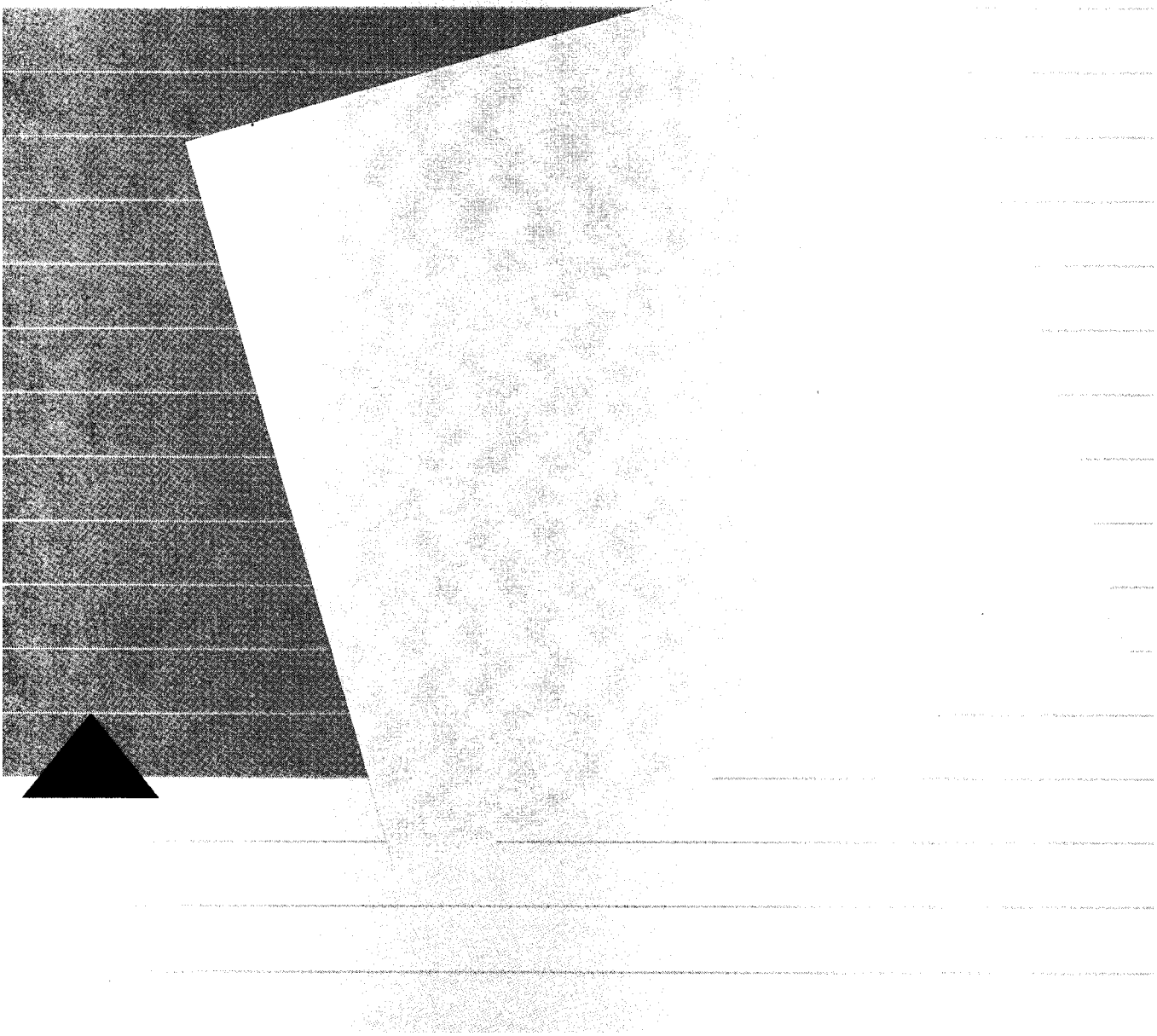
Table 2-12 Secondary I/O Memory Locations

For connector identification and pin numbers, refer to Tables 7-17 and 7-18.

Function	Address Decimal	Hex	Notes
Speaker	49200 -16336	$C030	Read
Cassette Out	49184 -16352	$C020	Read
Cassette In	49248 -16288	$C060	Read
Annunciator 0 On	49241 -16295	$C059	
Annunciator 0 Off	49240 -16296	$C058	
Annunciator 1 On	49243 -16293	$C05B	
Annunciator 1 Off	49242 -16294	$C05A	
Annunciator 2 On	49245 -16291	$C05D	
Annunciator 2 Off	49244 -16292	$C05C	
Annunciator 3 On	49247 -16289	$C05F	
Annunciator 3 Off	49246 -16290	$C05E	
Strobe Output	49216 -16320	$C040	Read
Switch Input 0 (OPEN-APPLE key)	49249 -16287	$C061	Read
Switch Input 1 (SOLID-APPLE key)	49250 -16286	$C062	Read
Switch Input 2	49251 -16285	$C063	Read
Analog Input Reset	49264 -16272	$C070	
Analog Input 0	49252 -16284	$C064	Read
Analog Input 1	49253 -16283	$C065	Read
Analog Input 2	49254 -16282	$C066	Read
Analog Input 3	49255 -16281	$C067	Read

Chapter 3

Built-in I/O Firmware



Chapter 3

Built-in I/O Firmware

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

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

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

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

Table 3-1 Standard I/O Subroutines

Subroutine Name	Location	Description
COUT	$FDED	Character Out: sends a character to the display.
RDKEY	$FD0C	Read Key: displays the blinking cursor; goes to the standard input routine, normally KEYIN.
KEYIN	$FD1B	Key In: with 80-column firmware active, displays checkerboard cursor. Accepts a character from the keyboard.
GETLN	$FD6A	Get Line: displays the prompt character; accepts a sequence of characters by means of RDKEY.

The standard I/O subroutines listed in Table 3-1 are fully described in this chapter. The Apple IIe firmware also contains many other subroutines that you might find useful. Those subroutines are described in Appendix C. Two of the built-in subroutines, AUXMOVE and XFER, can help you use the optional auxiliary memory; those subroutines are described in Chapter 4.

Using the I/O Subroutines

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

Apple II Compatibility

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

Table 3-2 Apple II Mode

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

If the Apple IIe does not have an 80-column text card installed in the auxiliary slot, it is almost in Apple II mode as soon as you turn it on or reset it. One exception is the keyboard, which is both uppercase and lowercase. To be compatible with older software, you have to switch the Apple IIe keyboard to uppercase by pressing the CAPS LOCK key.

Statements in Applesoft and Integer BASIC must be typed in uppercase letters. CAPS LOCK will take care of this, but it makes it inconvenient to use lowercase letters in PRINT statements. If the 80-column firmware is active (see below), you can use uppercase-restrict mode, which forces typed letters to uppercase except inside quotation marks (see Table 3-6).

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

The Apple IIe has another character set, called the *alternate character set*, that displays a full set of normal and inverse uppercase and lowercase characters, but can't display flashing characters. The primary and alternate character sets are described in Chapter 2. You can switch character sets at any time by means of the ALTCHARSET soft switch, also described in Chapter 2.

The 80-column Firmware

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

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

The built-in 80-column firmware is implemented as if it were installed in expansion slot 3. Programs written for older Apple IIs with 80-column display cards installed in slot 3 will run properly on an Apple IIe with an 80-column text card.

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

```
PR#3
```

To activate the 80-column firmware from the Monitor, type 3 and press CONTROL-P. Notice that this is the same procedure you use to activate a card in expansion slot 3. Any auxiliary card installed in the auxiliary slot takes precedence over a card installed in expansion slot 3: see the section "Switching I/O Memory" in Chapter 6 for details.

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

If there is no 80-column text card in your Apple IIe, you can still activate the 80-column firmware and use it with a 40-column display. First, set the INTC3ROM soft-switch located at $C00A (49162); this switch is described in Chapter 6 in the section "Switching I/O Memory". Then type PR#3 to transfer control to the firmware.

When the 80-column firmware is active without a card in the auxiliary slot, it does not work quite the same as it does with a card. The functions that clear the display (CLREOL, CLEOLZ, CLREOP, and HOME) work as if the firmware were inactive: they always clear to black, even in inverse format. Also, interrupts are locked out throughout long operations such as clearing the display. With a card installed, the firmware enables interrupts periodically during these long operations.

Warning

If you do not have either an 80-column text card in the auxiliary slot or a terminal card of some kind in expansion slot 3, don't try to activate the firmware by simply typing PR#3. Typing PR#3 with no card installed transfers control to the empty connector, with unpredictable results.

Programs activate the 80-column firmware by transferring control to address $C300. If there is no card in the auxiliary slot, you must set the INTC3ROM soft switch first. To deactivate the 80-column firmware from a program, write a CONTROL-U character via subroutine COUT.

The Old Monitor

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

```
PR#3
```

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

The Standard I/O Links

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

By calling the I/O subroutines that jump to the link addresses instead of calling the standard subroutines directly, you ensure that your program will work properly in conjunction with other software, such as DOS or a printer driver, that changes one or both of the I/O links. For the purposes of this chapter, we shall assume that the I/O links contain the addresses of the standard I/O subroutines `COUT1` and `KEYIN`. For more information about the I/O links, see the section "Changing the Standard I/O Links" in Chapter 6.

Standard Output Features

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

COUT Output Subroutine

Your program makes a subroutine call to $FDED with a character in the accumulator. COUT then passes control via the output link CSW to the current output, normally COUT1, which takes the character in the accumulator and writes it out. If the accumulator

Table 3.3a Control Characters with COUT1.

(1) Only available when 80-column firmware is active.
(2) Only works from the keyboard.
(3) Doesn't work from the keyboard.

Control Character	ASCII Name	Apple IIe Name	Action Taken by COUT1	Notes
CONTROL - G	(BEL)	bell	Produces a 1000 Hz tone for 0.1 second.	
CONTROL - H	(BS)	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.	
CONTROL - J	(LF)	line feed	Moves cursor position down to next line in window; scrolls if needed.	
CONTROL - K	(VT)	clear EOS	Clears from cursor position to the end of the window.	1
CONTROL - L	(FF)	clear	Moves cursor position to upper-left corner of window and clears window.	1
CONTROL - M	(CR)	return	Moves cursor position to left end of next line in window; scrolls if needed.	
CONTROL - N	(SO)	normal	Sets display format normal.	1, 3
CONTROL - O	(SI)	inverse	Sets display format inverse.	1, 3
CONTROL - Q	(DC1)	40-column	Sets display to 40-column.	1
CONTROL - R	(DC2)	80-column	Sets display to 80-column.1	
CONTROL - S	(DS3)	stop-list	Stops sending characters to the display, until a key is pressed.	1, 2

contains an uppercase or lowercase letter, a number, or a special character, the accumulator contains a control character, COUT1 either performs one of the special functions described below or ignores the character.

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

Table 3-3b Control Characters with COUT1, continued

(1) Only available when 80-column firmware is active.
(2) gotoXY is not supported under BASIC: see the *Apple Pascal Operating System Reference Manual.*

Control Character	ASCII Name	Apple IIe Name	Action Taken by COUT1	Notes
CONTROL - U	(NAK)	quit	Deactivates 80-column firmware, homes cursor, and clears screen.	1
CONTROL - V	(SYN)	scroll	Scrolls the display down one line, leaving the cursor in the current position.	1
CONTROL - W	(ETB)	scroll-up	Scrolls the display up one line, leaving the cursor in the current position.	1
CONTROL - Y	(EM)	home	Moves cursor position to upper-left corner of window (but doesn't clear).	1
CONTROL - Z	(SUB)	clear line	Clears the line the cursor position is on.	1
CONTROL - \	(FS)	fwd. space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below.	1
CONTROL -]	(GS)	clear EOL	Clears line from cursor position to the right edge of the window.	1
CONTROL - ^	(RS)	gotoXY	Using the next two characters, minus 32, as one-byte X and Y values, moves the cursor position to CH=X, CV=Y.	1, 2

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

Control Characters with COUT1

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

The Stop-list Feature

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

The Text Window

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

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

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

Memory location 33 (hexadecimal $21) holds the width of the text window. For a 40-column display, it is normally 40 (hexadecimal $28); for an 80-column display, it is normally 80 (hexadecimal $50). COUT1 truncates the width to an even value.

Warning

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

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

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

Warning

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

Table 3-4 Text Window Memory Locations

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

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

Inverse and Flashing Text

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

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

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

Table 3-5 Text Format Control Values

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

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

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

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

Standard Input Features

The Apple IIe's firmware includes two different subroutines for reading from the keyboard. One subroutine is named RDKEY, which stands for read key. It calls the standard character input subroutine KEYIN, which accepts one character at a time from the keyboard. The other subroutine is named GETLN, which stands for get line. By making repeated calls to RDKEY, GETLN accepts a sequence of characters terminated with a carriage return. GETLN also provides on-screen editing features: see the section "Editing with GETLN".

RDKEY *Input Subroutine*

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

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

The method RDKEY uses to display a cursor works as it did on the older model Apple IIs, which don't display lowercase characters. If you are running an Integer BASIC program with the 80-column firmware inactive, the RDKEY-style cursor is still appropriate. With lowercase characters or with the alternate character set, this method of displaying a cursor is no longer satisfactory.

KEYIN *Input Subroutine*

KEYIN is the standard input subroutine. When called, it waits until the user presses a key, then returns with the key code in the accumulator.

The problem of displaying a cursor without using flashing format is handled by KEYIN. If the 80-column firmware is inactive, KEYIN displays a cursor by alternately storing a checkerboard block in the cursor location, then storing the original character, then the checkerboard again. If the firmware is active, KEYIN displays a steady inverse space (rectangle), unless you are in escape mode, when it displays a plus sign (+) in inverse format. (Escape mode is described in the next section.)

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

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

Escape Codes with KEYIN

KEYIN has many special functions that you invoke by typing escape codes on the keyboard. An escape code is obtained by pressing the ESC key, releasing it, and then pressing some other key, as shown in Table 3-6. The notation in the table — ESC # — means press the ESC key, release it, then press the character that follows.

Table 3-6 Escape Codes

(1) Old-style cursor-control key: see text.
(2) Cursor-control key: see text.
(3) This code functions only when the 80-column firmware is active.

Escape Code	Function	Notes
ESC @	Clears the window and homes the cursor	
ESC A	Moves the cursor up one line	1
ESC B	Moves the cursor right one space	1
ESC C	Moves the cursor left one space	1
ESC D	Moves the cursor down one line	1
ESC E	Clears to the end of the line	
ESC F	Clears to the bottom of the window	
ESC I ESC ↑	Moves the cursor up one line and turns on escape mode	2
ESC J ESC ←	Moves the cursor left one space and turns on escape mode	2
ESC K ESC →	Moves the cursor right one space and turns on escape mode	2
ESC M ESC ↓	Moves the cursor down one line and turns on escape mode	2
ESC R	Turns on restricted-case mode	3
ESC T	Turns off restricted-case mode	3
ESC 4	Switches to 40-column mode, homes the cursor, and clears the screen	3
ESC 8	Switches to 80-column mode, homes the cursor, and clears the screen	3
ESC CONTROL -Q	Deactivates the 80-column firmware	3

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

Cursor Motion in Escape Mode

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

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

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

GETLN *Input Subroutine*

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

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

The prompt characters used by the different programs on the Apple IIe are shown in Table 3-7.

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

Table 3-7 Prompt Characters
*Note: Mini-assembler is available only with Integer BASIC active.

Prompt Character	Program Requesting Input
?	User's BASIC program (INPUT statement)
>	Integer BASIC
]	Applesoft BASIC
*	Firmware Monitor
!	Mini-assembler*

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

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

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

In the Apple II and the Apple II Plus, the GETLN routine converts all inputs to uppercase. GETLN in the Apple IIe does not do this, even in Apple II mode. To get uppercase input for BASIC, use the CAPS LOCK key or switch to restricted-case mode using the escape sequence shown in Table 3-6. With restricted-case active, letters are automatically shifted to uppercase except inside quotation marks.

Editing with GETLN

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

Cancel Line

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

Backspace

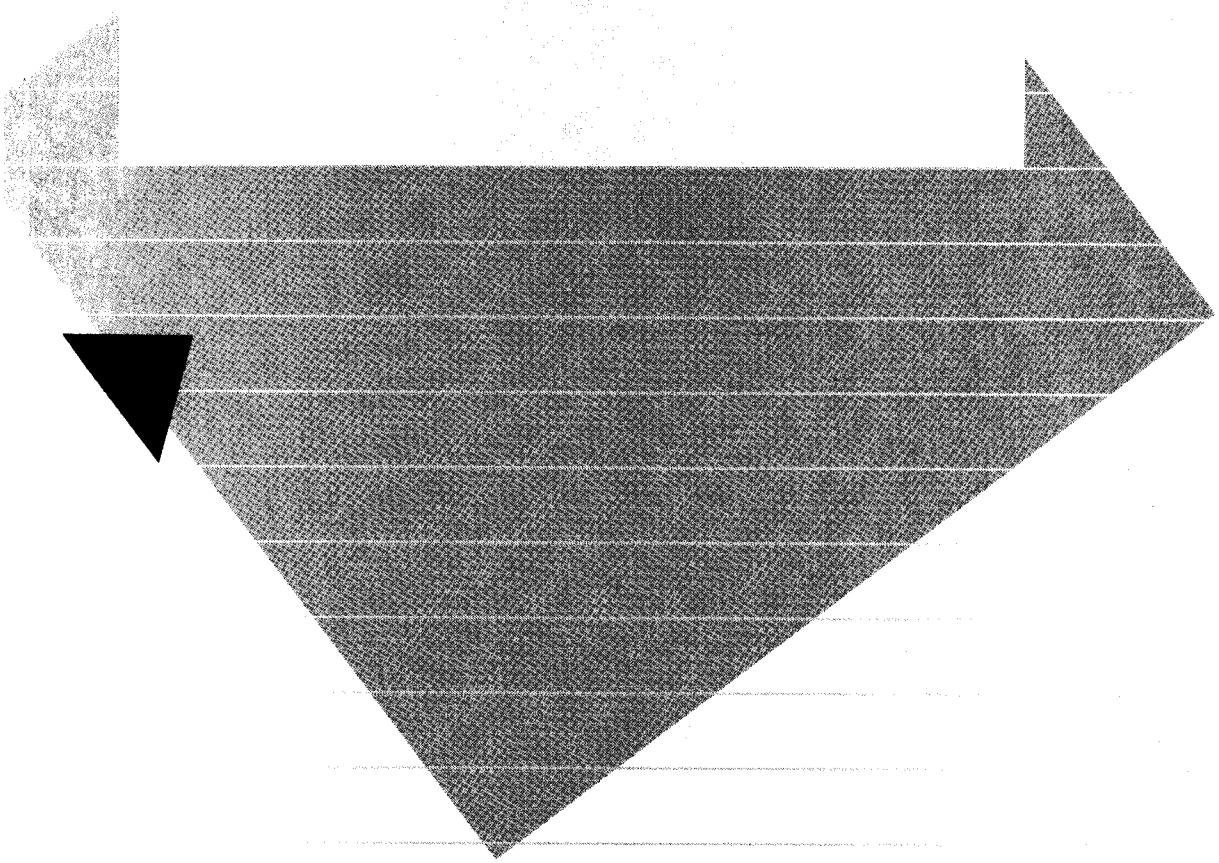
When you press the LEFT-ARROW key, GETLN moves its buffer pointer back one space, effectively deleting the last character in its buffer. It also sends a backspace character to routine COUT, which moves the display position and the cursor back one space. If you type another character now, it will replace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

Retype

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

Chapter 4

Memory Organization



Chapter 4

Memory Organization

The Apple IIe's 6502 microprocessor can address 65,536 (64K) memory locations (K stands for 1024; refer to the glossary). All of the Apple IIe's programmable storage (RAM, for random-access memory), read-only storage (ROM, for read-only memory) and input and output devices are allocated locations in this 64K address space. Some functions share the same addresses — but not at the same time. For information about these shared address spaces, see the section "Bank-switched Memory" in this chapter and the sections "Other Uses of I/O Memory Space" and "Expansion ROM Space" in Chapter 6.

All input and output in the Apple IIe is memory mapped. In this chapter, the I/O memory spaces are described simply as blocks of memory. For details of the built-in I/O features, refer to the descriptions in Chapters 2 and 3. For information about I/O operations with peripheral cards, refer to Chapter 6.

People often refer to the Apple IIe's memory in 256-byte blocks called pages. One reason for this is that a one-byte address counter or index register can specify one of 256 different locations. Thus, page 0 consists of memory locations from 0 to 255 (hexadecimal $0 to $FF), inclusive. Page 1 consists of locations 256 to 511 (hexadecimal $100 to $1FF — note that the page number is the high-order part of the hexadecimal address). Don't confuse this kind of page with the display buffers in the Apple IIe, which are sometimes referred to as Page 1 and Page 2.

Main Memory Map

The map of the main memory address space in Figure 4-1 shows the functions of the major areas of memory. For more details on the I/O space from 48K to 52K ($C000 through $CFFF),

refer to Chapter 2 and Chapter 6; the bank-switched memory in the memory space from 52K to 64K (`$D000` through `$FFFF`) is described below.

Figure 4-1 System Memory Map

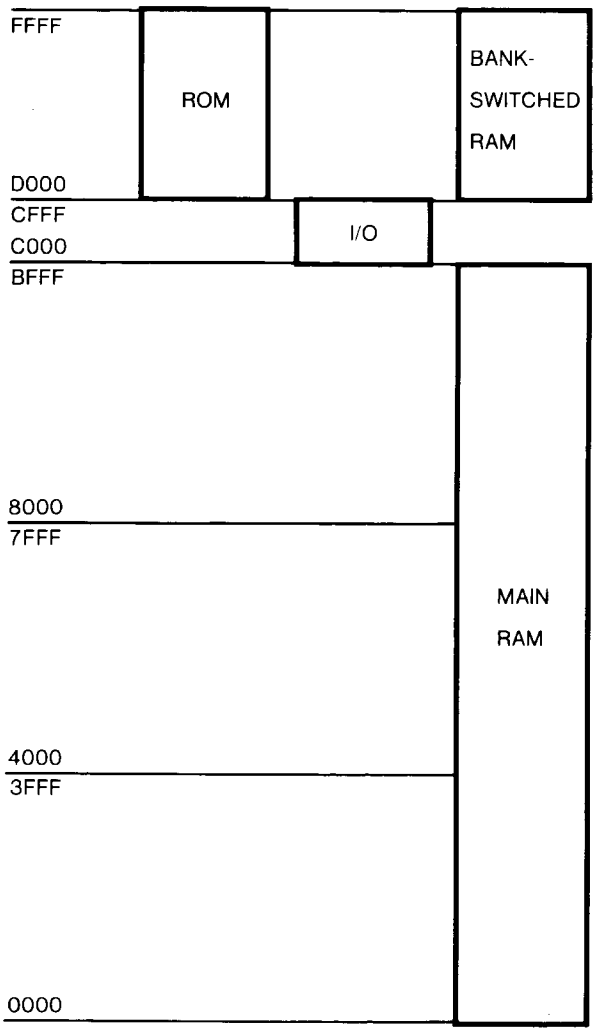
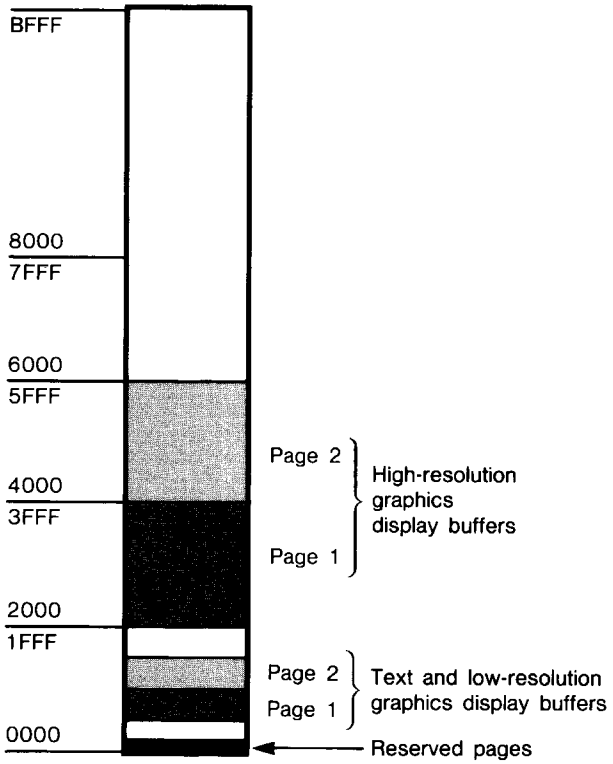


Figure 4-2 RAM Allocation Map



RAM Memory Allocation

As Figure 4-1 shows, the major portion of the Apple IIe's memory space is allocated to programmable storage (RAM). Figure 4-2 shows the areas allocated to RAM. The main RAM memory extends from location 0 to location 49151 (hex $BFFF), and occupies pages 0 through 191 (hexadecimal $BF). There is also RAM storage in the bank-switched space from 53248 to 65535 (hexadecimal $D000 to $FFFF), described in a separate section of this chapter, and auxiliary RAM on the 80-column text card or the extended 80-column text card, described in Chapter 6.

Reserved Memory Pages

Most of the Apple IIe's RAM is available for storing your programs and data. However, a few RAM pages are reserved for the use of the Monitor firmware and the BASIC interpreters. The reserved pages are described below.

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

Page Zero

Several of the 6502 microprocessor's addressing modes require the use of addresses in page zero, also called zero page. The Monitor, the BASIC interpreters, and DOS all make extensive use of page zero.

To use indirect addressing in your assembly-language programs, you must store base addresses in page zero. At the same time, you must avoid interfering with the other programs that use page zero — the Monitor, the BASIC interpreters, and the Disk Operating Systems. One way to avoid conflicts is to use only those page-zero locations not already used by other programs. Tables 4-1, 4-2, 4-3, and 4-4 show the locations in page zero used by the Monitor, Applesoft BASIC, Integer BASIC, and DOS 3.3.

As you can see from the tables, page zero is pretty well used up, except for a few bytes here and there. It's hard to find more than one or two bytes that aren't used by either BASIC or the Monitor or DOS. Rather than trying to squeeze your data into an unused corner, you may prefer a safer alternative: save the contents of part of page zero, use that part, then restore the previous contents before you pass control to another program.

The 6502 Stack

The 6502 microprocessor uses page 1 as the stack — the place where subroutine return addresses are stored, in first-in, last-out sequence. Many programs also use the stack for temporary storage of the registers (via push and pull operations). You can do the same, but you should use it sparingly. The stack pointer is eight bits long, so the stack can hold only 256 bytes of information at a time. When you store the 257th byte in the stack, the stack pointer repeats itself, or wraps around, so that the new byte replaces the first byte stored, which is now lost. This writing over old data is called stack overflow, and when it happens, the program continues to run normally until the lost information is needed, whereupon the program terminates catastrophically.

The Input Buffer

The GETLN input routine, which is used by the Monitor and the BASIC interpreters, uses page 2 as its keyboard-input buffer. The size of this buffer sets the maximum size of input strings. (Note: Applesoft only uses the first 237 bytes, although it permits

you to type all 256.) If you know that you won't be typing any long input strings, you can store temporary data at the upper end of page 2.

Link-Address Storage

The Monitor and DOS 3.3 both use the upper part of page 3 for link addresses or vectors. Table 4-10 shows the part of page 3 the Monitor uses; refer to the *DOS Manual* to see how DOS uses page 3.

BASIC programs sometimes need short machine-language routines. These routines are usually stored in the lower part of page 3.

The Display Buffers

The primary text and low-resolution-graphics display buffer occupies memory pages 4 through 7 (locations 1024 through 2047, hexadecimal $0400 through $07FF). This entire 1024-byte area is called display Page 1, and it is not usable for program and data storage. There are 64 locations in this area that are not displayed on the screen; these locations are reserved for use by the peripheral cards (see Chapter 6).

Display Page 2, the alternate text and low-resolution-graphics display buffer, occupies memory pages 8 through 11 (locations 2048 through 3071, hexadecimal $0800 through $0BFF). Most programs do not use Page 2 for displays, so they can use this area for program or data storage.

The primary high-resolution-graphics display buffer, called high-resolution Page 1, occupies memory pages 32 through 63 (locations 8192 through 16383, hexadecimal $2000 through $3FFF). If your program doesn't use high-resolution graphics, this area is usable for programs or data.

High-resolution-graphics Page 2 occupies memory pages 64 through 95 (locations 16384 through 24575, hexadecimal $4000 through $5FFF). Most programs use this area for program or data storage.

For more information about the display buffers, see Chapter 2.

Table 4-1 Monitor Zero-page Usage

High Nybble of Address	Low Nybble of Address $0	$1	$2	$3	$4	$5	$6	$7	$8	$9	$A	$B	$C	$D	$E	$F
$00																
$10																•
$20	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$30	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$40	•	•	•	•	•	•	•	•	•	•					•	•
$50	•	•	•	•	•	•										
$60																
$70																
$80																
$90																
$A0																
$B0																
$C0																
$D0																
$E0																
$F0																

Table 4-2 Applesoft Zero-page Usage

High Nybble of Address	Low Nybble of Address $0	$1	$2	$3	$4	$5	$6	$7	$8	$9	$A	$B	$C	$D	$E	$F
$00	•	•	•	•	•	•					•	•	•	•	•	•
$10	•	•	•	•	•	•	•	•	•	•				•		
$20																
$30																
$40																
$50	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$60	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$70	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$80	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$90	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$A0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$B0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$C0	•	•	•	•	•	•	•	•	•	•	•	•	•	•		
$D0	•	•	•	•	•	•			•	•	•	•	•	•	•	•
$E0	•	•	•	•	•	•	•	•	•	•	•					
$F0	•	•	•	•	•	•	•	•	•							

Table 4-3 Integer BASIC Zero-page Usage

High Nybble of Address	Low Nybble of Address $0	$1	$2	$3	$4	$5	$6	$7	$8	$9	$A	$B	$C	$D	$E	$F
$00														•		
$10																
$20																
$30																
$40											•	•	•	•		
$50						•	•	•	•	•	•	•	•	•	•	•
$60	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$70	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$80	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$90	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$A0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$B0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$C0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$D0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$E0																
$F0															•	•

Table 4-4 DOS 3.3 Zero-page Usage

High Nybble of Address	Low Nybble of Address $0	$1	$2	$3	$4	$5	$6	$7	$8	$9	$A	$B	$C	$D	$E	$F
$00																
$10																
$20							•	•			•	•	•	•	•	•
$30						•	•	•	•	•					•	•
$40	•	•	•	•	•	•	•	•	•		•	•	•	•		
$50																
$60								•	•	•	•					•
$70	•															
$80																
$90																
$A0																•
$B0	•															
$C0											•	•	•	•		
$D0									•							
$E0																
$F0																

Bank-switched Memory

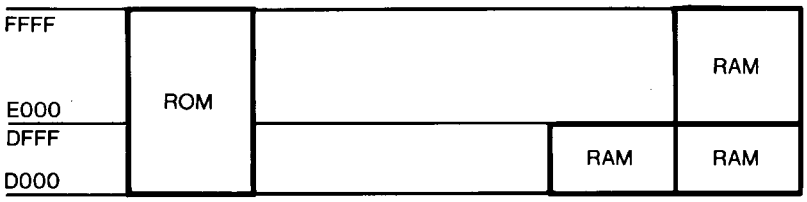
The memory address space from 52K to 64K (hexadecimal $D000 through $FFFF) is doubly allocated: it is used for both ROM and RAM. The 12K bytes of ROM (read-only memory) in this address space contain the Monitor and the Applesoft BASIC interpreter. Alternatively, there are 16K bytes of RAM in this space. The RAM is normally used for storing either the Integer BASIC interpreter or part of the Pascal Operating System (purchased separately).

You may be wondering why this part of memory has such a split personality. Some of the reasons are historical: the Apple IIe is able to run software written for the Apple II and Apple II Plus because it uses this part of memory in the same way they do. It is convenient to have the Applesoft interpreter in ROM, but the Apple IIe, like an Apple II with a language card, is also able to use that address space for other things when Applesoft is not needed.

You may also be wondering how 16K bytes of RAM is mapped into only 12K bytes of address space. The usual answer is that it's done with mirrors, and that isn't a bad analogy: the 4K-byte address space from 52K to 56K (hexadecimal $D000 through $DFFF) is used twice.

Switching different blocks of memory into the same address space is called bank switching. There are actually two examples of bank-switching going on here: first, the entire address space from 52K to 64K ($D000 through $FFFF) is switched between ROM and RAM, and second, the address space from 52K to 56K ($D000 to $DFFF) is switched between two different blocks of RAM.

Figure 4-3 Bank-switched Memory Map



Setting Bank Switches

You switch banks of memory in the same way you switch other functions in the Apple IIe: by using soft switches. These soft switches do three things: select either RAM or ROM in this memory space; enable or inhibit writing to the RAM (write-protect); and select the first or second 4K-byte bank of RAM in the address space $D000 to $DFFF.

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

Table 4-5 shows the addresses of the soft switches for enabling all combinations of reading and writing in this memory space. All of the hexadecimal values of the addresses are of the form $C08x. Notice that several addresses perform the same function: this is because the soft switches are activated by combinations of address bits. For example, any address of the form $C08x with a 1 in the low-order bit enables the RAM for writing. Similarly, bit 3 of the address selects which 4K block of RAM to use for the address space $D000-$DFFF; if bit 3 is 0, the first bank of RAM is used, and if bit 3 is 1, the second bank is used.

Table 4-5 Bank Select Switches

(1) This switch write-enables RAM and read-enables ROM.
(2) Two successive reads to this switch enables RAM both for reading and writing.

Switch Address	Write RAM	Read RAM	Read ROM	4K RAM Bank: First	4K RAM Bank: Second	Notes
$C080		•			•	
$C081	•		•		•	1
$C082			•		•	
$C083	•	•			•	2
$C084		•			•	
$C085	•		•		•	1
$C086			•		•	
$C087	•	•			•	2
$C088		•		•		
$C089	•		•	•		1
$C08A			•	•		
$C08B	•	•		•		2
$C08C		•		•		
$C08D	•		•	•		1
$C08E			•	•		
$C08F	•	•		•		2

When RAM is not enabled for reading, the ROM in this address space is enabled. Even when RAM is not enabled for reading, it can still be written to if it is write-enabled.

When you turn power on or reset the Apple IIe, it initializes the bank switches for reading the ROM and writing the RAM, using the second bank of RAM. Note that this is different from the reset on the Apple II Plus, which didn't affect the bank-switched memory (the language card). On the Apple IIe, you can't use the reset vector to return control to a program in bank-switched memory, as you could on the Apple II Plus.

When you are using Integer BASIC on the Apple IIe, reset works correctly, restarting BASIC with your program intact. This happens because the reset vector transfers control to DOS, and DOS resets the switches for the current version of BASIC.

Note that you can't read one RAM bank and write to the other; if you select either RAM bank for reading, you get that one for writing as well.

You can't read from ROM in part of the bank-switched memory and read from RAM in the rest: specifically, you can't read the Monitor in ROM while reading bank-switched RAM. If you want to use the Monitor firmware with a program in bank-switched RAM, first copy the Monitor from ROM (locations `$F800` through `$FFCB`) into lower RAM and then into bank-switched RAM.

To see how to use these switches, look at the following section of an assembly-language program:

```
AD 83 C0    LDA $C083     ; SELECT 2ND 4K BANK
                            READ/WRITE
AD 83 C0    LDA $C083     ; BY TWO CONSECUTIVE READS
A9 D0       LDA #$D0      ; SET UP...
85 01       STA BEGIN     ; ...NEW...
A9 FF       LDA #$FF      ; ...MAIN-MEMORY...
85 02       STA END       ; ...POINTERS...
20 97 C9    JSR RAMTST    ; ...FOR 12K BANK

AD 8B C0    LDA $C08B     ; SELECT 1ST 4K BANK
20 97 C9    JSR RAMTST    ; USE ABOVE POINTERS

AD 83 C0    LDA $C088     ; SELECT 1ST BANK &
                            WRITE PROTECT
```

```
A9 80       LDA #$80
E6 10       INC TSTNUM
20 58 C9    JSR WPTSINIT

AD 80 C0    LDA $C080     ; SELECT 2ND BANK &
                            WRITE PROTECT
E6 10       INC TSTNUM
A9 01       LDA #PAT12K
20 58 C9    JSR WPTSINIT

AD 8B C0    LDA $C08B     ; SELECT 1ST BANK &
                            READ/WRITE
AD 8B C0    LDA $C08B     ; BY TWO CONSECUTIVE READS
E6 0E       INC RWMODE    ; FLAG RAM IN READ/WRITE
E6 10       INC TSTNUM
A9 08       LDA #PAT4K
20 58 C9    JSR WPTSINIT
```

The LDA instruction, which performs a read operation to the specified memory location, is used for setting the soft switches. The unusual sequence of two consecutive LDA instructions performs the two consecutive reads that write-enable this area of RAM; in this case, the data that are read are not used.

Auxiliary Memory and Firmware

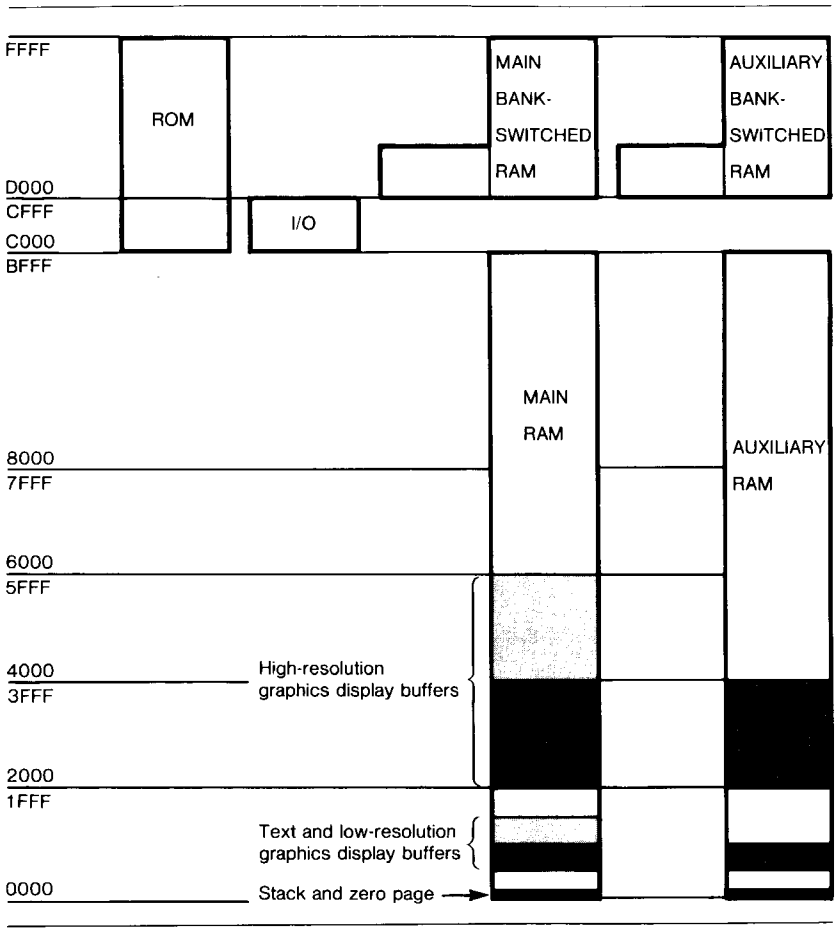
By installing an optional card in the auxiliary slot, you can add more memory to the Apple IIe. One such card is the 80-column text card, which has 1K bytes of additional RAM for expanding the text display from 40 columns to 80 columns.

Another optional card, the extended 80-column text card, has 64K of additional RAM. A 1K-byte area of this memory serves the same purpose as the memory on the 80-column text card: expanding the text display to 80 columns. The other 63K bytes can be used as auxiliary program and data storage. If you use only 40-column displays, the entire 64K bytes is available for programs and data.

Warning

Do not attempt to use the auxiliary memory from a BASIC program. The BASIC interpreter uses several areas in main RAM, including the stack and the zero page. If you switch to auxiliary memory in these areas, the BASIC interpreter fails and you must reset the system and start over.

Figure 4-4 Memory map with Auxiliary Memory



As you can see by studying the memory map in Figure 4-4, the auxiliary memory is broken into two large sections and one small one. The largest section is switched into the memory address space from 512 to 49151 ($200 through $BFFF). This space includes the display buffer pages: as described in Chapter 2, space in auxiliary memory is used for one half of the 80-column text display. You can switch to the auxiliary memory for this entire memory space, or you can switch just the display pages: see the section "Memory Mode Switching", below.

If the only reason you are using auxiliary memory is for the 80-column display, note that you can store into the display page in auxiliary memory by using the 80STORE and PAGE2 soft switches described in the section "Display Mode Switching" in Chapter 2.

The other large section of auxiliary memory is switched into the memory address space from 52K to 64K ($D000 through $FFFF). This memory space and the switches that control it are described above in the section "Bank-switched Memory". If you use the auxiliary RAM in this space, the soft switches have the same effect on the auxiliary RAM that they do on the main RAM: the bank switching is independent of the auxiliary-RAM switching.

Note that the soft switches for the bank-switched memory, described in the previous section, do not change when you switch to auxiliary RAM. In particular, if ROM is enabled in the bank-switched memory space before you switch to auxiliary memory, the ROM will still be enabled after you switch. Any time you switch the bank-switched section of auxiliary memory in and out, you must also make sure that the bank switches are set properly.

When you switch in the auxiliary RAM in the bank-switched space, you also switch the first two pages, from 0 to 511 ($0000 through $01FF). This part of memory contains page zero, which is used for important data and base addresses, and page one, which is the 6502 stack. The stack and zero page are switched this way so that system software running in the bank-switched memory space can maintain its own stack and zero page while it manipulates the 48K address space (from $0200 to $BFFF) in either main memory or auxiliary memory.

Memory Mode Switching

Switching the 48K section of memory is performed by two soft switches: the switch named RAMRD selects main or auxiliary memory for reading, and the one named RAMWRT selects main or auxiliary memory for writing. As shown in Table 4-6, each switch has a pair of memory locations dedicated to it, one to select main memory, and the other to select auxiliary memory. Enabling the read and write functions independently makes it possible for a program whose instructions are being fetched from one memory space to store data into the other memory space.

Warning

Do not use these switches without careful planning. Careless switching between main and auxiliary memories is almost certain to have catastrophic effects on the operation of the Apple IIe. For example, if you switch to auxiliary memory with no auxiliary memory card installed, the program that is running will stop and you will have to reset the Apple IIe and start over.

Writing to the soft-switch at location $C003 turns RAMRD on and enables auxiliary memory for reading; writing to location $C002 turns RAMRD off and enables main memory for reading. Writing to the soft-switch at location $C005 turns RAMWRT on and enables the auxiliary memory for writing; writing to location $C004 turns RAMWRT off and enables main memory for writing. By setting these switches independently, you can use any of the four combinations of reading and writing in main or auxiliary memory.

Auxiliary memory corresponding to text Page 1 and high-resolution graphics Page 1 can be used as part of the address space from $0200 to $BFFF by using RAMRD and RAMWRT as described above. These areas in auxiliary RAM can also be controlled separately by using the switches described in the section "Display Mode Switching" in Chapter 2. Those switches are named 80STORE, PAGE2, and HIRES.

As shown in Table 4-6, the 80STORE switch functions as an enabling switch: with it on, the PAGE2 switch selects main memory or auxiliary memory. With the HIRES switch off, the memory space switched by PAGE2 is the text display Page 1, from $0400 to $07FF; with HIRES on, PAGE2 switches both text Page 1 and high-resolution graphics Page 1, from $2000 to $3FFF.

If you are using both the auxiliary-RAM control switches and the auxiliary-display-page control switches, the display-page control switches take priority: if 80STORE is off, RAMRD and RAMWRT work for the entire memory space from $0200 to $BFFF, but if 80STORE is on, RAMRD and RAMWRT have no effect on the display page. Specifically, if 80STORE is on and HIRES is off, PAGE2 controls text Page 1 regardless of the settings of RAMRD and RAMWRT. Likewise, if 80STORE and HIRES are both on, PAGE2 controls both text Page 1 and high-resolution graphics Page 1, again regardless of RAMRD and RAMWRT.

A single soft switch named ALTZP (for alternate zero page) switches the bank-switched memory and the associated stack and zero page area between main and auxiliary memory. As shown in Table 4-6, writing to location $C009 turns ALTZP on and selects auxiliary-memory stack and zero page; writing to the soft switch at location $C008 turns ALTZP off and selects main-memory stack and zero page for both reading and writing. The section "Auxiliary-memory Routines", below, describes firmware that you can call to help you switch between main and auxiliary memory.

There are three more locations associated with the auxiliary-memory switches. The high-order bits of the bytes you read at these locations tell you the settings of the three soft switches described above. The byte you read at location $C013 has its high bit set to 1 if RAMRD is on (auxiliary memory is read-enabled), or 0 if RAMRD is off (the 48K block of main memory is read-enabled). The byte location $C014 has its high bit set to 1 if RAMWRT is on (auxiliary memory is write-enabled), or 0 if RAMWRT is off (the 48K block of main memory is write-enabled). The byte at location $C016 has its high bit set to 1 if ALTZP is on (the bank-switched area, stack, and zero page in the auxiliary memory are selected), or 0 if ALTZP is off (these areas in main memory are selected).

When these switches are on, auxiliary memory is being used; when they are off, main memory is being used.

Table 4-6 Auxiliary-memory Select Switches

(1) When 80STORE is on, the PAGE2 switch selects main or auxiliary display memory.
(2) When 80STORE is on, the HIRES switch enables you to use the PAGE2 switch to switch between the high-resolution Page-1 area in main memory or auxiliary memory.

Name	Function	Location Hex	Location Decimal	Notes
RAMRD	Read auxiliary memory	$C003	49155 -16381	Write
	Read main memory	$C002	49154 -16382	Write
	Read RAMRD switch	$C013	49171 -16365	Read
RAMWRT	Write auxiliary memory	$C005	49157 -16379	Write
	Write main memory	$C004	49156 -16380	Write
	Read RAMWRT switch	$C014	49172 -16354	Read
80STORE	On: access display page	$C001	49153 -16383	Write
	Off: use RAMRD, RAMWRT	$C000	49152 -16384	Write
	Read 80STORE switch	$C018	49176 -16360	Read
PAGE2	Page 2 on (Aux. memory)	$C055	49237 -16299	1
	Page 2 off (Main memory)	$C054	49236 -16300	1
	Read PAGE2 switch	$C01C	49180 -16356	Read
HIRES	On: access hi-res pages	$C057	49239 -16297	2
	Off: use RAMRD, RAMWRT	$C056	49238 -16298	2
	Read HIRES switch	$C01D	49181 -16355	Read
ALTZP	Auxiliary stack & z. p.	$C009	49161 -16373	Write
	Main stack & zero page	$C008	49160 -16374	Write
	Read ALTZP switch	$C016	49174 -16352	Read

In order to have enough memory locations for all of the soft switches and remain compatible with the Apple II and Apple II Plus, the soft switches listed in Table 4-6 share their memory locations with the keyboard functions listed in Table 2-2. The operations — read or write — shown in Table 4-6 for controlling the auxiliary memory are just the ones that are not used for reading the keyboard and clearing the strobe.

Auxiliary-memory Subroutines

If you want to write assembly-language programs that use auxiliary memory but you don't want to manage the auxiliary memory yourself, you can use the built-in auxiliary-memory subroutines. These subroutines make it possible to use the auxiliary memory without having to manipulate the soft switches described in the previous section.

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

You use these built-in subroutines the same way you use the I/O subroutines described in Chapter 3: by making subroutine calls to their starting locations. Those locations are shown in Table 4-7.

Table 4-7 Auxiliary-memory Routines

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

Moving Data to Auxiliary Memory

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

Warning

Don't try to use AUXMOVE to copy data in page zero or page one (the 6502 stack) or in the bank-switched memory ($D000-$FFFF). AUXMOVE uses page zero all during the copy, so it can't handle moves in the memory space switched by ALTZP.

The pairs of bytes you use for passing addresses to this subroutine are called A1, A2, and A4, and they are used for parameter passing by several of the Apple IIe's built-in routines. The addresses of these byte pairs are shown in Table 4-8.

Table 4-8 Parameters for AUXMOVE Routine

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

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

The AUXMOVE routine uses the carry bit to select the direction to copy the data. To copy data from main memory to auxiliary memory, set the carry bit; to copy data from auxiliary memory to main memory, clear the carry bit.

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

Transferring Control to Auxiliary Memory

You can use the built-in routine named XFER to transfer control to and from program segments in auxiliary memory. You must set up three parameters before using XFER: the address of the routine you are transferring to, the direction of the transfer (main to auxiliary or auxiliary to main), and which page zero and stack you want to use.

Table 4-9 Parameters for XFER Routine

Name or Location	Parameter passed
Carry	1 = Transfer from main to auxiliary memory 0 = Transfer from auxiliary to main memory
Overflow	1 = Use page zero and stack in auxiliary memory 0 = Use page zero and stack in main memory
$3ED	Program starting address, low-order byte
$3EE	Program starting address, high-order byte

Put the transfer address into the two bytes at locations $3ED and $3EE, with the low-order byte first, as usual. The direction of the transfer is controlled by the carry bit: set the carry bit to transfer to a program in auxiliary memory; clear the carry bit to transfer to a program in main memory. Use the overflow bit to select which page zero and stack you want to use: clear the overflow bit to use the main memory; set the overflow bit to use the auxiliary memory.

After you have set up the parameters, pass control to the XFER routine by a jump instruction, rather than a subroutine call. XFER saves the accumulator and the transfer address on the current stack, then sets up the soft switches for the parameters you have selected and jumps to the new program.

Warning

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

The Reset Routine

To put the Apple IIe into a known state when it has just been turned on or after a program has malfunctioned, there is a procedure called the reset routine. The reset routine is built into the Apple IIe's firmware, and it is initiated any time you turn power on or press the RESET key while holding down the CONTROL key. The reset routine puts the Apple IIe into its normal operating mode and restarts the resident program.

When you initiate a reset, hardware in the Apple IIe sets the memory-controlling soft switches to normal: main board RAM and ROM are enabled, and, if there is an 80-column text or extended 80-column text card in the auxiliary slot, expansion slot 3 is allocated to the built-in 80-column firmware. Auxiliary RAM is disabled and the bank-switched memory space is set up to read from ROM and write to RAM, using the second bank at $D000.

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

The reset routine sets the keyboard and display as the standard input and output devices by loading the standard I/O links (see Chapter 6). It turns annunciators 0 and 1 off and annunciators 2 and 3 on, clears the keyboard strobe, turns off any active accessory-card ROM (see Chapter 6) and outputs a bell (tone).

The Apple IIe has three types of reset: power-on reset, also called cold-start reset; warm-start reset; and forced cold-start reset. The procedure described above is the same for any type of reset. What happens next depends on the reset vector. The reset routine checks the reset vector to determine whether it is

valid or not, as described below in the section, "The Reset Vector". If the reset was caused by turning the power on, the vector will not be valid, and the reset routine will perform the cold-start procedure. If the vector is valid, the routine will perform the warm-start procedure.

The Cold-start Procedure

If the reset vector is not valid, either the Apple IIe has just been turned on or something has caused memory contents to be changed. The reset routine clears the display and puts the string "Apple][" at the top of the display. It loads the reset vector and the validity-check byte as described below, then starts checking the expansion slots to see if there is a disk drive controller card in one of them, starting with slot 7 and working down. If it finds a controller card, it initiates the bootstrap (startup) routine that resides in the controller card's firmware. The bootstrap then loads the Disk Operating System from the disk in drive 1. When DOS has been loaded, it displays other messages on the screen. If there is no disk in the disk drive, the drive motor just keeps spinning until you press CONTROL - RESET . For more information about DOS and the startup procedure, see the *DOS Manual*.

If the reset routine doesn't find a controller card, or if you press CONTROL - RESET again before the startup procedure has been completed, the reset routine will continue without using the disk, and pass control to the built-in Applesoft interpreter.

The Warm-start Procedure

Whenever you press CONTROL - RESET when the Apple IIe has already completed a cold-start reset, the reset vector is still valid and it is not necessary to reinitialize the entire system. The reset routine simply uses the vector to transfer control to the resident program, which is normally the built-in Applesoft interpreter. If the resident program is indeed Applesoft, your Applesoft program and variables are still intact. If you are using DOS, it is the resident program and it restarts either Applesoft or Integer BASIC, whichever you were using when you pressed CONTROL - RESET .

A program in bank-switched RAM cannot use the reset vector to regain control after a reset, because the reset routine enables ROM in the bank-switched memory space. If you are using Integer BASIC, which is in the bank-switched RAM, you are also using DOS, and it is DOS that controls the reset vector and restarts BASIC.

Forced Cold Start

If a program has loaded the reset vector to point to the beginning of the program, as described below, pressing CONTROL - RESET causes a warm-start reset that uses the vector to transfer control to that program. If you want to stop such a program without turning the power off and on, you can force a cold-start reset by holding down the CONTROL key and the OPEN-APPLE key, then pressing and releasing the RESET key.

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

Whenever you press CONTROL - RESET, firmware in the Apple IIe always checks to see whether either APPLE key is down. If the SOLID-APPLE key is down, with or without the OPEN-APPLE key, the firmware performs the self-test described below. If only the OPEN-APPLE key is down, the firmware starts a forced cold-start reset. First, it destroys the program or data in memory by writing two bytes of arbitrary data into each page of main RAM. The two bytes that get written over in page 3 are the ones that contain the reset vector. The reset routine then performs a normal cold-start reset.

The Reset Vector

When you reset the Apple IIe, the reset routine transfers control to the resident program by means of an address stored in page 3 of main RAM. This address is called a *vector* because it directs program control to a specified destination. There are several other vector addresses stored in page 3, as shown in Table 4-10, including the interrupt vectors described in Chapter 6, and the DOS vectors described in the *DOS Manual*.

The cold-start reset routine stores the starting address of the built-in Applesoft interpreter, low-order byte first, in the reset vector address at locations 1010 and 1011 (hexadecimal $3F2 and $3F3). It then stores a validity-check byte, also called the power-up byte, at location 1012 (hexadecimal $3F4). The validity-check byte is computed by performing an exclusive-OR of the second byte of the vector with the constant 165 (hexadecimal $A5). Each time you reset the Apple IIe, the reset routine uses this byte to determine whether the reset vector is still valid.

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

Table 4-10 Page 3 Vectors

Vector address Decimal	Hex	Vector function
1008 1009	$3F0 $3F1	Address of the subroutine that handles BRK requests (normally $59, $FA).
1010 1011	$3F2 $3F3	Reset vector (see text).
1012	$3F4	Power-up byte (see text).
1013 1014 1015	$3F5 $3F6 $3F7	Jump instruction to the subroutine that handles Applesoft "&" commands (normally $4C, $58, $FF).
1016 1017 1018	$3F8 $3F9 $3FA	Jump instruction to the subroutine that handles user (CONTROL-Y) commands.
1019 1020 1021	$3FB $3FC $3FD	Jump instruction to the subroutine that handles non-maskable interrupts.
1022 1023	$3FE $3FF	Interrupt vector (address of the subroutine that handles interrupt requests).

The reset routine has a subroutine that generates the validity-check byte for the current reset vector. You can use this subroutine by doing a subroutine call to location -1169 (hexadecimal $FB6F). When your program finishes, it can return the Apple IIe to normal operation by restoring the original reset vector and again calling the subroutine to fix up the validity-check byte.

Automatic Self-test

If you reset the Apple IIe by holding down the CONTROL key and the SOLID-APPLE key while pressing and releasing the RESET key, the reset routine will start running the built-in self-test. Successfully running this test assures you that the Apple IIe is operational.

Warning

The self-test routine tests the Apple IIe's programmable memory by writing and then reading it. All programs and data in programmable memory when you run the self-test are destroyed.

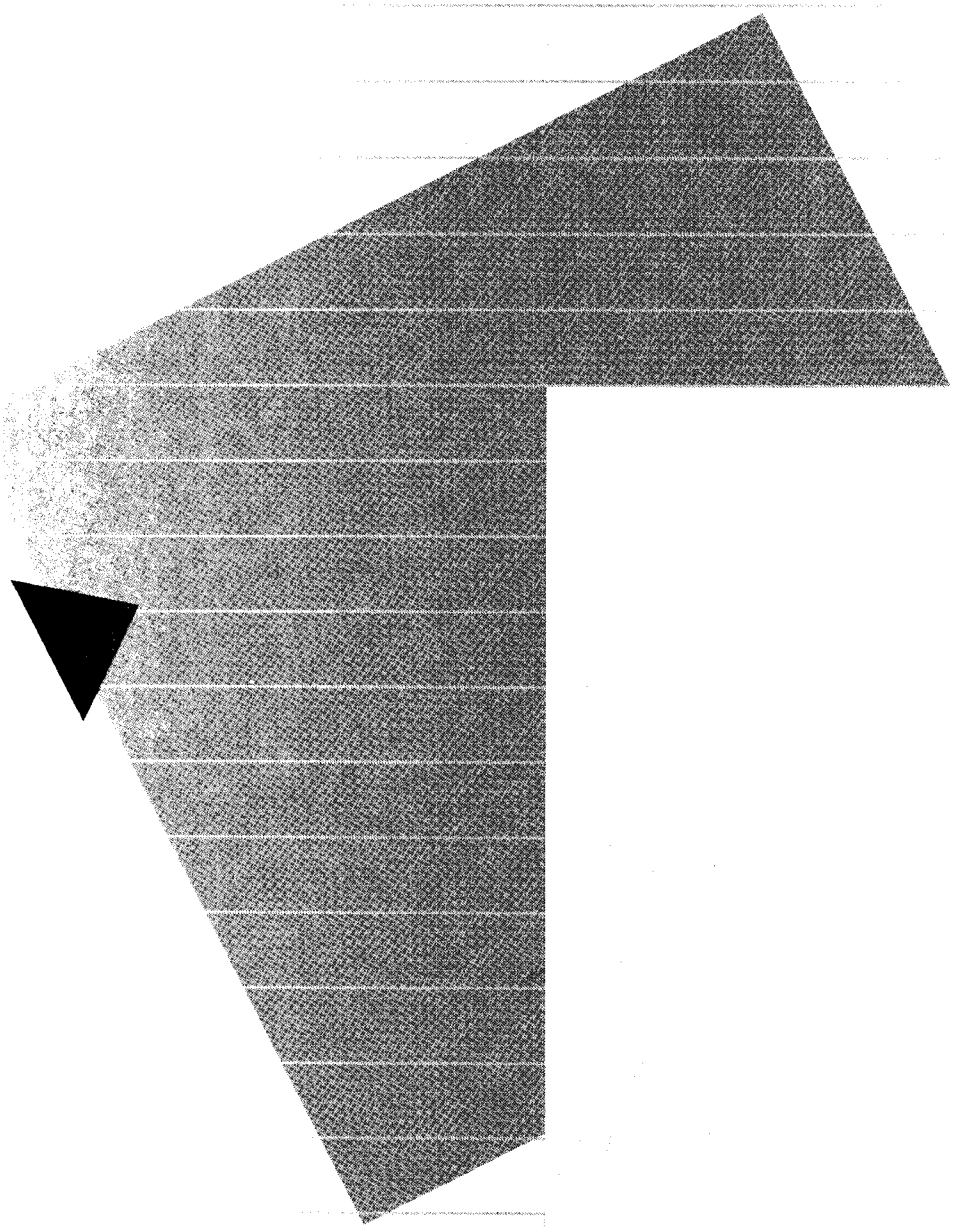
The self-test takes several seconds to run. While it is running, the display changes from black to white and back twice. If the test finishes normally, the Apple IIe displays an "OK" message and waits for you to request a normal reset.

If the self-test detects something wrong, it displays an error message and stops. If you have been running programs prior to running the self-test, some soft-switches could be on, causing the self-test to fail and display an error message. If this happens, turn the power off for several seconds, then turn it back on and run the self-test again. If it still fails, there is really something wrong; to get it corrected, contact your Apple dealer for service.

BLANK PAGE

Chapter 5

Using The Monitor



Chapter 5

Using The Monitor

The System Monitor is a set of subroutines in the Apple IIe firmware. The Monitor provides a standard interface to the built-in I/O devices described in Chapter 2. The I/O subroutines described in Chapter 3 are part of the System Monitor.

The Disk Operating System and the BASIC interpreters use these subroutines by direct calls to their starting locations, as described for the I/O subroutines in Chapter 3; the starting addresses for all of the standard subroutines are listed in Appendix C. If you wish, you can call the standard subroutines from your programs in the same fashion.

You can perform most of the Monitor functions directly from the keyboard. This chapter tells you how to use the Monitor to

- look at one or more memory locations
- change the contents of any location
- write programs in machine language to be executed directly by the Apple IIe's microprocessor
- save blocks of data and programs onto cassette tape and read them back in again
- move and compare blocks of memory
- invoke other programs from the Monitor

Invoking the Monitor

The System Monitor starts at memory location `$FF69` (decimal `65385` or `-151`). To invoke the Monitor, you make a `CALL` statement to this location from the keyboard or from a BASIC program. When the Monitor is running, its prompting character, an asterisk (`*`), appears on the left side of the display screen, followed by a blinking cursor.

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

Syntax of Monitor Commands

To give a command to the Monitor, you type a line on the keyboard, then press RETURN. The Monitor accepts the line using the standard I/O subroutine GETLN described in Chapter 3. A Monitor command can be up to 255 characters in length, ending with a carriage return.

A Monitor command can include three kinds of information: addresses, data values, and command characters. You type addresses and data values in hexadecimal notation. Hexadecimal notation uses the ten decimal digits (0-9) and the first six letters (A-F) to represents the sixteen values from 0 to I5. A pair of hexadecimal digits represents values from 0 to 255, corresponding to a byte, and a group of four hexadecimal digits can represent values from 0 to 65,536, corresponding to a word. Any address in the Apple IIe can be represented by four hexadecimal digits.

When the command you type calls for an address, the Monitor accepts any group of hexadecimal digits. If there are fewer than four digits in the group, it adds leading zeros; if there are more than four hexadecimal digits, the Monitor uses only the last four digits. It follows a similar procedure when the command syntax calls for two-digit data values.

Each command you type consists of one command character, usually the first letter of the command name. The Monitor recognizes 22 different command characters. Some of them are punctuation marks, some are uppercase letters, and some are control characters. Note: although the Monitor recognizes and interprets them, control characters typed on an input line do not appear on the screen. (See the "Summary of Monitor Commands" at the end of the chapter.)

This chapter contains many examples of the use of Monitor commands. In the examples, the commands and values you type are shown in a normal typeface and the responses of the Monitor are in a computer typeface. Of course, when you perform the

examples, all of the characters that appear on the display screen will be in the same typeface. Some of the data values displayed by your Apple IIe may differ from the values printed in these examples, because they are variables stored in programmable memory.

Monitor Memory Commands

When you use the Monitor to examine and change the contents of memory, it keeps track of the address of the last location whose value you inquired about and the address of the location that is next to have its value changed. These are called the last opened location and the next changeable location.

Examining Memory Contents

When you type the address of a memory location and press RETURN, the Monitor responds with the address you typed, a dash, a space, and the value stored at that location, like this:

```
*E000
E000- 20
*33
0033-AA
*
```

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

Memory Dump

When you type a period (.) followed by an address, and then press RETURN, the Monitor displays a memory dump: the data values stored at all the memory locations from the one following the last opened location to the location whose address you typed following the period. The Monitor saves the last location displayed as both the last opened location and the next changeable location. In these examples, the amount of data displayed by the Monitor depends on how much larger than the last opened location the address after the period is.

```
*20

0020- 00

*.2B

0021- 28 00 18 0F 0C 00 00
0028- A8 06 D0 07

*300

0300- 99

*.315

0301- B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03

*.32A

0316- 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20
*
```

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

When the Monitor performs a memory dump, it starts at the location immediately following the last opened location and displays that address and the data value stored there. It then displays the values of successive locations up to and including the location whose address you typed, but only up to eight values on a line. When it reaches a location whose address is a multiple of eight—that is, one that ends with an 8 or a 0—it displays that address as the beginning of a new line, then continues displaying more values.

After the Monitor has displayed the value at the location whose address you specified in the command, it stops the memory dump and sets that location as both the last opened location and the next changeable location. If the address specified on the input line is less than the address of the last opened location, the Monitor displays only the address and value of the location following the last opened location.

You can combine the two commands, opening a location and dumping memory, by simply concatenating them: type the first address, a period, and the second address. This combination of two addresses separated by a period is called a memory range.

```
*300.32F
0300- 99 B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20 46 03 A5 3D 4D

*30.40
0030- AA 00 FF AA 05 C2 05 C2
0038- 1B FD D0 03 3C 00 40 00
0040- 30

*E015.E025
E015- 4C ED FD
E018- A9 20 C5 24 B0 0C A9 8D
E020- A0 07 20 ED FD A9
*
```

Pressing the RETURN key by itself causes the Monitor to display one line of a memory dump; that is, a memory dump from the location following the last opened location to the next multiple-of-eight boundary. The Monitor saves the address of the last location displayed as the last opened location and the next changeable location.

```
*5
0005- 00
* RETURN
00 00
* RETURN
0008- 00 00 00 00 00 00 00 00
*32
0032- FF
* RETURN
AA 00 C2 05 C2
* RETURN
0038- 1B FD D0 03 3C 00 3F 00
*
```

Changing Memory Contents

The previous section showed you how to display the values stored in the Apple IIe's memory; this section shows you how to change those values. You can change any location in RAM (programmable memory) and you can also change the soft switches and output devices by changing the locations assigned to them.

Warning

Use these commands carefully. If you change the zero-page locations used by Applesoft and DOS, you may lose programs or data stored in memory.

Changing One Byte

The previous commands keep track of the next changeable location; these commands make use of it. In the next example, you open location 0, then type a colon followed by a value.

```
*0
0000- 00
*:5F
```

The contents of the next changeable location have just been changed to the value you typed, as you can see by examining that location:

```
*0
0000- 5F
*
```

You can also combine opening and changing into one operation by typing an address followed by a colon and a value. In the example, you type the address again to verify the change.

```
*302:42
*302
0302- 42
*
```

When you change the contents of a location, the value that was contained in that location disappears, never to be seen again. The new value will remain until you replace it with another value.

Changing Consecutive Locations

You don't have to type a separate command with an address, a colon, a value, and RETURN for each location you want to change. You can change the the values of up to eighty-five

consecutive locations at a time (or even more, if you omit leading zeros from the values) by typing only the initial address and colon followed by all the values separated by spaces, and ending with RETURN. The Monitor will duly store the consecutive values in consecutive locations, starting at the location whose address you typed. After it has processed the string of values, it takes the location following the last changed location as the next changeable location. Thus, you can continue changing consecutive locations without typing an address on the next input line by typing another colon and more values. In these examples, you first change some locations, then examine them to verify the changes.

```
*300:69 01 20 ED FD 4C 0 3

*300

0300- 69

* RETURN

01 20 ED FD 4C 00 03

*10:0 1 2 3

*:4 5 6 7

*10.17

0010- 00 01 02 03 04 05 06 07
*
```

Moving Data in Memory

You can copy a block of data stored in a range of memory locations from one area in memory to another by using the Monitor's MOVE command. To move a range of memory, you must tell the Monitor both where the data is now situated in memory — the source locations — and where you want the copy to go — the destination locations. You give this information to the Monitor by means of three addresses: the address of the first location in the destination and the addresses of the first and last locations in the source. You specify the starting and ending addresses of the source range by separating them with a period. You separate the destination address from the range addresses with a less-than

character (<), which you may think of as an arrow pointing in the direction of the move. Finally, you tell the Monitor that this is a MOVE command by typing the letter M. The format of the complete MOVE command looks like this:

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

When you type the actual command, the words in curly braces should be replaced by hexadecimal addresses, and the braces and spaces should be omitted. Here are some examples of memory moves. First, you examine the values stored in one range of memory, then store several values in another range of memory; the actual MOVE commands end with the letter M:

```
*0.F
0000- 5F 00 05 07 00 00 00 00
0008- 00 00 00 00 00 00 00 00
*300:A9 8D 20 ED FD A9 45 20 DA FD 4C 00 03
*300.30C
0300- A9 8D 20 ED FD A9 45 20
0308- DA FD 4C 00 03
*0<300.30CM
*0.C
0000- A9 8D 20 ED FD A9 45 20
0008- DA FD 4C 00 03
*310 8.AM
*310.312
0310- DA FD 4C
*2<7.9M
*0.C
0000- A9 8D 20 DA FD A9 45 20
0008- DA FD 4C 00 03
*
```

The Monitor moves a copy of the data stored in the source range of locations to the destination locations. The values in the source range are left undisturbed. The Monitor remembers the last location in the source range as the last opened location, and the first location in the source range as the next changeable location. If the second address in the source range specification is less than the first, then only one value (that of the first location in the range) will be moved.

If the destination address of the MOVE command is inside the source range of addresses, then strange (and sometimes wonderful) things happen: the locations between the beginning of the source range and the destination address are treated as a sub-range and the values in this sub-range are replicated throughout the source range. See the section "Special Tricks with the Monitor" for an interesting application of this feature.

Comparing Data in Memory

You can use the VERIFY command to compare two ranges of memory using the same format you use to move a range of memory from one place to another. In fact, the VERIFY command can be used immediately after a MOVE to make sure that the move was successful. The VERIFY command, like the MOVE command, needs a range and a destination. The syntax of the VERIFY command is:

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

The Monitor compares the values in the source locations with the values in the locations beginning at the destination address. If any values don't match, the Monitor displays the address at which the discrepancy was found and the two values that differ. In the example, you store data values in the range of locations from 0 to $D, copy them to locations starting at $300 with the MOVE command, and then compare them using the VERIFY command. When you use the VERIFY command after you change the value at location 6 to $E4, it detects the change.

```
*0:D7 F2 E9 F4 F4 E5 EE A0 E2 F9 A0 C3 C4 C5

*300<0.DM

*300<0.DV

*6:E4

*300<0.DV

0006-E4 (EE)
*
```

If the VERIFY command finds a discrepancy, it displays the address of the location in the source range whose value differs from its counterpart in the destination range. If there is no discrepancy, VERIFY displays nothing. The VERIFY command leaves the values in both ranges unchanged. The last opened location is the last location in the source range, and the next changeable location is the first location in the source range, just as in the MOVE command. If the ending address of the range is less than the starting address, the values of only the first locations in the ranges will be compared. Like the MOVE command, the VERIFY command also does unusual things if the destination address is within the source range; see the section "Special Tricks with the Monitor".

Monitor Register Command

Even though the actual contents of the 6502's internal registers are changing as you use the Monitor, you can examine the values that the registers contained at the time the Monitor gained control, either because you called it or because the program you are debugging stopped at a break (BRK). You can also store new register values that will be used when you execute a program from the Monitor using the GO command, described below.

Examining and Changing Registers

When you call the Monitor, it stores the contents of the 6502 registers in memory. The registers are stored in the order A, X, Y, P (processor status register), and S stack pointer), starting at location $45 (decimal 69). When you give the Monitor a GO command, the Monitor loads the registers from these five locations before it executes the first instruction in your program.

Typing CONTROL-E and pressing RETURN invokes the Monitor's EXAMINE command, which displays the stored register values and sets the location containing the contents of the A-register as the next changeable location. After using the EXAMINE command, you can change the values in these locations by typing a colon and then typing the new values separated by spaces. In the following example, you display the registers, change the first two, and then display them again to verify the change.

```
*CONTROL-E
A=0A X=FF Y=D8 P=B0 S=F8
*:B0 02
*CONTROL-E
A=B0 X=02 Y=D8 P=B0 S=F8
*
```

Monitor Cassette Tape Commands

The Apple IIe has two jacks for connecting an audio cassette tape recorder. With a recorder connected, you can use the Monitor commands described below to save the contents of a range of memory onto a standard cassette and recall it again for later use.

Saving Data on Tape

The Monitor's WRITE command saves the contents of up to 65,536 memory locations on cassette tape. To save a range of memory on tape, give the Monitor the starting and ending addresses of the range, followed by the letter W (for WRITE), like this:

{start} . {end} W

Don't press RETURN yet: first, put the tape recorder in record mode and let the tape run for a second, then press RETURN. The Monitor will write a ten-second tone onto the tape and then write the data. The tone acts as a leader: later, when the Monitor reads the tape, the leader enables the Monitor to get in step with the signal from the tape. When the Monitor is finished writing the range you specified, it will sound a bell (beep) and display a prompt. You should rewind the tape and label it with the memory range that's on the tape and what it's supposed to be.

Here's a small example you can save and use later to try out the READ command. Remember that you must start the cassette recorder in record mode before you press RETURN after typing the WRITE command.

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

*0.14
```


```
*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 an average data transfer rate of about 1,350 bits per second.

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, which is the eight-bit partial sum of all values in the range. When the Monitor reads the tape, it uses this value to determine if the data has been written and read correctly (see below).

Reading Data 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 computer by using the Monitor's READ command. The data values 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 memory range in the computer's memory, provided that it's the same size as the range you saved. The format of the READ command is the same as that of the WRITE command, except that the command letter is R:

```
{start} . {end} R
```

Once again, after typing the command, don't press RETURN. Instead, start the tape recorder in play mode and wait a few seconds. 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 to lock on to the signal from the tape. 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.

This example has two parts. First, you set a range of memory to zero, verify the contents of memory, and then type the READ command, but don't press RETURN.

```
*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
0008- 00 00 00 00 00 00 00 00
0010- 00 00 00 00 00

*0.14R
```

Now start the cassette running in play mode, wait a few seconds, and press RETURN. After the Monitor sounds the bell (beep) and displays the prompt, examine the range of memory to see that the values from the tape were read correctly:

```
*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
*
```

After the Monitor has read all the data values on the tape, it reads the checksum value. It computes the checksum on the data it read and compares it to the checksum from the tape. If the two checksums differ, the Monitor sends a beep to the speaker and displays "ERR". This warns you that there was a problem reading the tape and that the values stored in memory aren't the values that were recorded on the tape. If the two checksums match, the Monitor will just send out a beep and display a prompt.

Miscellaneous Monitor Commands

These Monitor commands enable you to change the video display format from normal to inverse and back, and to assign input and output to peripherals in expansion slots.

Display Inverse and Normal

You can control the setting of the inverse-normal mask location used by the COUT subroutine (described in Chapter 3) from the Monitor so that all of the Monitor's output will be in inverse format. The INVERSE command, I, sets the mask such that all subsequent inputs and outputs are displayed in inverse format. To switch the Monitor's output back to normal format, use the NORMAL command, N.

```
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*I
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*N
*0.F
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
*
```

Back to BASIC

Use the BASIC command, CONTROL-B, to leave the Monitor and enter the BASIC that was active when you entered the Monitor. Normally, this is Applesoft BASIC, unless you deliberately switched to Integer BASIC. Any program or variables that you had previously in BASIC will be lost. If you want to re-enter BASIC with your previous program and variables intact, use the CONTINUE BASIC command, CONTROL-C. If you are using the Apple Disk Operating System (DOS), press CONTROL-RESET or type

```
3D0G
```

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

If you type the latter command, make sure that the third character you type is a zero, not a letter O. The letter G is the Monitor's GO command, described below in the section "Machine-language Programs".

Redirecting Input and Output

The PRINTER command, activated by a CONTROL-P, diverts all output normally destined for the screen to an interface card in a specified exapnsion slot, from 1 to 7. There must be an interface card in the specified slot, or you will lose control of the computer and your program and variables may be lost. The format of the command is

{slot number} CONTROL-P

A PRINTER command to slot number 0 will switch the stream of output characters back to the Apple IIe's video display.

Warning

Don't give the PRINTER command with slot number 0 to deactivate the 80-column firmware, even though you used this command to activate it in slot 3. The command works, but it just disconnects the firmware, leaving some of the soft-switches set for 80-column display.

In much the same way that the PRINTER command switches the output stream, the KEYBOARD command substitutes the interface card in a specified expansion slot for the Apple IIe's normal input device, the keyboard. The format for the KEYBOARD command is:

{slot number} CONTROL -K

A slot number of 0 for the KEYBOARD command directs the Monitor to accept input from the Apple IIe's built-in keyboard.

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

Hexadecimal Arithmetic

The Monitor will also perform one-byte hexadecimal addition and subtraction. Just type a line in one of these formats:

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

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

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

Special Tricks with the Monitor

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

Multiple Command Lines

You can put as many Monitor commands on a single line as you like, as long as you separate them with spaces and the total number of characters in the line is less than 254. Adjacent single-letter commands such as `L`, `S`, `I`, and `N` need not be separated by spaces.

You can freely intermix all of the commands except the `STORE` (`:`) command. Since the Monitor takes all values following a colon and places them in consecutive memory locations, the last value in a `STORE` must be followed by a letter command before another address is encountered. You can use the `NORMAL` command as the required letter command in such cases; it usually has no effect and can be used anywhere.

In the following example, you display a range of memory, change it, and display it again, all with one line of commands.

```
*300.307 300:18 69 1 N 300.302
0300- 00 00 00 00 00 00 00 00
0300- 18 69 01
*
```

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

Filling Memory

The `MOVE` command can be used to replicate a pattern of values throughout a range of memory. To do this, first store the pattern in the first locations in the range:

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

Remember the number of values in the pattern: in this case, it is 3. Use the number to compute addresses for the MOVE command, like this:

```
{start+number} < {start} . {end-number} M
```

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

```
*303<300.32DM

*300.32F

0300- 11 22 33 11 22 33 11 22
0308- 33 11 22 33 11 22 33 11
0310- 22 33 11 22 33 11 22 33
0318- 11 22 33 11 22 33 11 22
0320- 33 11 22 33 11 22 33 11
0328- 22 33 11 22 33 11 22 33
*
```

You can do a similar trick with the VERIFY command to check whether a pattern repeats itself through memory. This is especially useful to verify that a given range of memory locations all contain the same value. In this example,-you first fill the memory range from $300 to $320 with zeros and verify it, then change one location and verify again, to see the VERIFY command detect the discrepancy:

```
*300:0

*301<300.31FM

*301<300.31FV

*304:02

*301<300.31FV

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

Repeating Commands

You can create a command line that repeats one or more commands over and over. You do this by beginning the part of the command line that you want to repeat with a letter command, such as N, and ending it with the sequence 34:n, where n is a hexadecimal number that specifies the position in the line of the command where you want to start repeating; for the first character in the line, n=0. The value for n must be followed with a space in order for the loop to work properly.

This trick takes advantage of the fact that the Monitor uses an index register to step through the input buffer, starting at location $200. Each time the Monitor executes a command, it stores the value of the index at location $34; when that command is finished, the Monitor reloads the index register with the value at location $34. By making the last command change the value at location $34, you change this index so that the Monitor picks up the next command character from an earlier point in the buffer.

The only way to stop a loop like this is to press CONTROL - RESET ; that is how this example ends.

```
*N 300 302 34:0

0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
0300- 11
0302- 33
030
*
```

Creating your Own Commands

The USER command, CONTROL -Y , forces the Monitor to jump to memory location $3F8. You can put a JMP instruction there that jumps to your own machine-language program. Your program can then examine the Monitor's registers and pointers or the input buffer itself to obtain its data. For example, here is a

program that displays everything on the input line after the CONTROL-Y . The program starts at location $300; the command line that starts with $3F8 stores a jump to $300 at location $3F8.

```
*300:A4 34 B9 00 02 20 ED FD C8 C9 8D D0 F5 4C 69 FF

*3F8:4C 00 03

*CONTROL-Y THIS IS A TEST

THIS IS A TEST
*
```

Machine-Language Programs

The main reason to program in machine language is to get more speed. A program in machine language can run much faster than the same program written in high-level languages such as BASIC or Pascal, but the machine-language version usually takes a lot longer to write. There are other reasons to use machine language: you might want your program to do something that isn't included in your high-level language, or you might just enjoy the challenge of using machine language to work directly on the bits and bytes.

> If you have never used machine language before, you'll need to learn the 6502 instructions listed in Appendix A. To become proficient at programming in machine language, you'll have to spend some time at it, and study one of the books on 6502 programming listed in the Bibliography.

You can get a hexadecimal dump of your program, move it around in memory, or save it on tape and recall it again using the commands described in the previous sections. The Monitor commands in this section are intended specifically for you to use in creating, writing, and debugging machine-language programs.

Running a Program

The Monitor command you use to start execution of your machine-language program is the GO command. When you type an address and the letter G, the Apple IIe starts executing machine language instructions starting at the specified location. If you just type the G, execution starts at the last opened location.

The Monitor treats this program as a subroutine: it should end with an RTS (return from subroutine) instruction to transfer control back to the Monitor.

The Monitor has some special features that make it easier for you to write and debug machine-language programs, but before you get into that, here is a small machine-language program that you can run using only the simple Monitor commands already described. The program in the example merely displays the letters A through Z: you store it starting at location $300, examine it to be sure you typed it correctly, then type 300G to start it running.

```
*300:A9 C1 20 ED FD 18 69 1 C9 DB D0 F6 60

*300.30C

0300- A9 C1 20 ED FD 18 69 01
0308- C9 DB D0 F6 60

*300G

ABCDEFGHIJKLMNOPQRSTUVWXYZ
*
```

Disassembled Programs

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

Machine-language code in hexadecimal isn't the easiest thing in the world to read and understand. To make this job a little easier, machine-language programs are usually written in assembly language and converted into machine-language code by programs called assemblers.

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

The Monitor's LIST command displays machine-language code in assembly-language form. Instead of unformatted hexadecimal gibberish, the LIST command displays each instruction on a separate line, with a three-letter instruction name, or mnemonic, and a formatted hexadecimal operand. The LIST command also converts the relative addresses used in branch instructions to absolute addresses.

The Monitor LIST command has the format:

{location} L

The LIST command starts at the specified location and displays as much memory as it takes to make up a screenfull (20 lines) of instructions, as shown in the following example:

```
*300L

0300-   A9 C1      LDA   #$C1
0302-   20 ED FD   JSR   $FDED
0305-   18         CLC
0306-   69 01      ADC   #$01
0308-   C9 DB      CMP   #$DB
030A-   D0 F6      BNE   $0302
030C-   60         RTS
030D-   00         BRK
030E-   00         BRK
030F-   00         BRK
0310-   00         BRK
0311-   00         BRK
0312-   00         BRK
0313-   00         BRK
0314-   00         BRK
0315-   00         BRK
0316-   00         BRK
0317-   00         BRK
0318-   00         BRK
0319-   00         BRK
*
```

The first seven lines of this example are the assembly-language form of the program you typed in the previous example. The rest of the lines are BRK instructions only if this part of memory has zeros in it: other values will be disassembled as other instructions.

The Monitor saves the address that you specify in the LIST command, but not as the last opened location used by the other commands. Instead, the Monitor saves this address as the program counter, which it uses only to point to locations within programs. Whenever the Monitor performs a LIST command, it sets the program counter to point to the location immediately following the last location displayed on the screen, so that if you type another LIST command it will display another screenfull of instructions, starting where the previous display left off.

The Mini-assembler

Without an assembler, you have to write your machine language program, take the hexadecimal values for the opcodes and operands, and store them in memory using the commands covered in the previous sections. That is exactly what you did when you ran the previous examples.

The Integer BASIC interpreter includes an assembler called the Apple Mini-assembler that enables you to type programs into the Apple IIe using the same assembly-language format that the `LIST` command displays. It is called a mini-assembler because it doesn't include symbolic labels, an important feature of all full-sized assemblers like the Assembler/Editor in the DOS Tool Kit (Apple product number A2D0029).

Before you can use the Mini-assembler, the Apple IIe has to be running Integer BASIC. When you start up the computer using DOS or either BASIC, the Apple IIe loads the Integer BASIC interpreter from the file named `INTBASIC` into the bank-switched RAM.

To run Integer BASIC after you have started up the computer with DOS, type

```
INT
```

The Apple IIe displays the Integer prompt character (>) and a cursor.

If you have not activated the 80-column firmware since you started up with DOS, the cursor now looks like a blinking rectangle: it is actually a space character displayed in flashing format. This is evidence that the old Monitor is operating (see Chapter 3). The old Monitor is loaded into RAM along with Integer BASIC and the mini-assembler; the next time you activate the 80-column firmware, it copies the current version of the Monitor from ROM into RAM. Once this has happened, the current Monitor is active even with Integer BASIC, and the cursor is either a blinking checkerboard or a steady rectangle.

Now enter the Monitor by typing

```
CALL -151
```

After you enter the Monitor from Integer BASIC, invoke the Mini-assembler by typing:

```
F666G
```

This is just the GO command described above starting the program stored at location $F666 — the mini assembler. You can tell that the mini-assembler is running because it displays an exclamation point (!) as its prompt character. While the mini-assembler is running, you can execute any Monitor command by preceding it with a dollar sign ($). Aside from that, the Mini-assembler has an instruction set and syntax all its own.

The Mini-assembler saves one address, that of the program counter. Before you start to type a program, you must set the program counter to point to the location where you want the Mini-assembler to store your program. Do this by typing the address followed by a colon.

After the colon, type the mnemonic for the first instruction in your program, followed by a space and the operand of the instruction (formats for operands are listed Table 5-1). Now press RETURN. The Mini-assembler converts the line you typed into hexadecimal, stores it in memory beginning at the location of the Program Counter, and then disassembles it again and displays the disassembled line. It then displays a prompt on the next line.

Now the Mini-assembler is ready to accept the second instruction in your program. To tell it that you want the next instruction to follow the first, don't type an address or a colon: just type a space and the next instruction's mnemonic and operand, then press RETURN. The Mini-assembler assembles that line and waits for another.

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

```
!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 $C030
0307-   8D 30 C0   STA   $C030
! BPL $302
030A-   10 F6      BPL   $0302
! BRK
030C-   00         BRK
!
```

There are two ways to leave the Mini-assembler and re-enter the Monitor. One way is to type the Monitor command, FF69G, preceded by a dollar sign:

```
!$FF69G
*
```

Another way to leave the Mini-assembler is to press CONTROL - RESET, which warm-starts BASIC, then type

```
CALL-151
```

Your assembly language program is now stored in memory. You can display it with the `LIST` command:

```
*300L

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

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 Synertek Programming Manual (Apple part number A2L0003), but the addressing formats are somewhat different. Table 5-1 shows the Apple standard address-mode formats for 6502 assembly language.

An address consists of one or more hexadecimal digits. The Mini-assembler interprets addresses the same way the Monitor does: if an address has fewer than four digits, the Mini-assembler adds leading zeros; if the address has more than four digits, then it uses only the last four.

> In this book, dollar signs (`$`) in addresses signify that the addresses are in hexadecimal notation. They are ignored by the Mini-assembler and may be omitted when typing programs.

There is no syntactical distinction between the absolute and zero-page addressing modes. If you give an instruction to the Mini-assembler that can be used in both absolute and zero-page mode, the Mini-assembler assembles that instruction in absolute mode if the operand for that instruction is greater than `$FF`, and it assembles it in zero-page mode if the operand is less than `$100`.

Table 5-1 Mini-assembler Address Formats

*Note: Accumulator and Implied-address instructions have no operands.

Addressing Mode	Format	Notes
Accumulator		*
Implied		*
Immediate	#${value}	
Absolute	${address}	
Zero page	${address}	
Indexed zero page	${address}, X ${address}, Y	
Indexed absolute	${address}, X ${address}, Y	
Relative	${address}	
Indexed indirect	(${address}, X)	
Indirect indexed	(${address}), Y	
Absolute indirect	(${address})	

Instructions in accumulator mode and implied addressing mode need no operands.

Branch instructions, which use the relative addressing mode, require the target address of the branch. The Mini-assembler calculates the relative distance to use in the instruction automatically. If the target address is more than 127 locations distant from the instruction, the Mini-assembler sounds a bell (beep), displays a circumflex (^) under the target address, and does not assemble the line.

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

Summary of Monitor Commands

Here is a summary of the Monitor commands, showing the syntax diagram for each one. The Mini-assembler commands are included, even though they are only available when Integer BASIC is active (see the section "The Mini-assembler").

Examining Memory

{adrs}	Examines the value contained in one location.
{adrs1}.{adrs2}	Displays the values contained in all locations between {adrs1} and {adrs2}.
RETURN	Displays the values in up to eight locations following the last opened location.

Changing the Contents of Memory

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

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}.

The Register Command

CONTROL -E	Displays the locations where the contents of the 6502's registers are stored and opens them for changing.

Cassette Tape Commands

{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.

Miscellaneous Monitor Commands

I	Sets Inverse display mode.
N	Sets Normal display mode.
CONTROL -B	Enters the language currently active (usually Applesoft).
CONTROL -C	Returns to the language currently active (usually Applesoft).
{val}+{val}	Adds the two values and prints the hexadecimal result.
{val}−{val}	Subtracts the second value from the first and prints the result.

{slot} CONTROL -P	Diverts output to the device whose interface card is in slot number {slot}. If {slot}=0, accepts input from the keyboard.
CONTROL -Y	Jumps to the machine language subroutine at location $3F8.

Running and Listing Programs

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

The Mini-assembler

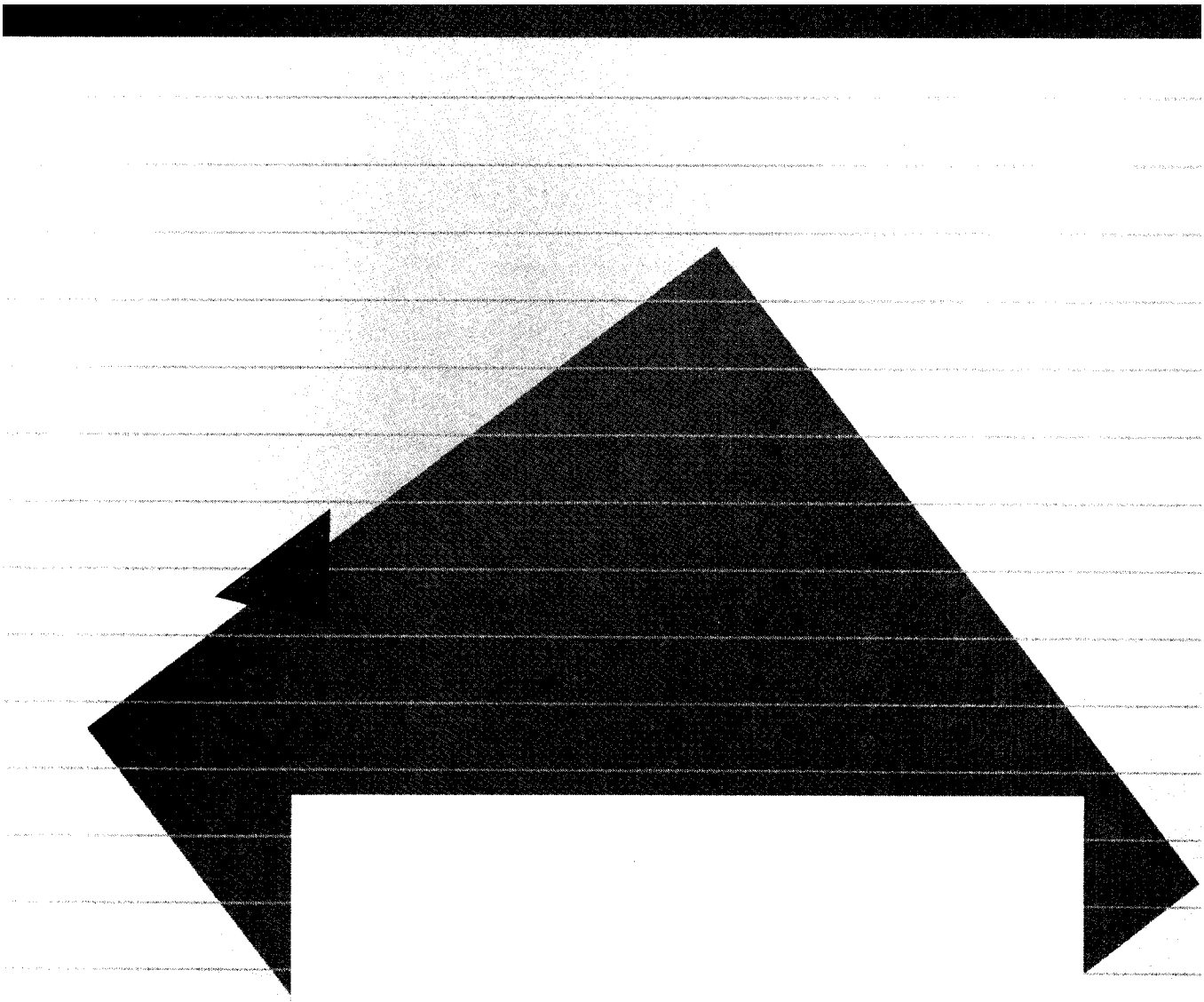
The Mini-assembler is only available when Integer BASIC is active.

F666G	Invokes the Mini-assembler.
${command}	Executes a Monitor command from the Mini-assembler.
$FF69G	Leaves the Mini-assembler.

BLANK PAGE

Chapter 6

Programming for Peripheral Cards



Chapter 6

Programming for Peripheral Cards

The seven expansion slots on the Apple IIe's main circuit board are used for installing circuit cards containing the hardware and firmware needed to interface peripheral devices to the Apple IIe. These slots are not simple I/O ports; peripheral cards can access the Apple IIe's data, address, and control lines via these slots. The expansion slots are numbered from 1 to 7, and certain signals, described below, are used to select a specific slot.

The older Apple II and Apple II Plus models have an eighth expansion slot: slot number 0. On those models, slot 0 is normally used for a language card or a ROM card; the functions of the Apple II Language Card are built into the main circuit board of the Apple IIe.

Peripheral-card Memory Spaces

Because the Apple IIe's 6502 microprocessor does all of its I/O through memory locations, portions of the Apple IIe's memory space have been allocated for the exclusive use of the cards in the expansion slots. In addition to the memory locations used for actual I/O, there are memory spaces available for programmable memory (RAM) in the main memory and for read-only memory (ROM or PROM) on the peripheral cards themselves.

The memory spaces allocated for the peripheral cards are described below. Those memory spaces are used for small dedicated programs such as I/O drivers. Peripheral cards that contain their own driver routines in firmware like this are called intelligent peripherals. They make it possible for you to add peripheral hardware to your Apple IIe without having to change your programs, provided that your programs follow normal practice for data input and output.

Peripheral-card I/O Space

Each expansion slot has the exclusive use of sixteen memory locations for data input and output in the memory space beginning at location $C090. Slot 1 uses locations $C090 through $C09F, slot 2 uses locations $C0A0 through $C0AF, and so on through location $C0FF, as shown in Table 6-1.

These memory locations are used for different I/O functions, depending on the design of each peripheral card. Whenever the Apple IIe addresses one of the sixteen I/O locations allocated to a particular slot, the signal on pin 41 of that slot, called DEVICE SELECT', switches to the active (low) state. This signal can be used to enable logic on the peripheral card that uses the four low-order address lines to determine which of its sixteen I/O locations is being accessed.

Table 6-1 Peripheral-card I/O Memory Locations

Note: The enabling signal is marked with a prime, to indicate that it is an active-low signal.

Slot	Locations	Enabled by
1	$C090-$C09F	DEVICE SELECT'
2	$C0A0-$C0AF	DEVICE SELECT'
3	$C0B0-$C0BF	DEVICE SELECT'
4	$C0C0-$C0CF	DEVICE SELECT'
5	$C0D0-$C0DF	DEVICE SELECT'
6	$C0E0-$C0EF	DEVICE SELECT'
7	$C0F0-$C0FF	DEVICE SELECT'

Peripheral-card ROM Space

One 256-byte page of memory space is allocated to each peripheral card. This space is normally used for read-only memory (ROM or PROM) on the card with driver programs that control the operation of the peripheral device connected to the card.

The page of memory allocated to each expansion slot begins at location $Cn00, where n is the slot number, as shown in Table 6-2 and Figure 6-3. Whenever the Apple IIe addresses one of the

256 ROM memory locations allocated to a particular slot, the signal on pin 1 of that slot, called I/O SELECT', switches to the active (low) state. This signal enables the ROM or PROM devices on the card, and the eight low-order address lines determine which of the 256 memory locations is being accessed.

Table 6-2 Peripheral-card ROM Memory Locations

Note: The enabling signal is marked with a prime, to indicate that it is an active-low signal.

Slot	Locations	Enabled by
1	$C100-$C1FF	I/O SELECT'
2	$C200-$C2FF	I/O SELECT'
3	$C300-$C3FF	I/O SELECT'
4	$C400-$C4FF	I/O SELECT'
5	$C500-$C5FF	I/O SELECT'
6	$C600-$C6FF	I/O SELECT'
7	$C700-$C7FF	I/O SELECT'

If there is an 80-column text card installed in the auxiliary slot, some of the functions normally associated with slot 3 are performed by the 80-column text card and the built-in 80-column firmware. With a 80-column text card installed, the I/O SELECT' signal is not available for slot 3, so firmware in ROM on a card in slot 3 will not run.

Expansion ROM Space

In addition to the small areas of ROM memory allocated to each expansion slot, peripheral cards can use the 2K-byte memory space from $C800 to $CFFF for larger programs in ROM or PROM. This memory space is called expansion ROM space (see the memory map in Figure 6-3). Besides being larger, the expansion ROM memory space is always at the same locations regardless of which slot is occupied by the card, making programs that occupy this memory space easier to write. (See the section "I/O Programming Suggestions", below.)

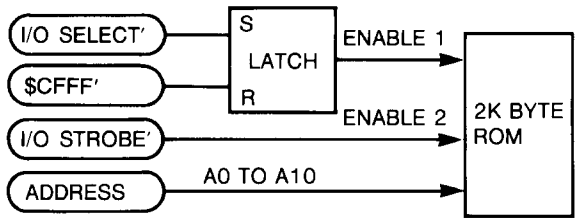
This memory space is available to any peripheral card that needs it. More than one peripheral card can have expansion ROM on it, but only one of them can be active at a time.

Each peripheral card that uses expansion ROM must have a circuit on it to enable the ROM. The circuit does this by a two-stage process: first, it sets a flip-flop when the I/O SELECT' signal, pin 1 on the slot, becomes active (low); second, it enables the expansion ROM devices when the I/O STROBE' signal, pin 20 on the slot, becomes active (low). Figure 6-1 shows a typical ROM-enable circuit.

The I/O SELECT' signal on a particular slot becomes active whenever the Apple IIe's 6502 microprocessor addresses a location in the 256-byte ROM address space allocated to that slot. The I/O STROBE' signal on **all** of the expansion slots becomes active (low) when the 6502 addresses a location in the expansion-ROM memory space, $C800-$CFFF. The I/O STROBE' signal is used to enable the expansion-ROM devices on a peripheral card (see Figure 6-1).

If there is an 80-column text card installed in the auxiliary slot, some of the functions normally associated with slot 3 are performed by the text card and the built-in 80-column firmware. With the text card installed, the I/O STROBE' signal is not available on slot 3, so firmware in expansion ROM on a card in slot 3 will not run.

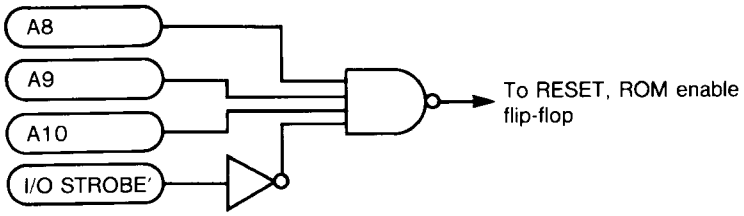
Figure 6-1 Expansion ROM Enable Circuit



A program on a peripheral card can get exclusive use of the expansion ROM memory space by referring to location $CFFF in its initialization phase. This location is special: all peripheral cards that use expansion ROM must recognize a reference to $CFFF as a signal to reset their ROM-enable flip-flops and disable their expansion ROMs. Of course, doing so also disables the expansion ROM on the card that is about to use it, but the next instruction in the initialization code sets the flip-flop on the expansion-ROM enable circuit on the card. Once this has been done, this card has exclusive use of the expansion memory space and its program can jump directly into the expansion ROM.

As described above, the expansion-ROM disable circuit resets the enable flip-flop whenever the 6502 addresses location $CFFF.

Figure 6-2 ROM Disable Address Decoding



To do this, the peripheral card must detect the presence of $CFFF on the address bus. You can use the I/O STROBE' signal for part of the address decoding, since it is active for addresses from $C800 through $CFFF. If you can afford to sacrifice some ROM space, your can simplify the address decoding even further and save circuitry on the card. For example, if you give up the last 256 bytes of expansion ROM space, your disable circuit only needs to detect addresses of the form $CFxx, and you can use the minimal disable-decoding circuitry shown in Figure 6-2.

Peripheral-card RAM Space

There are 56 bytes of main memory allocated to the peripheral cards, eight bytes per card, as shown in Table 6-3. These 56 locations are actually in the RAM memory reserved for the text and low-resolution graphics displays, but these particular locations are not displayed on the screen and their contents are not changed by the built-in output routine COUT1. Programs in ROM on peripheral cards use these locations for temporary data storage.

Table 6-3 Peripheral-card RAM Memory Locations

*Note: The RAM locations normally allocated to slot 3 are taken over by any card installed in the auxiliary slot.

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

A program on a peripheral card can use the eight base addresses shown in the table to access the eight RAM locations allocated for its use, as shown in the next section, "I/O Programming Suggestions".

I/O Programming Suggestions

A program in ROM on a peripheral card should work no matter which slot the card occupies. If the program includes a jump to an absolute location in one of the 256-byte memory spaces, then the card will only work when it is plugged into the slot that uses that memory space. If you are writing the program for a peripheral card that will be used by many people, you should avoid placing such a restriction on the use of the card.

> To function properly no matter which slot a peripheral card is installed in, the program in the card's 256-byte memory space must not make any absolute references to itself. Instead of using jump instructions, you should force conditions on branch instructions, which use relative addressing.

The first thing a peripheral-card subroutine should do is to save the contents of the 6502's registers. One way to do this is to use the monitor subroutine `IOSAVE`. This subroutine, which starts at location `$FF4A`, stores the registers in zero-page memory locations `$45-$49`. A companion subroutine, `IOREST`, restores the registers from these memory locations. Your program should call `IOREST`, which starts at location `$FF3F`, just before it returns control to the program that called it.

This method of saving the registers is convenient, but it is not always safe. If a second subroutine calls `IOSAVE`, or if an interrupt occurs, the new register contents get saved in the same locations, and the old ones get destroyed. It is safer, though somewhat slower, to save the registers on the stack, and restore them just before returning control to the calling program.

Most single-character I/O is done via the 6502's accumulator. A character being output through your subroutine will be in the accumulator with its high bit set when your subroutine is called. Likewise, if your subroutine is performing character input, it must leave the character in the accumulator with its high bit set when it returns to the calling program.

Finding the Slot Number

The memory addresses used by a program on a peripheral card differ depending on which expansion slot the card is installed in. Before it can refer to any of those addresses, the program must somehow determine the correct slot number. One way to do this is to execute a JSR (Jump to Subroutine) to a location with an RTS (Return from Subroutine) instruction in it, and then derive the slot number from the return address saved on the stack, as shown in the following example.

```
PHP              ; save status
SEI              ; inhibit interrupts
JSR $FF58        ; -> a known RTS instruction
TSX              ; get high byte of the...
LDA $0100,X      ; ...return address from stack
AND #$0F         ; low-order digit is slot no.
PLP              ; restore status
```

The slot number can now be used in addressing the memory allocated to the peripheral card, as shown below.

I/O Addressing

Once your peripheral-card program has the slot number, it can use it to address the I/O locations allocated to the slot. Table 6-4 shows how these locations are related to sixteen base addresses starting with $C080. Notice that the difference between the base address and the desired I/O location has the form $n0, where n is the slot number. Starting with the slot number in the accumulator, the following example computes this difference by four left shifts, then loads it into an index register and uses the base address to specify one of sixteen I/O locations.

```
ASL              ; get n into...
ASL              ;
ASL              ;
ASL              ; ...high-order nybble...
TAX              ; ... of index register.
LDA $C080,X      ; load from first I/O location
```

You must make sure that you get an appropriate value into the index register when you address I/O locations this way. For example, starting with 1 in the accumulator, the instructions in the above example perform an LDA from location $C090, the first I/O location allocated to slot 1. If the value in the accumulator had been 0, the LDA would have accessed location $C080, thereby setting the soft switch that selects the second bank of RAM at location $D000 and enables it for reading (see Chapter 5).

Table 6-4 Peripheral-card I/O Base Addresses

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

RAM Addressing

A program on a peripheral card can use the eight base addresses shown in Table 6-3 to access the eight RAM locations allocated for its use. The program does this by putting its slot number into the Y index register and using indexed addressing mode with the base addresses. The base addresses can be defined as constants because they are the same no matter which slot the peripheral card occupies.

If you start with the correct slot number in the accumulator (by using the example shown earlier), the following example uses all eight RAM locations allocated to the slot.

```
TAY
LDA   $0478,Y
STA   $04F8,Y
LDA   $0578,Y
STA   $05F8,Y
LDA   $0678,Y
STA   $06F8,Y
LDA   $0778,Y
STA   $07F8,Y
```

Warning

Peripheral-card programs must not store data at the base-address locations themselves; the RAM at those locations is used by the Disk Operating System. DOS stores the first byte of the ROM location of the expansion slot that is currently active ($Cn) in location $7F8, and the first byte of the ROM location of the slot holding the controller card for the startup disk drive in location $5F8.

Changing the Standard I/O Links

There are two pairs of locations in the Apple IIe that are used for controlling character input and output. They are called the I/O links (see Chapter 3). In a Apple IIe running without a Disk Operating System, the I/O links normally contain the starting addresses of the standard input and output routines KEYIN and COUT1. If a disk operating system is running, one or both of the links will hold the addresses of the DOS input and output routines.

The link at 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 link holds the starting address of the subroutine the Apple IIe is currently using for single-character output. This address is normally $FDF0, the address of routine COUT1, described in Chapter 3.

When you issue a PR#n from BASIC or an n CONTROL-P from the Monitor, the Apple IIe changes this link address to the first address in the ROM memory space allocated to slot number n. That address has the form $CnOO. Subsequent calls for character output are thus transferred to the program on the peripheral card. That program can use the instruction sequences given above to find its slot number and use the I/O and RAM locations allocated to it. When it is finished, the program can execute an RTS (Return

from Subroutine) instruction to return control to the calling program, or jump to the output routine COUT1 at location $FDF0 to display the output character (which must be in the accumulator) on the screen, then let COUT1 return to the calling program.

A similar link at locations $38 and $39 (decimal 56 and 57) is called KSW, for Keyboard input Switch. Individually, location $38 is called KSWL (for KSW Low) and location $39 is called KSWH (KSW High). This link holds the starting address of the routine currently being used for single-character input. This address is normally $FD1B, the starting address of the standard input routine KEYIN (see Chapter 3).

When you issue an IN#n command from BASIC or an n CONTROL -K from the monitor, the Apple IIe changes this link address to $Cn00, the beginning of the ROM memory space that is allocated to slot number n. Subsequent calls for character input are thus transferred to the program on the peripheral card. That program can use the instruction sequences given above to find its slot number and use the I/O and RAM locations allocated to it. The program should put the input character, with its high bit set, into the accumulator and execute an RTS (Return from Subroutine) instruction to return control to the program that requested input.

When the Disk Operating System (DOS) is running, one or both of the standard I/O links hold addresses of the Disk Operating System's input and output routines. The DOS has internal locations that hold the addresses of the character input and output routines that are currently active.

> If a program that is running with DOS changes the standard link addresses, either directly or via IN# and PR# commands, DOS is disconnected from the system.

To avoid disconnecting DOS each time they initiate I/O to a slot, BASIC programs that run with DOS must always issue an IN# or a PR# command from inside a PRINT statement that starts with a CONTROL -D character. For assembly-language programs, there is a DOS subroutine call to use when changing the link addresses. After changing CSW or KSW, the program calls this subroutine at location $3EA (decimal 1002). The subroutine transfers the link address to a location inside DOS and then restores the DOS address in the standard link location. Refer to the section on input and output link registers in the *DOS Manual* for further details.

Using Interrupts

Although programs running on the Apple IIe do not normally use interrupts, it is possible to do so. To use interrupts on the Apple IIe, your peripheral card must be able to send an interrupt request (IRQ') to the 6502 microprocessor, and you must store the address of your interrupt-handling routine in the user interrupt vector, as described below.

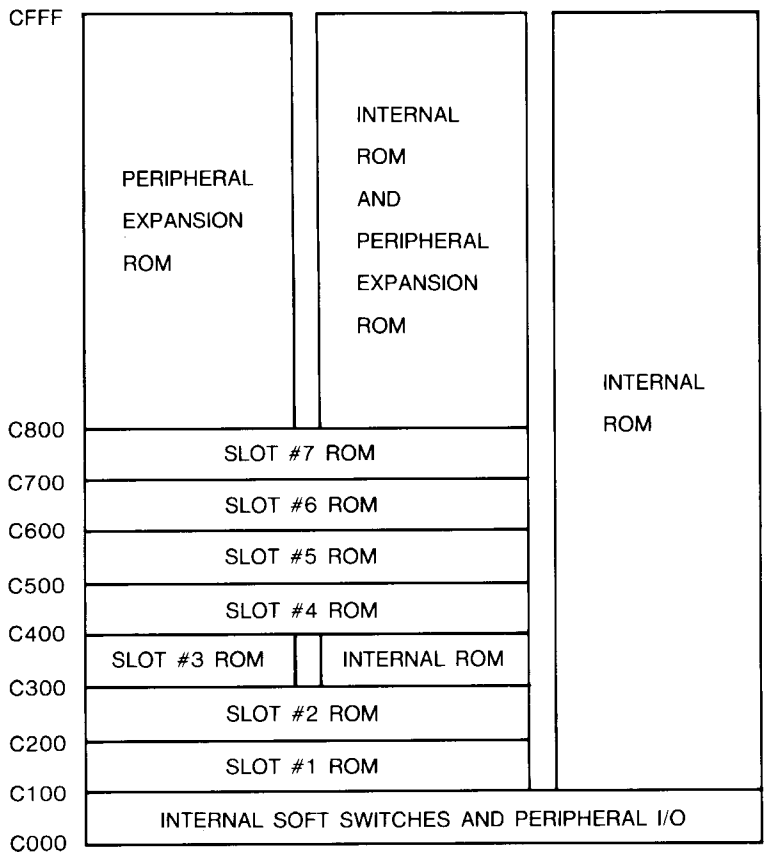
Interrupt priority is handled by a daisy-chain arrangement using two pins, INT IN and INT OUT, on each expansion slot. As described in Chapter 7, each peripheral card breaks the chain when it makes an interrupt request. On peripheral cards that don't use interrupts, these pins should be connected together. The daisy chain gives priority to the peripheral card in slot 7: if this card opens the connection between INT IN and INT OUT, or if there is no card in this slot, interrupt requests from cards in slots 1 through 6 can't get through. Similarly, slot 6 controls IRQ from slots 1 through 5, and so on down the line.

When the IRQ' line on the 6502 microprocessor is activated (pulled low), the 6502 transfers control through the vector in locations $FFFE-$FFFF. This vector is the address of the Monitor's interrupt handler, which determines whether the request is due to an external IRQ or a BRK instruction and transfers control to the appropriate routine via the vectors stored in memory page 3. The BRK vector is in locations $3F0-$3F1 and the IRQ vector is in locations $3FE-$3FF (see Table 4-10). The Monitor normally stores the address of its reset routine in the IRQ vector; you should substitute the address of your program's interrupt-handling routine.

Other Uses of I/O Memory Space

The portion of memory space from location $C000 through $CFFF (decimal 49152 through 53247) is normally allocated to I/O and program memory on the peripheral cards, but there are two other functions that also use this memory space: the built-in self-test firmware and the 80-column display firmware. The soft switches that control the allocation of this memory space are described below.

Figure 6-3 I/O Memory Map



Switching I/O Memory

The built-in firmware uses two soft switches to control the allocation of the I/O memory space from $C000 to $CFFF. The locations of these soft switches, SLOTCXROM and SLOTC3ROM, are given in Table 6-5.

Like the display switches described in Chapter 2, these soft switches share their locations with the keyboard data and strobe functions. The switches are activated only by writing, and the states can be determined only by reading, as indicated in Table 6-5.

When SLOTC3ROM is on, the 256-byte ROM area at $C300 is available to a peripheral card in slot 3, which is the slot normally used for a terminal interface. If a card is installed in the auxiliary slot when you turn on the power or reset the Apple IIe, the SLOT3ROM switch is turned off. Turning SLOTC3ROM off disables

Table 6-5 I/O Memory Switches

Name	Function	Location Hex	Location Decimal		Notes
SLOTC3ROM	Slot ROM at $C300	$C00B	49163	-16373	Write
	Internal ROM at $C300	$C00A	49162	-16374	Write
	Read SLOTC3ROM switch	$C017	49175	-16361	Read
SLOTCXROM	Slot ROM at $Cx00	$C007	49159	-16377	Write
	Internal ROM at $Cx00	$C006	49158	-16378	Write
	Read SLOTCXROM switch	$C015	49173	-16363	Read

peripheral-card ROM in slot 3 and enables the built-in 80-column firmware, as shown in Figure 6-3. The 80-column firmware is assigned to slot-3 address space because slot 3 is normally used with a terminal interface, so the built-in firmware will work with programs that use slot 3 this way.

> Installing an 80-column text card in the auxiliary slot makes it impossible to run any peripheral card that has built-in firmware in slot 3. If an 80-column text card is not installed, a peripheral card in slot 3 will work properly.

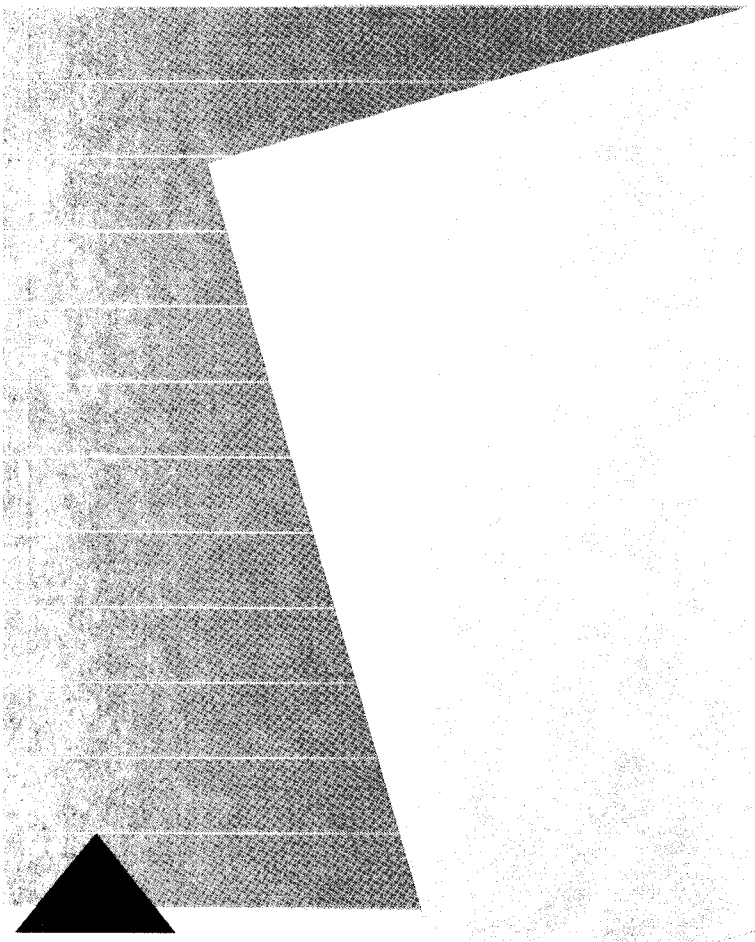
The bus and I/O signals are always available to a peripheral card in slot 3, even when the 80-column hardware and firmware are operating. Thus it is always possible to use this slot for any I/O accessory that does **not** have built-in firmware.

When SLOTCXROM is active (high), the I/O memory space from $C100 to $C7FF is allocated to the expansion slots, as described previously. Setting SLOTCXROM inactive (low) disables the peripheral-card ROM and selects built-in ROM in all of the I/O memory space except the part from $C000 to $C0FF (used for soft switches and data I/O), as shown in Figure 6-3. In addition to the 80-column firmware at $C300 and $C800, the built-in ROM includes firmware that performs the self-test of the Apple IIe's hardware.

> Setting SLOTCXROM low enables built-in ROM in all of the I/O memory space (except the soft-switch area), including the $C300 space, which contains the 80-column firmware.

Chapter 7

Hardware Implementation



Chapter 7

Hardware Implementation

Most of this manual describes functions — what the Apple IIe does. This chapter, on the other hand, describes objects: the pieces of hardware the Apple IIe uses to carry out its functions. If you are designing a piece of peripheral hardware to attach to the Apple IIe, or if you just want to know more about how the Apple IIe is built, you should study this chapter.

Environmental Specifications

The Apple IIe is quite sturdy when used in the way it was intended. Table 7-1 defines the conditions under which the Apple IIe is designed to function properly.

Table 7-1 Summary of Environmental Specifications

Operating Temperature:	0° to 45°C (30° to 115°F)
Relative Humidity:	5% to 85%
Line Voltage:	107 to 132 VAC

You should treat the Apple IIe with the same kind of care as any other electrical appliance. You should protect it from physical violence, such as hammer blows or defenestration. You should protect the mechanical keyboard and the electrical connectors inside the case from spilled liquids, especially those with dissolved contaminants, such as coffee and cola drinks.

In normal operation, enough air flows through the slots in the case to keep the insides from getting too hot, although some of the parts inside the Apple IIe normally get rather warm to the touch. If you manage to overheat your Apple IIe, by blocking the ventilation slots in the top and bottom for example, the first symptom will be erratic operation. The memory devices in the Apple IIe are sensitive to heat: when they get too hot, they

occasionally change a bit of data. The exact result depends on what kind of program you are running and on just which bit of memory is affected.

The Power Supply

The power supply in the Apple IIe operates on normal household AC power and provides enough low-voltage electrical power for the built-in electronics plus a full complement of peripheral cards, including disk controller cards and communications interfaces. The basic specifications of the power supply are listed in Table 7-2.

The Apple IIe's power cord should be plugged into a three-wire 110- to 120-volt outlet. You must connect the Apple IIe to a grounded outlet or to a good earth ground. Also, the line voltage must be in the range given in Table 7-2. If you try to operate the Apple IIe from a power source with more than 140 volts, you will damage the power supply.

Table 7-2 Power Supply Specifications

Line voltage:	107V to 132V AC
Maximum power consumption:	60W continuous 80W intermittent*
Supply voltages:	+5V ±3% +11.8V ±6% −5.2V ±10% −12V ±10%
Maximum supply currents:	+5V: 2.5A +12V: 1.5A continuous, 2.5A intermittent* −5V: 250mA −12V: 250mA
Maximum case temperature:	55°C (130°F)

*Intermittent operation: The Apple IIe can safely operate for up to twenty minutes at the higher load if followed by at least ten minutes at normal load.

The Apple IIe uses a custom-designed switching-type power supply. It is small and lightweight, and it generates less heat than other types of power supplies do.

The Apple IIe's power supply works by converting the AC line voltage to DC and using this DC voltage to power a variable-frequency oscillator. The oscillator drives a small transformer with many separate windings to produce the different voltages required. A circuit compares the voltage of the +5-volt supply with a reference voltage and feeds an error signal back to the

oscillator circuit. The oscillator circuit uses the error signal to control the frequency of its oscillation and keep the output voltages in their normal ranges.

The power supply includes circuitry to protect itself and the other electronic parts of the Apple IIe by turning off all four supply voltages whenever it detects one of the following malfunctions:

- any supply voltage short-circuited to ground;
- the power-supply cable disconnected;
- any supply voltage outside the normal range.

Any time one of these malfunctions occurs, the protection circuit stops the oscillator, and all the output voltages drop to zero. After about half a second, the oscillator starts up again. If the malfunction is still occurring, the protection circuit stops the oscillator again. The power supply will continue to start and stop this way until the malfunction is corrected or the power is turned off.

Warning
If you think the power supply is broken, do not attempt to repair it yourself. The power supply is in a sealed enclosure because some of its circuits are connected directly to the power line. Special equipment is needed to repair the power supply safely, so see your Apple dealer for service.

The Power Connector

The cable from the power supply is connected to the main circuit board by a six-pin connector with a strain-relief catch. The connector pins are identified in Table 7-3 and Figure 7-14d.

Table 7-3 Power Connector Signal Specifications

Pin Number	Name	Description
1,2	Ground	Common electrical ground
3	+5V	+5V from power supply
4	+12V	+12V from power supply
5	−12V	−12V from power supply
6	−5V	−5V from power supply

The 6502 Microprocessor

The Apple IIe uses a 6502B microprocessor as its central processing unit (CPU). The 6502B in the Apple IIe runs at a clock rate of 1.023 MHz and performs up to 500,000 eight-bit operations per second. You should not use the clock rate as a criterion for comparing different types of microprocessors. The 6502 has a simpler instruction cycle than most other microprocessors and it uses instruction pipelining for faster processing. The speed of the 6502 with a 1MHz clock is equivalent to other types of microprocessors with clock rates up to 2.5MHz.

The 6502 has a sixteen-bit address bus, giving it an address space of 64K (2 to the sixteenth power or 65536) bytes. The Apple IIe uses special techniques to address a total of more than 64K: see the sections "Bank-switched Memory" and "Auxiliary Memory and Firmware" in Chapter 4 and the section "Switching I/O Memory" in Chapter 6.

Table 7-4 6502 Microprocessor Specifications

Type:	6502B
Register complement:	Accumulator (A) Index Registers (X, Y) Stack Pointer (S) Processor Status (P)
Register size:	Eight bits
Data bus:	Eight bits wide
Address bus:	Sixteen bits wide
Address range:	65,536 (64K)
Interrupts:	IRQ (maskable) NMI (nonmaskable) BRK (programmed)
Operating voltage:	+5V (± 5%)
Power dissipation:	500mW (typical)

6502 Timing

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

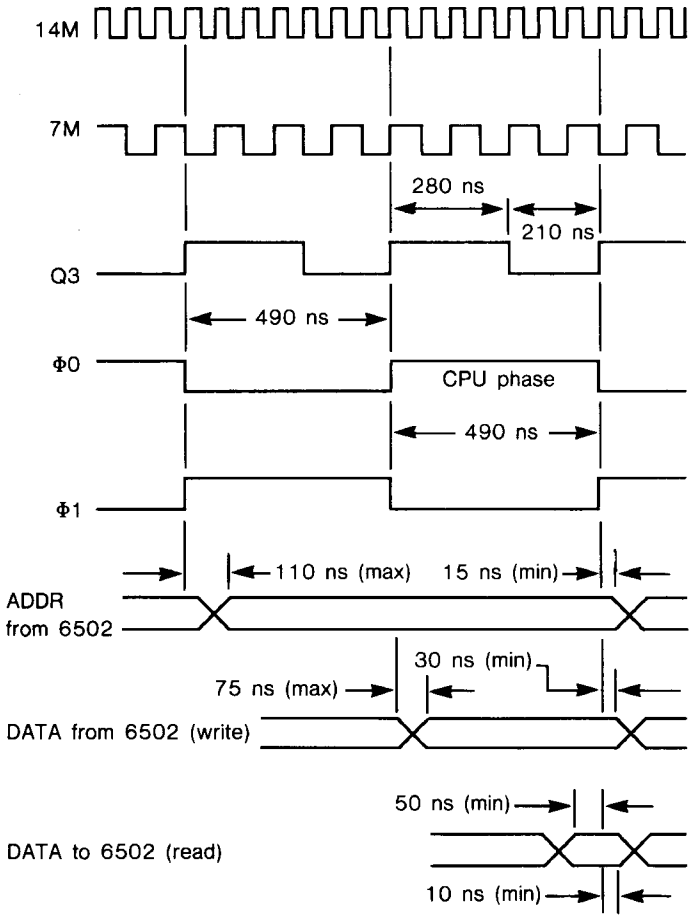
The frequency of the oscillator that generates the master timing signal is 14.31818 MHz. Circuitry in the Apple IIe uses this clock signal, called 14M, to produce all the other timing signals. These timing signals perform two major tasks: controlling the computing functions, and generating the video display. The timing signals directly involved with the operation of the 6502 are described in this section. Other timing signals are described in the sections "RAM Addressing", "Video Display Modes", and "The Expansion Slots".

The main 6502 timing signals are listed in Table 7-5, and their relationships are diagrammed in Figure 7-1. The 6502 clock signals are ϕ1 and ϕo, complementary signals at a frequency of 1.02273 MHz. If you need more information about the 6502 itself, refer to the Synertek *Hardware Manual* (Apple product number A2L0002). The Apple IIe signal named ϕo is equivalent to the signal called ϕ2 in the hardware manual (it isn't identical: it's a tiny bit early).

Table 7-5 6502 Timing Signal Descriptions

Signal Name	Description
14M	Master oscillator, 14.31818 MHz; also 80-column dot clock.
7M	Intermediate timing signal and 40-column dot clock.
Q3	Intermediate timing signal, 2.04545 MHz with asymmetrical duty cycle.
ϕ0	Phase 0 of 6502 clock, 1.022727 MHz; Complement of ϕ1.
ϕ1	Phase 1 of 6502 clock, 1.022727 MHz; Complement of ϕ0.

Figure 7-1 6502 Timing Signals

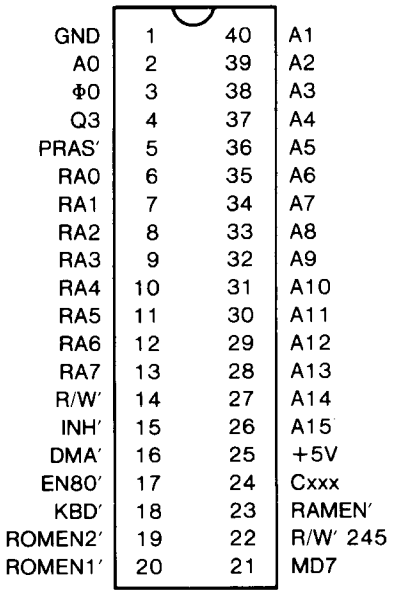


The operations of the 6502 are related to the clock signals in a simple way: address during ϕ1, data during ϕo. The 6502 puts an address on the address bus during ϕ1. This address is valid not later than 110 nanoseconds after ϕ1 goes high and remains valid through all of ϕo. The 6502 reads or writes data during ϕo. If the 6502 is writing, the read/write signal is low during ϕo and the 6502 puts data on the data bus. The data is valid not later than 75 nanoseconds after ϕo goes high. If the 6502 is reading, the read/write signal remains high. Data on the data bus must be valid no later than 50 nanoseconds before the end of ϕo.

The Custom Integrated Circuits

Most of the circuitry that controls memory and I/O addressing in the Apple IIe is in three custom integrated circuits called the Memory Management Unit (MMU), the Input-Output Unit (IOU), and the Programmed Array Logic device (PAL). The soft switches used for controlling the various I/O and addressing modes of the Apple IIe are addressable flags inside the MMU and the IOU. The functions of these two devices are not as independent as their names suggest; working together, they generate all of the addressing signals. For example, the MMU generates the address signals for the CPU, while the IOU generates similar address signals for the video display.

Figure 7-2 The MMU Pinouts



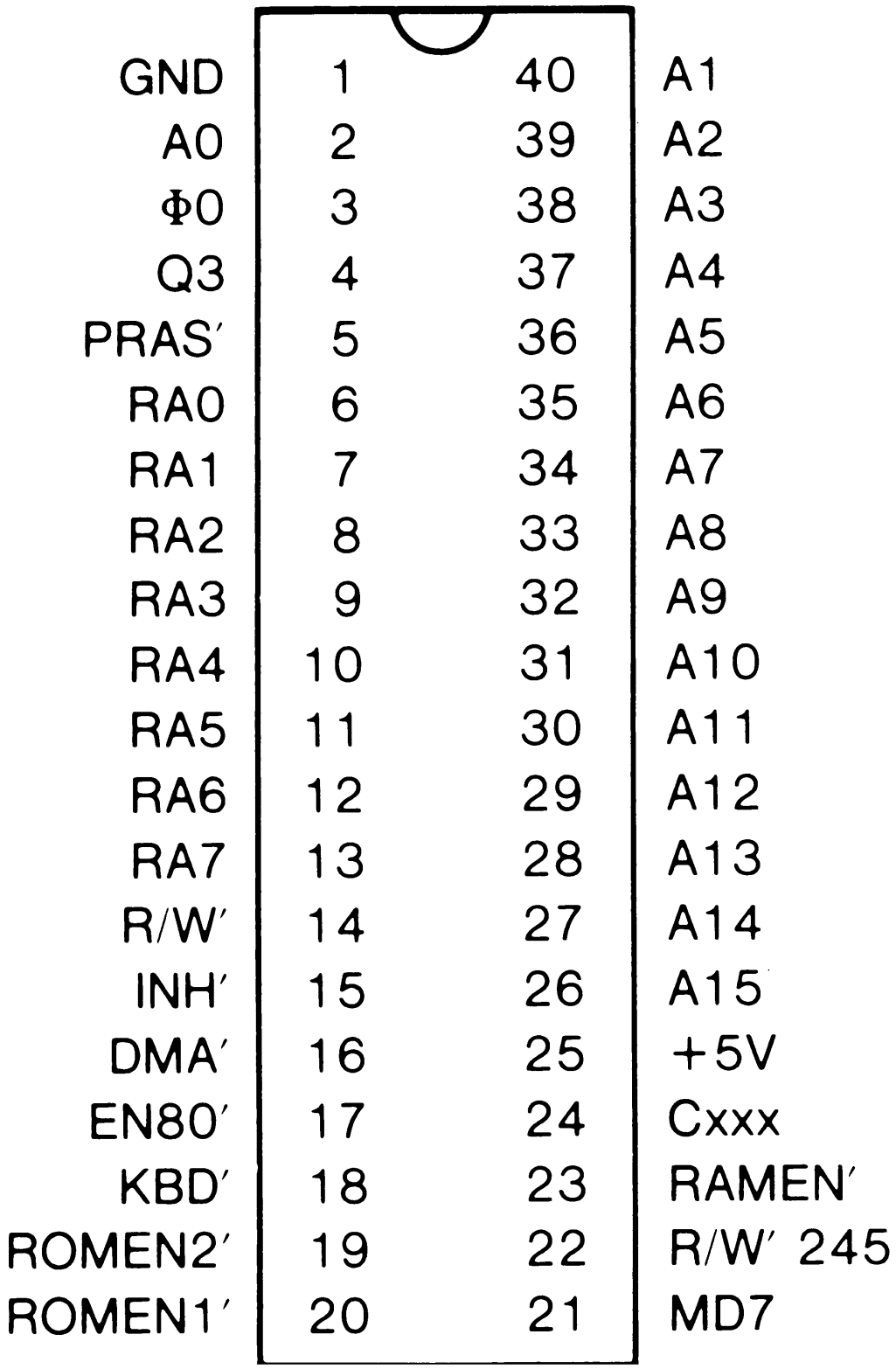
The Memory Management Unit

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

Page 2 display (PAGE2): Chapter 2
Hi-res mode (HIRES): Chapter 2
Store to 80-column card (80STORE): Chapter 2
Select bank 2: Chapter 4
Enable bank-switched RAM: Chapter 4
Read auxiliary memory (RAMRD): Chapter 4
Write auxiliary memory (RAMWRT): Chapter 4
Auxiliary stack and zero page (ALTZP): Chapter 4
Slot ROM for connector #3 (SLOTC3ROM): Chapter 6
Slot ROM in I/O space (SLOTCXROM): Chapter 6

The 64K dynamic RAMs used in the Apple IIe use a multiplexed address, as described below in the section "Dynamic-RAM Timing". The MMU generates this multiplexed address for memory reading and writing by the 6502 CPU.

Figure 7-2 The MMU Pinouts

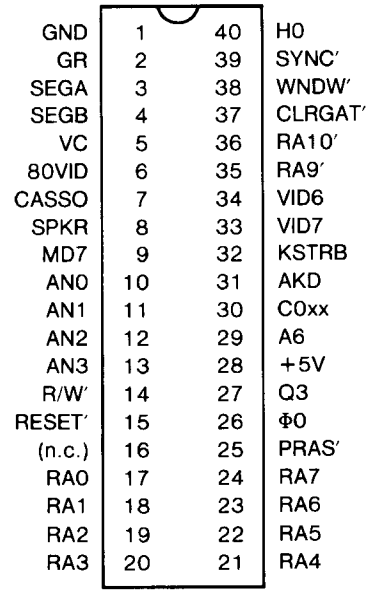


"A2_030-0357-B_1982_1-143a.pict" 213 KB 2002-10-27 dpi: 1200h x 1200v pix: 2348h x 3972v

Table 7-6 The MMU Signal Descriptions

Pin Number	Name	Description
1	GND	Power and signal common
2	A0	6502 address input
40-26	A1-A15	6502 address input
3	ϕ0	Clock phase 0
4	Q3	Timing signal
5	PRAS'	Memory Row-address strobe
6-13	RA0-RA7	Multiplexed address output
14	R/W'	6502 read-write control signal
15	INH'	Inhibits main memory
16	DMA'	Controls data bus for DMA transfers
17	EN80'	Enables auxiliary RAM
18	KBD'	Enables keyboard data bit 0-6
19	ROMEN2'	Enables built-in firmware ROM #2
20	ROMEN1'	Enables built-in firmware ROM #1
21	MD7	State of MMU flags
22	RW'245	Controls 74LS245 data-bus buffer
23	RAMEN'	Enables main RAM
24	CXXX	Enables peripheral-card memory
25	+5V	Power

Figure 7-3 The IOU Pinouts



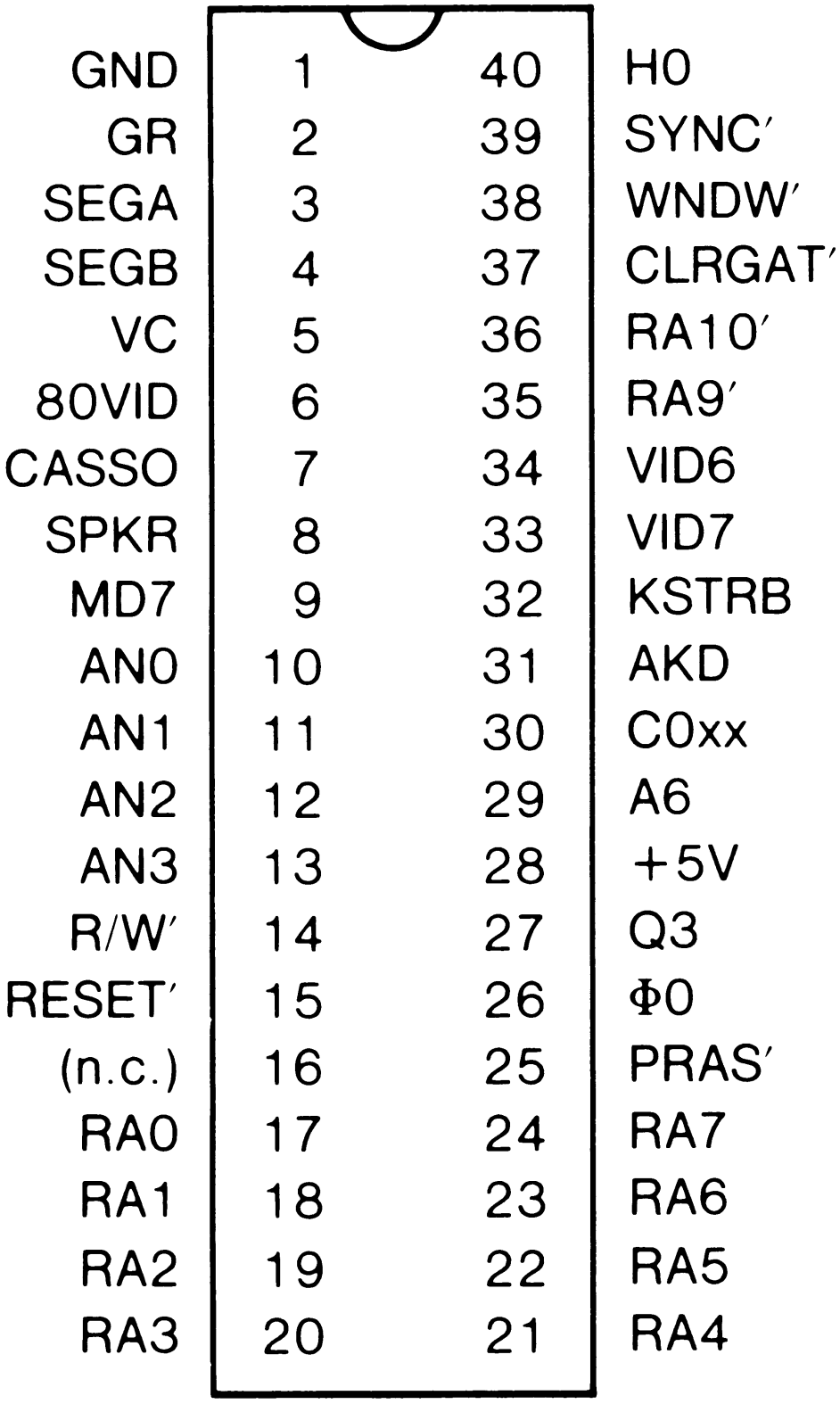
The Input/Output Unit

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

Page 2 display (PAGE2)
Hi-res mode (HIRES)
Text mode (TEXT)
Mixed mode (MIXED)
80-column display (80COL)
Character-set select (ALTCHARSET)
Any-key-down
Annunciators
Vertical blanking (VBL)

The 64K dynamic RAMs used in the Apple IIe require a multiplexed address, as described below in the section "Dynamic-RAM Timing". The IOU generates this multiplexed address for the data transfers required for display and memory refresh during clock phase 1. The way this address is generated is described below in the section "Video Display Generation".

Figure 7-3 The IOU Pinouts



"A2_030-0357-B_1982_1-145a.pict" 224 KB 2002-10-27 dpi: 1200h x 1200v pix: 2264h x 3965v

Table 7-7 The IOU Signal Descriptions

Note: Pin 16 is not connected.

Pin Number	Name	Description
1	GND	Power and signal common
2	GR	Graphics mode enable
3,4	SEGA, SEGB	Display vertical counter bits
5	VC	Display vertical counter bit
6	80VID'	80-column video enable
7	CASSO	Cassette output signal
8	SPKR	Speaker output signal
9	MD7	Internal flags to data bus
10-13	AN0-AN3	Annunciator outputs
14	R/W'	6502 read-write control signal
15	RESET'	Power on and reset output
17-24	RA0-RA7	Multiplexed RAM address (phase 0)
25	PRAS'	Row-address strobe (phase 0)
26	ϕ0	Master clock phase 0
27	Q3	Intermediate timing signal
28	+5V	Power
29	A6	Address bit 6 from 6502
30	C0XX'	I/O address enable
31	AKD	Any-key-down signal
32	KSTRB	Keyboard strobe signal
33, 34	VID7, VID6	Video display control bits
35, 36	RA9', RA10'	Video display control bits
37	CLRGAT'	Color-burst gate (enable)
38	WNDW'	Display blanking signal
39	SYNC'	Display synchronization signal
40	H0	Display horizontal timing signal

The PAL Circuit

A Programmed Array Logic device, type PAL 16R8, generates several timing and control signals in the Apple IIe. These signals are listed in Table 7-8.

Table 7-8 The PAL Signal Descriptions

Pin Number	Name	Description
1	14M	14.31818 MHz master timing signal
2	7M	7.15909 MHz timing signal
3	3.58M	3.579545 MHz timing signal
4	H0	Horizontal video timing signal
5	VID7	Video data bit 7
6	SEGB	Video timing signal
7	GR	Video display graphics-mode enable
8	RAMEN'	RAM enable (CAS enable)
9	80VID'	Enable 80-column display mode
10	GND	Power and signal common
11	ENTMG	Enable master timing
12	LDPS'	Video shift-register load enable
13	VID7M	Video dot clock, 7 or 14 MHz
14	ϕ1	Phase 1 system clock
15	ϕ0	Phase 0 system clock
16	Q3	Intermediate timing and strobe signal
17	PCAS'	RAM Column-address strobe
18	N. C.	(This pin is not used.)
19	PRAS'	RAM Row-address strobe
20	+5V	Power

Figure 7-4 The PAL Pinouts

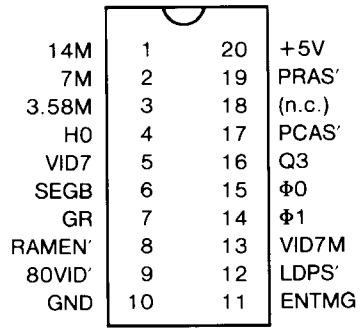
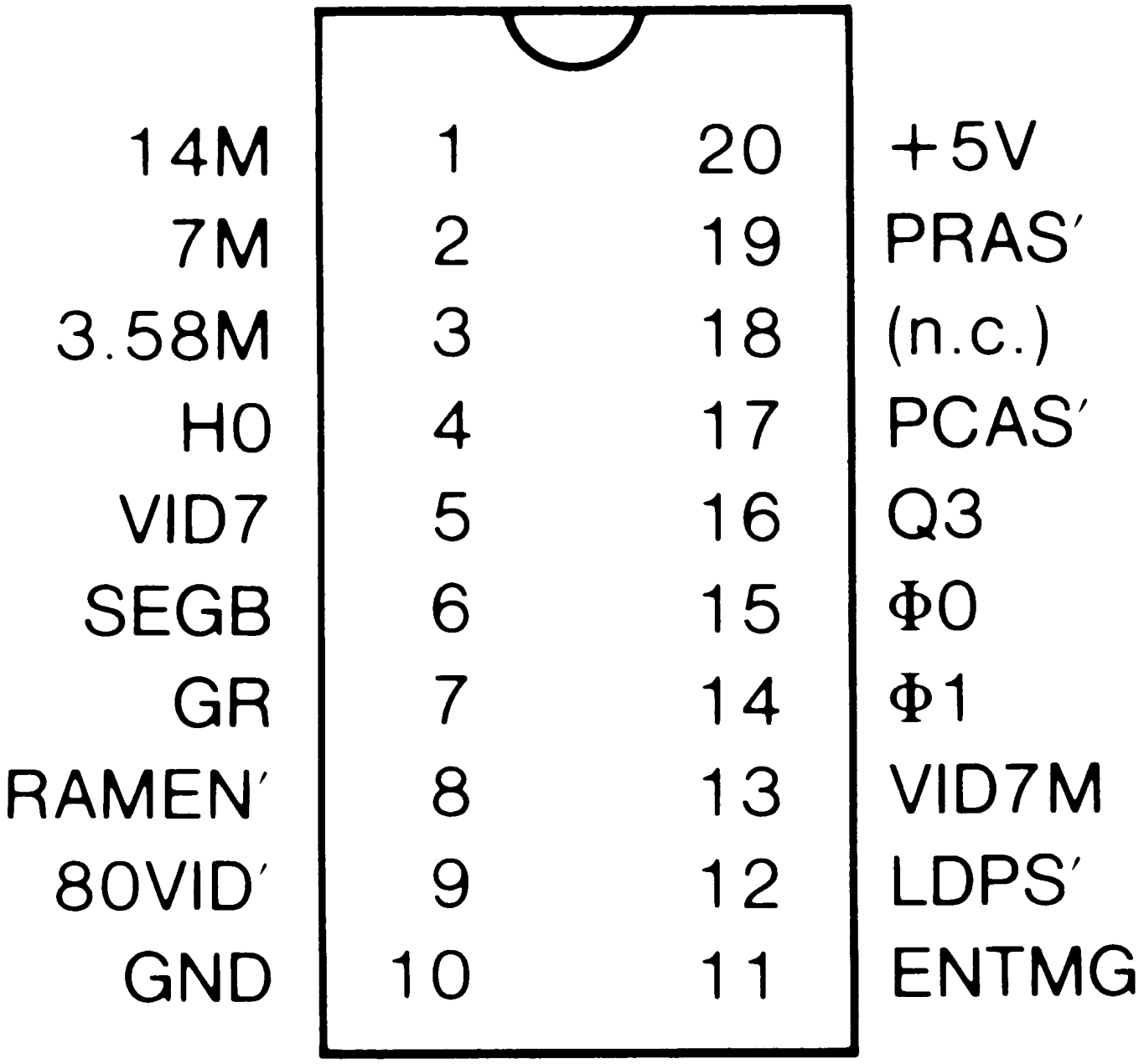


Figure 7-4 The PAL Pinouts



"A2_030-0357-B_1982_1-147a.pict" 117 KB 2002-10-27 dpi: 1200h x 1200v pix: 2168h x 2212v

Memory Addressing

The 6502 microprocessor can address 65,536 locations. The Apple IIe uses this entire address space, and then some: some areas in memory are used for more than one function. The following sections describe the memory devices used in the Apple IIe and the way they are addressed. Input and output also use portions of the memory address space; refer to the section "Peripheral-card Memory Spaces" in Chapter 6 for information.

ROM Addressing

In the Apple IIe, the following programs are permanently stored in two type 2364 8K by 8-bit ROMs (read-only memory):

- Applesoft editor and interpreter
- Monitor
- 80-column display firmware
- Self-test routines

These two ROMs are enabled by two signals called ROMEN1 and ROMEN2. The ROM enabled by ROMEN1, sometimes called the Diagnostics ROM, occupies the memory address space from $C100 to $DFFF. The address space from $C300 to $C3FF and from $C800 to $CFFF contains the 80-column display firmware. Those address spaces are normally assigned to ROM on a peripheral card in slot 3; for a discussion of the way the 80-column firmware overrides the peripheral card, see the section "Other Uses of I/O Memory Space" in Chapter 6.

Two other portions of the Diagnostics ROM, addressed from $C100 to $C2FF and from $C400 to $C7FF, contain the built-in self-test routines. These address spaces are normally assigned to the peripheral cards; when the self-test programs are running, the peripheral cards are disabled.

The remainder of the Diagnostics ROM, addressed from $D000 to $DFFF, contains part of the Applesoft BASIC interpreter.

The ROM enabled by ROMEN2, sometimes called the Monitor ROM, occupies the memory address space from $E000 to $FFFF. This ROM contains the rest of the Applesoft interpreter, in the address space from $E000 to $EFFF, and the Monitor subroutines, from $F000 to $FFFF.

Figure 7-5 The 2364 ROM Pinouts

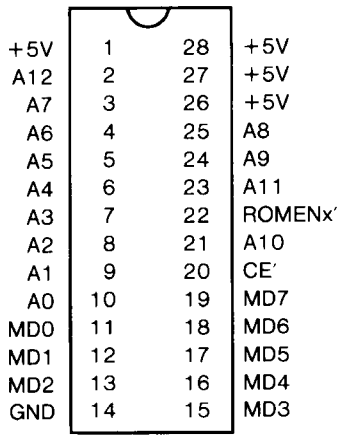


Figure 7-6 The 2316 ROM Pinouts

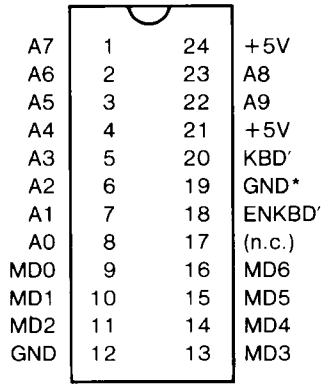
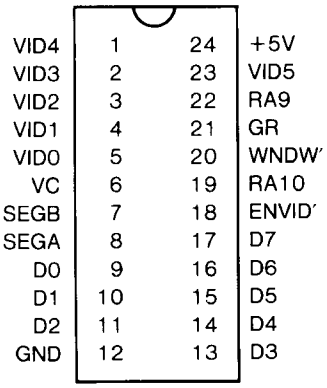


Figure 7-7 The 2333 ROM Pinouts



The other ROMs in the Apple IIe are a type 2316 ROM used for the keyboard character decoder and a type 2333 ROM used for character sets for the video display. This 2333 ROM is rather large because it includes a section of straight-through bit-mapping for the graphics modes. This way, graphics display video can pass through the same circuits as text without additional switching circuitry.

RAM Addressing

The RAM (programmable) memory in the Apple IIe is used both for program and data storage and for the video display. The areas in RAM that are used for the display are accessed both by the 6502 microprocessor and by the video display circuits. In some computers, this dual access results in addressing conflicts (cycle stealing) that can cause temporary dropouts in the video display. This problem does not occur in the Apple IIe, thanks to the way the microprocessor and the video circuits share the memory.

The memory circuits in the Apple IIe take advantage of the two-phase system clock described in the section "System Timing" to interleave the microprocessor memory accesses and the display memory accesses so that they never interfere with each other. The microprocessor reads or writes to RAM only during $\phi 0$, and the display circuits read data only during $\phi 1$.

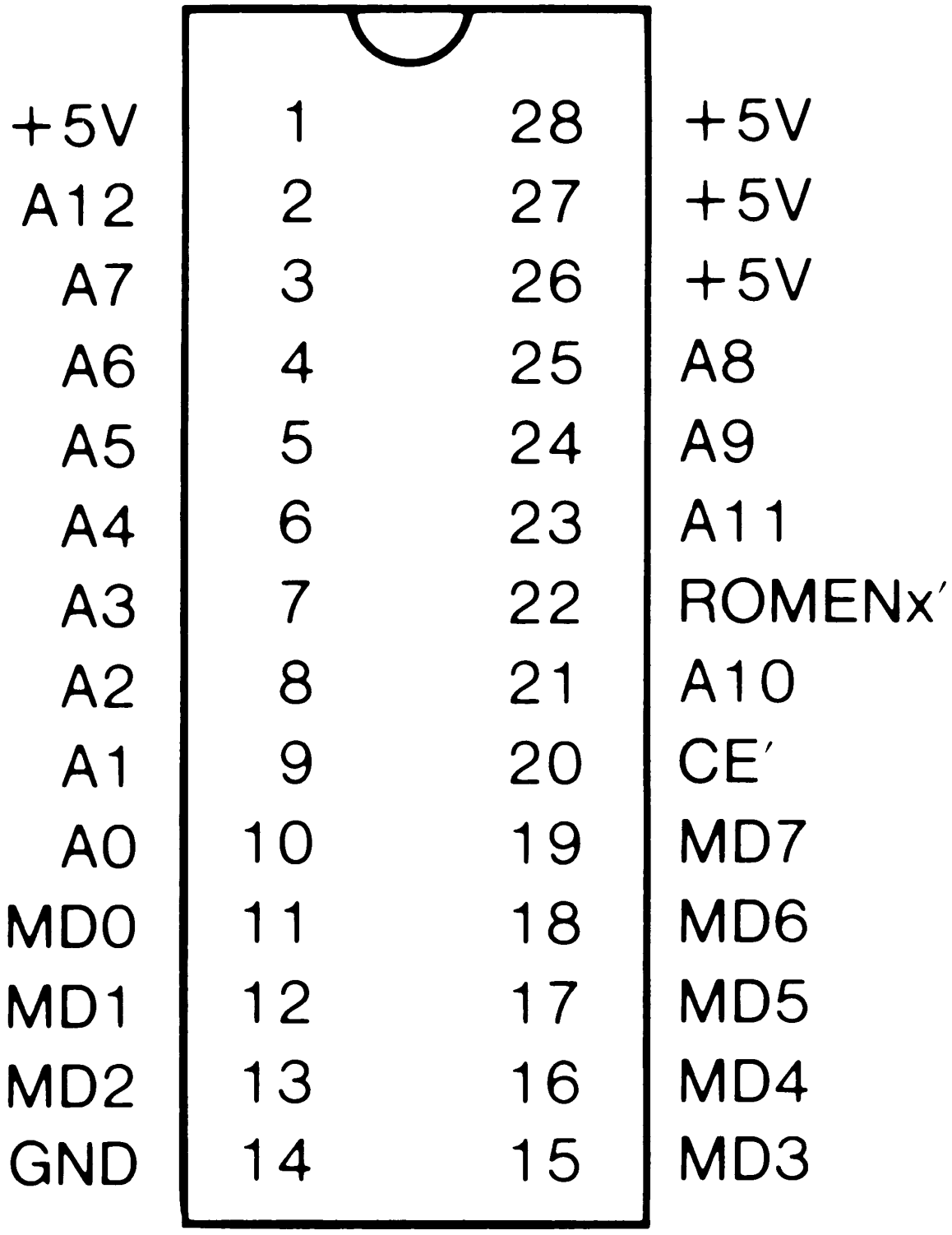
Dynamic-RAM Refreshment

The image on a video display is not permanent; it fades rapidly and must be refreshed periodically. To refresh the video display, the Apple IIe reads the data in the active display page and sends it to the display. To prevent visible flicker in the display, and to conform to standard practice for broadcast video, the Apple IIe refreshes the display sixty times per second.

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

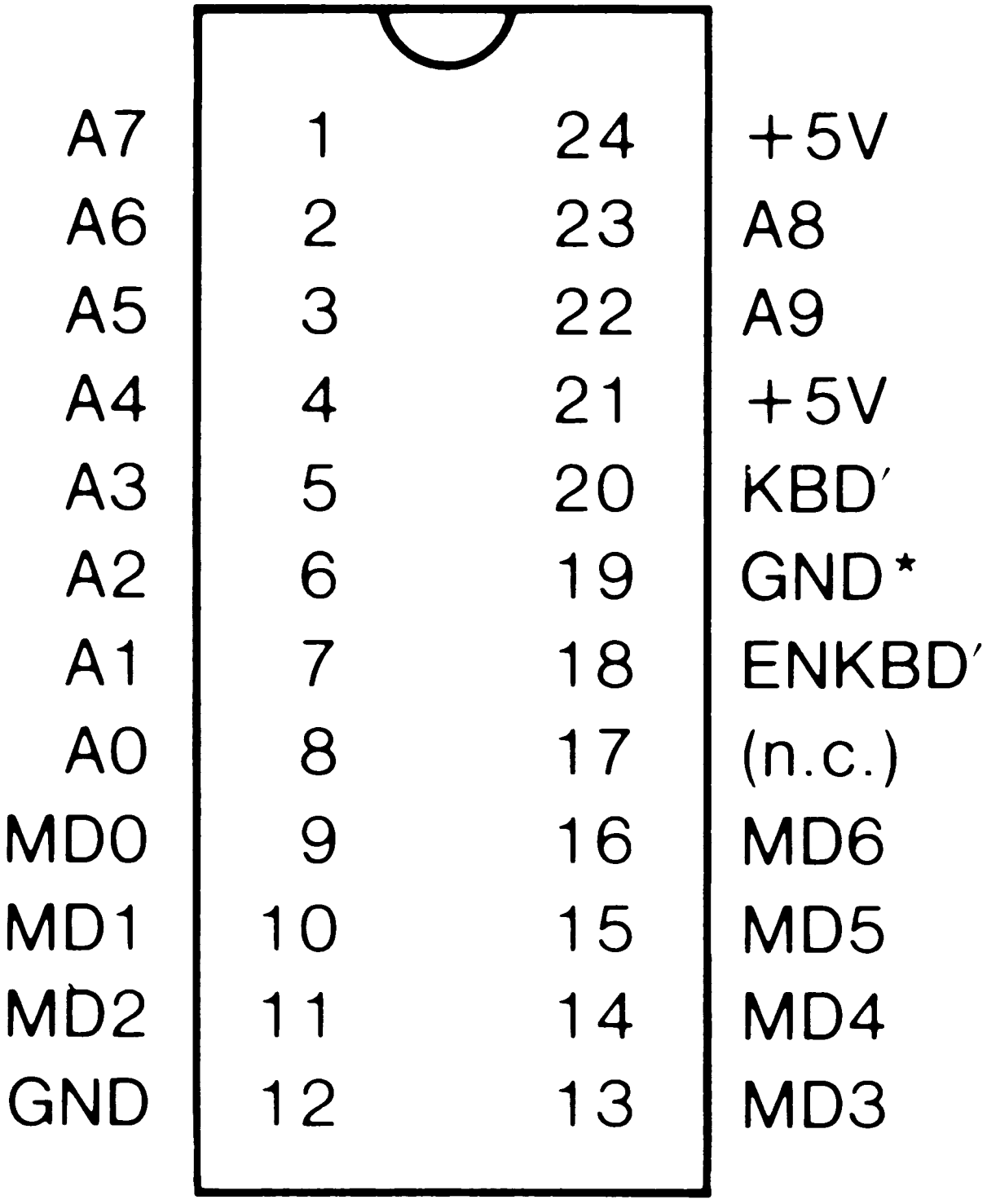
The job of refreshing the dynamic RAM devices is minimized by the structure of the devices themselves. The individual data cells in each RAM device are arranged in a rectangular array of rows

Figure 7-5 The 2364 ROM Pinouts



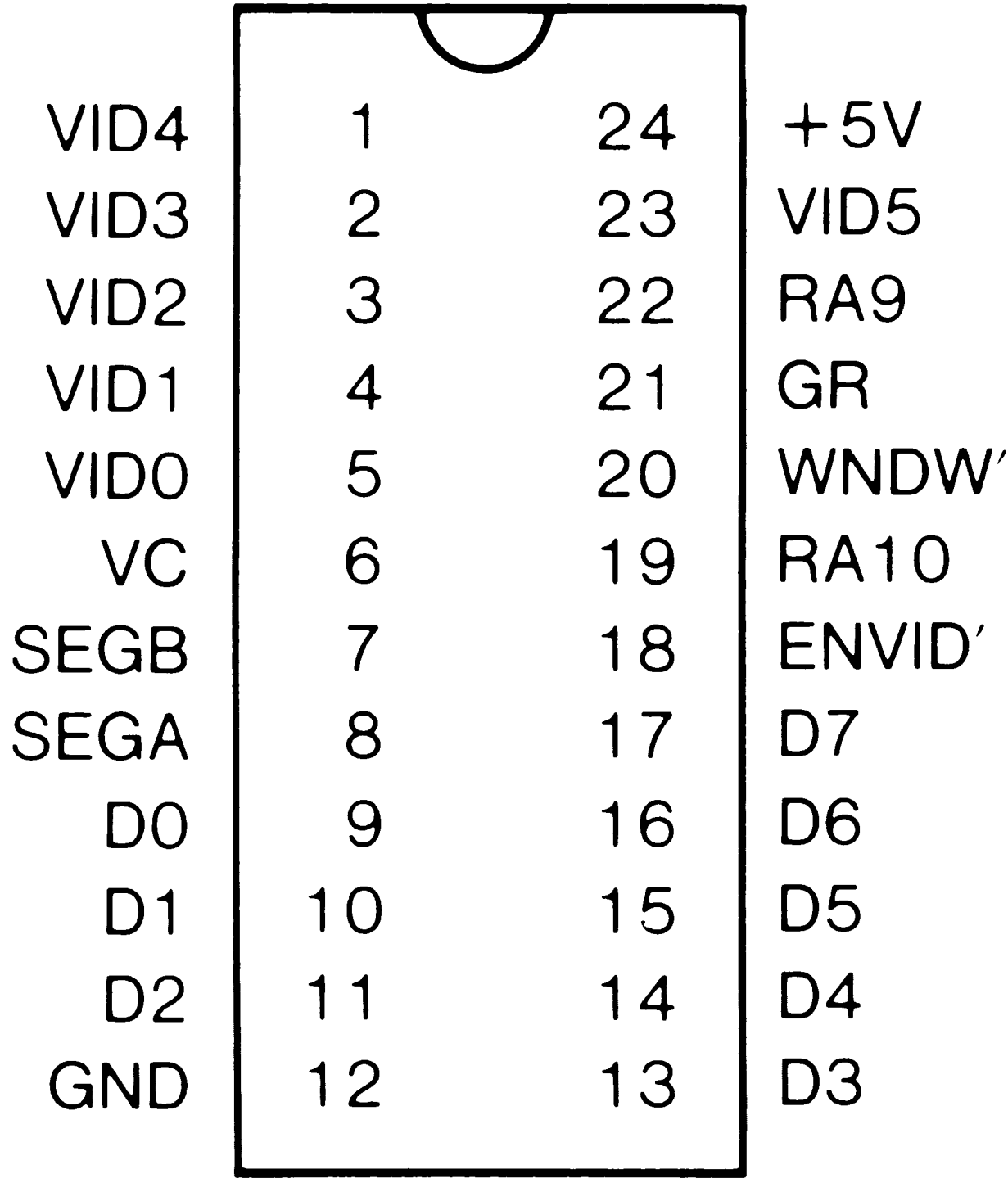
"A2_030-0357-B_1982_1-149a.pict" 149 KB 2002-10-27 dpi: 1200h x 1200v pix: 2319h x 3047v

Figure 7-6 The 2316 ROM Pinouts



"A2_030-0357-B_1982_1-149b.pict" 133 KB 2002-10-27 dpi: 1200h x 1200v pix: 2191h x 2744v

Figure 7-7 The 2333 ROM Pinouts



"A2_030-0357-B_1982_1-149c.pict" 134 KB 2002-10-27 dpi: 1200h x 1200v pix: 2239h x 2716v

and columns. When the device is addressed, the part of the address that specifies a row is presented first, followed by the address of the column. Splitting information into parts that follow each other in time is called *multiplexing*. Since only half of the address is needed at one time, multiplexing the address reduces the number of pins needed for connecting the RAMs.

Different manufacturers' 64K RAMs have cell arrays of either 128 rows by 512 columns or 256 rows by 256 columns. Only the row portion of the address is used in refreshing the RAMs.

Now consider how the display is refreshed. As described later in this chapter in the section "The Video Counters", the display circuitry generates a sequence of 8,192 memory addresses in high-resolution mode; in text and low-resolution modes, this sequence is the 1,024 display-page addresses repeated eight times. The display address cycles through this sequence 60 times a second, or once every 17 milliseconds. The way the low-order address lines are assigned to the RAMs, the row address cycles through all 256 possible values once every half-millisecond (see Table 7-9). This more than satisfies the refresh requirements of the dynamic RAMs.

Table 7-9 RAM Address Multiplexing

Mux'd Address	Row Address	Column Address
RA0	A0	A9
RA1	A1	A6
RA2	A2	A10
RA3	A3	A11
RA4	A4	A12
RA5	A5	A13
RA6	A7	A14
RA7	A8	A15

Figure 7-8 The 64K RAM Pinouts

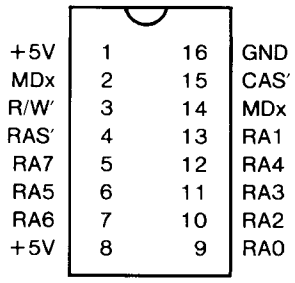
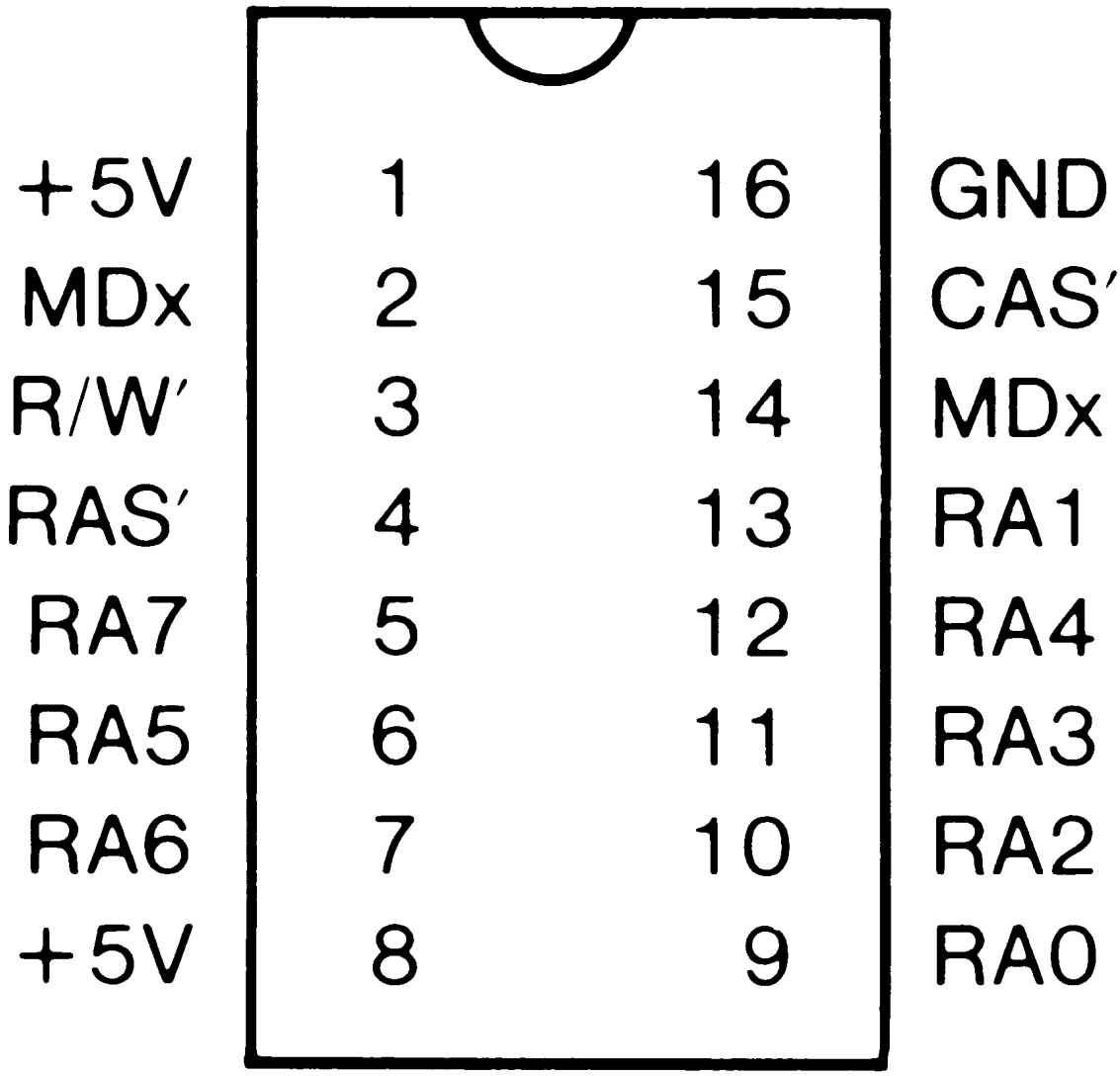


Figure 7-8 The 64K RAM Pinouts



Dynamic-RAM Timing

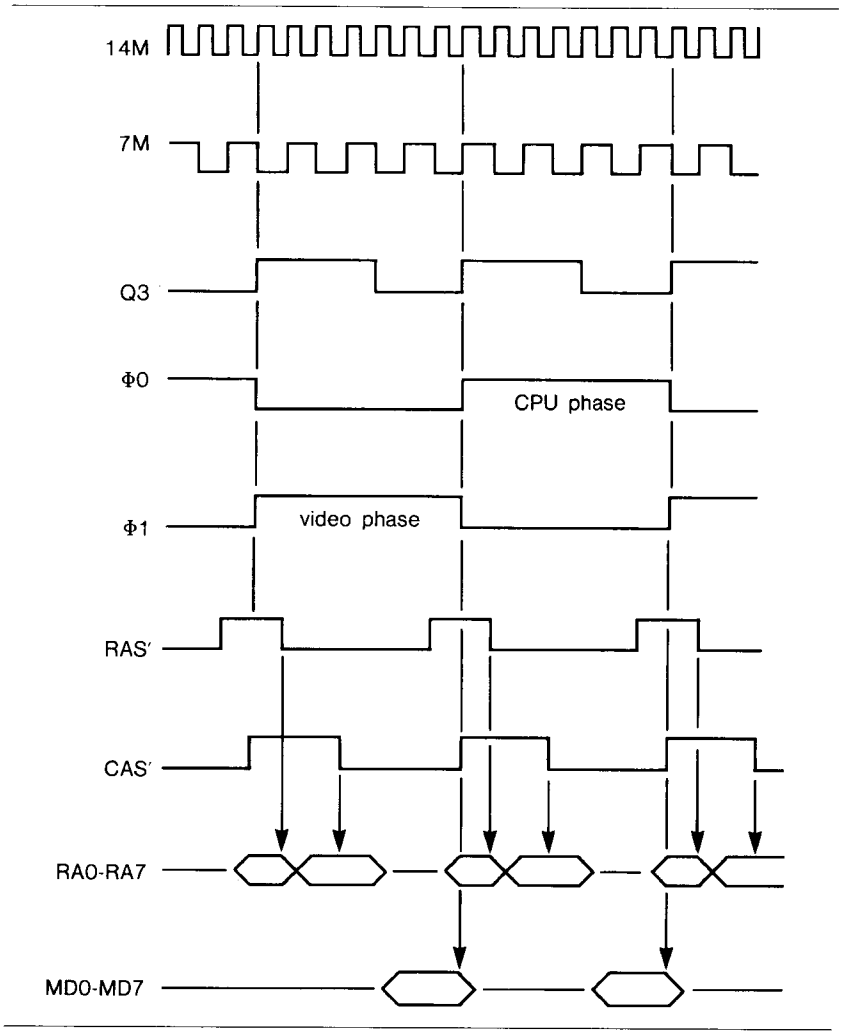
The Apple IIe's microprocessor clock runs at a moderate speed, about 1.023 MHz, but the interleaving of CPU and display cycles means that the RAM is being accessed at a 2 MHz rate, or a cycle time of just under 500 nanoseconds. Data for the CPU is strobed by the falling edge of ϕo, and display data is strobed by the falling edge of ϕ1, as shown in Figure 7-9.

The RAM timing looks complicated because the RAM address is multiplexed, as described in the previous section. The MMU takes care of multiplexing the address for the CPU cycle, and the IOU performs the same function for the display cycle. The multiplexed address is sent to the RAM ICs over the lines labelled RA0-RA7. Along with the other timing signals, the PAL generates two signals that control the RAM addressing: Row-address Strobe (RAS) and Column-address Strobe (CAS).

Table 7-10 Dynamic RAM Timing Signals

Signal Name	Description
ϕ0	Clock phase 0 (CPU phase)
ϕ1	Clock phase 1 (display phase)
RAS	Row-address strobe
CAS	Column-address strobe
Q3	Alternative column-address strobe
RA0-RA7	Multiplexed address bus
MD0-MD7	Internal data bus

Figure 7-9 RAM Timing Signals



The Video Display

The Apple IIe produces a video signal that creates a display on a standard video monitor or, if you add an RF modulator, on a black-and-white or color television set. The video signal is a composite made up of the data that is being displayed plus the horizontal and vertical synchronization signals that the video monitor uses to arrange the lines of display data on the screen.

Apple IIe's manufactured for sale in the U.S. generate a video signal that is compatible with the standards set by the NTSC (National Television Standards Committee). Apple IIe's manufactured for sale in European countries generate video that is compatible with the standard used there, which is called PAL (for Phase Alternating Lines). This manual describes only the NTSC version of the video circuits.

The display portion of the video signal is a time-varying voltage generated from a stream of data bits, where a one corresponds to a voltage that generates a bright dot, and a zero to a dark dot. The display bit stream is generated in bursts that correspond to the horizontal lines of dots on the video screen. The signal named WNDW' is low during these bursts.

During the time intervals between bursts of data, nothing is displayed on the screen. During these intervals, called the *blanking intervals*, the display is blank and the WNDW' signal is high. The synchronization signals, called sync for short, are produced by making the signal named SYNC' low during portions of the blanking intervals. The sync pulses are at a voltage equivalent to blacker-than-black video and don't show on the screen.

The Video Counters

The address and timing signals that control the generation of the video display are all derived from a chain of counters inside the IOU. Only a few of these counter signals are accessible from outside the IOU, but they are all important in understanding the operation of the display generation process, particularly the display memory addressing described in the next section.

The horizontal counter is made up of seven stages: H0, H1, H2, H3, H4, H5, and HPE'. The input to the horizontal counter is the 1 MHz signal that controls the reading of data being displayed. The complete cycle of the horizontal counter consists of 65 states. The six bits H0 through H5 count normally from 0 to 63, then start over at 0. Whenever this happens, HPE' forces another count with H0 through H5 held at zero, thus extending the total count to 65.

The IOU uses the forty horizontal count values from 25 through 64 in generating the low-order part of the display data address, as described below in the section "Display Address Mapping". The IOU uses the count values from 0 to 24 to generate the horizontal blanking, the horizontal sync pulse, and the color-burst gate.

When the horizontal count gets to 65, it signals the end of a line by triggering the vertical counter. The vertical counter has nine stages: VA, VB, VC, V0, V1, V2, V3, V4, and V5. When the vertical count reaches 262, the IOU resets it and starts counting again from zero. Only the first 192 scanning lines are actually displayed; the IOU uses the vertical counts from 192 to 261 to

generate the vertical blanking and sync pulse. Nothing is displayed during the vertical blanking interval. (The vertical line count is 262 rather than the standard 262.5 because, unlike normal television, the Apple IIe's video display is not interlaced.)

Animation displays sometimes have an erratic flicker caused by changing the display data at the same time it is being displayed. You can avoid this on the Apple IIe by reading the vertical-blanking signal (VBL) at location $C019 and only changing display data while VBL is low (data value less than 128).

Display Memory Addressing

As described in Chapter 2 in the section "Addressing Display Pages Directly", data bytes are not stored in memory in the same sequence in which they appear on the display. You can get an idea of the way the display data is stored by using the Monitor to set the display to graphics mode, then storing data starting at the beginning of the display page at hexadecimal $400 and watching the effect on the display. If you do this, you should use the graphics display instead of text to avoid confusion: the text display is also used for Monitor input and output.

If you want your program to display data by storing it directly into the display memory, you must first transform the display coordinates into the appropriate memory addresses, as shown in Chapter 2. The descriptions that follow will help you understand how this address transformation is done and why it is necessary. They will not (alas!) eliminate that necessity.

The address transformation that folds three rows of forty display bytes into 128 contiguous memory locations is the same for all display modes, so it is described first. The differences among the different display modes are described in the section "Video Display Modes", below.

Display Address Mapping

Consider the simplest display on the Apple IIe, the 40-column text mode. To address forty columns requires six bits, and to address twenty-four rows requires another five bits, for a total of eleven address bits. Addressing the display this way would involve 2048 (two to the eleventh power) bytes of memory to display a mere 960 characters. The 80-column text mode would require 4096 bytes to display 1920 characters. The leftover

chunks of memory that were not displayed could be used for storing other data, but not easily, because they would not be contiguous.

Instead of using the horizontal and vertical counts to address memory directly, the circuitry inside the IOU transforms them into the new address signals described below. The transformed display address must meet the following criteria:

- Map the 960 bytes of 40-column text into only 1024 bytes.
- Scan the low-order address to refresh the dynamic RAMs.
- Continue to refresh the RAMs during video blanking.

The requirements for RAM refreshing are discussed above, in the section "Dynamic-RAM Refreshment".

The transformation involves only horizontal counts H3, H4, and H5, and vertical counts V3 and V4. Vertical count bits VA, VB, and VC address the lines making up the characters, and are not involved in the address transformation. The remaining low-order count bits, H0, H1, H2, V0, V1, and V2 are used directly, and are not involved in the transformation.

The IOU performs an addition that reduces the five significant count bits to four new signals called S0, S1, S2, and S3, where S stands for sum. Figure 7-10 is a diagram showing the addition in binary form, with V3 appearing as the carry in and H5 appearing as its complement H5′. A constant value of one appears as the low-order bit of the addend. The carry bit generated with the sum is not used.

Figure 7-10 Display Address Transformation

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

If this transformation seems terribly obscure, try it with actual values. For example, for the upper-left corner of the display, the vertical count is zero and the horizontal count is 24: H0, H1, H2, and H5 are zeros and H3 and H4 are ones. The value of the sum is zero, so the memory location for the first character on the display is the first location in the display page, as you might expect.

Horizontal bits H0, H1, and H2 and sum bits S0, S1, and S2 make up the transformed horizontal address (A0 through A6 in Table 7-11). As the horizontal count increases from 24 to 63, the value of the sum (S3 S2 S1 S0) increases from zero to four and the transformed address goes from 0 to 39, relative to the beginning of the display page.

The low-order three bits of the vertical row counter are V0, V1, and V2. These bits control address bits A7, A8, and A9, as shown in Table 7-11, so that rows 0 through 7 start on 128-byte boundaries. When the vertical row counter reaches 8, V0, V1, and V2 are zero again, and V3 changes to one. If you do the addition in Figure 7-10 with H equal to 24 (the horizontal count for the first column displayed) and V equal to 8, the sum is 5 and the horizontal address is 40: the first character in row 8 is stored in the memory location 40 bytes from the beginning of the display page.

Figure 7-11 40-column Text Display Memory. Memory locations marked with an asterisk (*) are reserved for use by peripheral I/O firmware: refer to the section "Peripheral-card RAM Space", in Chapter 6.

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

Figure 7-11 shows how groups of three forty-character rows are stored in blocks of 120 contiguous bytes starting on 128-byte address boundaries. This diagram is another way of describing the display mapping shown in Figure 2-5. Notice that the three rows in each block of 120 bytes are not adjacent on the display.

Table 7-11 Display Memory Addressing

*For these address bits, see text and Table 7-12.

Memory Address Bit	Display Address Bit
A0	H0
A1	H1
A2	H2
A3	S0
A4	S1
A5	S2
A6	S3
A7	V0
A8	V1
A9	V2
A10	*
A11	*
A12	*
A13	*
A14	*
A15	GND

Table 7-11 shows how the signals from the video counters are assigned to the address lines. H0, H1, and H2 are horizontal-count bits, and V0, V1, and V2 are vertical-count bits. S0, S1, S2 and S3 are the folded address bits described above. Address bits marked with asterisks (*) are different for different modes: see Table 7-12 and the next three sections.

Table 7-12 Memory Address Bits for Display Modes

Address Bit	Display Mode: Text and Lo-Res	Hi-Res
A10	80VID+PG2'	VA
A11	80VID'·PG2	VB
A12	0	VC
A13	0	80VID+PG2'
A14	0	80VID'·PG2

Video Display Modes

The different display modes all use the address-mapping scheme described in the previous section, but they use different-sized memory areas in different locations. The next three sections describe the addressing schemes and the methods of generating the actual video signals for the different display modes.

Text Displays

The text and low-resolution graphics pages begin at memory locations $400 and $800. Table 7-12 shows how the display-mode signals control the address bits to produce these addresses. Address bits A10 and A11 are controlled by PG2 and 80VID, which are set by the display-page and 80-column-video soft switches. Address bits A12, A13, and A14 are set to zero. Notice that 80VID active inhibits PG2: there is only one display page in 80-column mode.

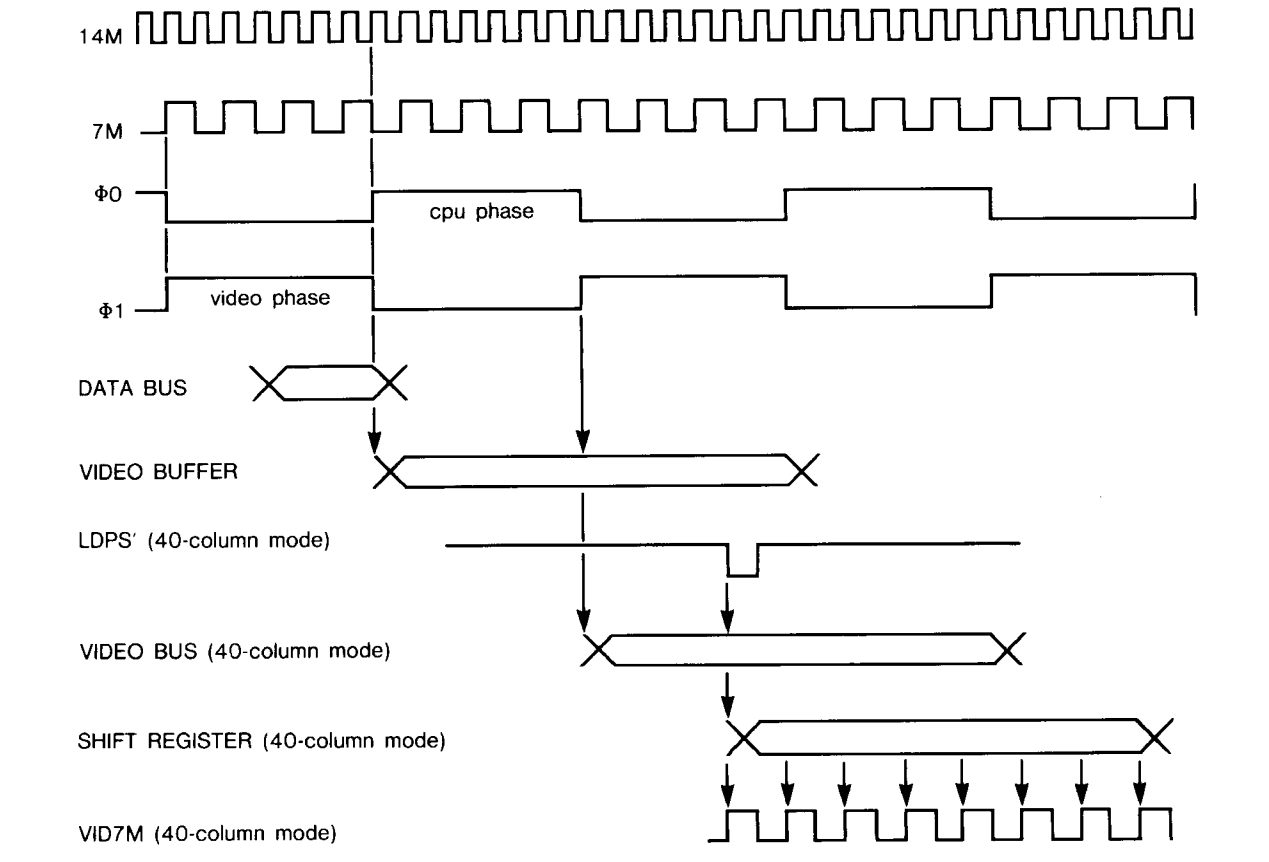
The low-order six bits of each data byte reach the character generator directly, via the video data bus VID0-VID5. The two high-order bits are modified by the IOU to select between the primary and alternate character sets and are sent to the character generator on lines RA9 and RA10.

The data for each row of characters are read eight times, once for each of the eight lines of dots making up the row of characters. The data bits are sent to the character generator along with VA, VB, and VC, the low-order bits from the vertical counter. For each character being displayed, the character generator puts out one of eight stored bit patterns selected by the three-bit number made up of VA, VB, and VC.

The bit patterns from the character generator are loaded into the 74166 parallel-to-serial shift register and output as a serial bit stream that goes to the video output circuit. The shift register is controlled by signals named LDPS' (for load parallel-to-serial shifter) and VID7M (for video 7 Mhz). In 40-column mode, LDPS' strobes the output of the character generator into the shift register once each microsecond, and VID7M shifts the bits out at 7 MHz.

The addressing for the 80-column display is exactly the same as for the 40-column display: the 40 columns of display memory on the 80-column card are addressed in parallel with the 40 columns in main memory. The data from these two memories reach the video data bus (lines VID0-VID7) via separate 74LS374 three-state buffers. These buffers are loaded simultaneously, but their outputs are sent to the character generator alternately by ϕo and ϕ1. In 80-column mode, LDPS' loads data from the character generator into the shift register twice during each microsecond, once during ϕo and once during ϕ1, and VID7M runs at 14 MHz to shift the data bits out twice as fast.

Figure 7-12 Video Timing Signals



Low-Resolution Display

In the graphics modes, VA and VB are not used by the character generator, so the IOU uses lines SEGA and SEGB to transmit H0 and HIRES', as shown in Table 7-13.

Table 7-13 Character-generator Control Signals

Display mode	SEGA	SEGB	SEGC
Text	VA	VB	VC
Graphics	H0	HIRES'	VC

The low-resolution graphics display uses VC to divide the eight display lines corresponding to a row of characters into two groups of four lines each. Each row of data bytes is addressed eight times, the same as in text mode, but each byte is interpreted as two nybbles. Each nybble selects one of sixteen colors. During the upper four of the eight display lines, VC is low and the low-order nybble determines the color. During the lower four display lines, VC is high and the high-order nybble determines the color.

The bit patterns that produce the low-resolution colors are read from the character-generator ROM in the same way the bit patterns for characters are produced in text mode. The 74166 parallel-to-serial shift register converts the bit patterns to a serial bit stream for the video circuits.

The video signal generated by the Apple IIe includes a short burst of 3.58 MHz signal that is used by an NTSC color monitor or color TV set to generate a reference 3.58 MHz color signal. The Apple IIe's video signal produces color by interacting with this 3.58 MHz signal inside the monitor or TV set. Different bit patterns produce different colors by changing the duty cycles and delays of the bit stream relative to the 3.58 MHz color signal. To produce the small delays required for so many different colors, the shift register runs at 14 MHz and shifts out 14 bits during each cycle of the 1-MHz data clock. To generate a stream of fourteen bits from each eight-bit pattern read from the ROM, the output of the shift register is connected back to the register's serial input to repeat the same eight bits; the last two bits are ignored the second time around.

Each bit pattern is output for the same amount of time as a character: 1.02 microseconds. Because that is exactly enough time for three and a half cycles of the 3.58 MHz color signal, the phase relationship between the bit patterns and the signal changes by a half cycle for each successive pattern. To compensate for this, the character generator puts out one of two different bit patterns for each nybble, depending on the state of H0, the low-order bit of the horizontal counter.

High-Resolution Display

The high-resolution graphics pages begin at memory locations $2000 and $4000 (decimal 8192 and 16384). These page addresses are selected by address bits A13 and A14. In high-resolution mode, these address bits are controlled by PG2 and 80VID, the signals controlled by the display-page (PAGE2) and 80-column-video (80COL) soft switches. As in text mode, 80VID inhibits addressing of the second page because there is only one page of 80-column text available for mixed mode.

In high-resolution graphics mode, the display data are still stored in blocks like the one shown in Figure 7-11, but there are eight of these blocks. As Table 7-11 and Table 7-12 show, vertical counts VA, VB, and VC are used for address bits A10, A11, and A12, which address eight blocks of 1024 bytes each. Remember that in the display VA, VB, and VC count adjacent horizontal lines in groups of eight. This addressing scheme maps each of those lines into a different 1024-byte block. It might help to think of it as a kind of eight-way multiplexer: it's as if eight text displays were combined to produce a single high-resolution display, with each text display providing one line of dots in turn, instead of a row of characters.

The high-resolution bit patterns are produced by the character-generator ROM. In this mode, the bit patterns simply reproduce the eight bits of display data. The low-order six bits of data reach the ROM via the video data bus VID0-VID5. The IOU sends the other two data bits to the ROM via RA9 and RA10.

The high-resolution colors described in Chapter 2 are produced by the interaction between the video signal the bit patterns generate and the 3.58 MHz color signal generated inside the monitor or TV set. The high-resolution bit patterns are always shifted out at 7 MHz, so each dot corresponds to a half-cycle of the 3.58 MHz color signal. Any part of the video signal that

produces a single white dot between two black dots, or vice-versa, is effectively a short burst of 3.58 MHz and is therefore displayed as color. In other words, a bit pattern consisting of alternating ones and zeros gets displayed as a line of color. The high-resolution graphics subroutines produce the appropriate bit patterns by masking the data bits with alternating ones and zeros.

To produce different colors, the bit patterns must have different phase relationships to the 3.58 MHz color signal. If alternating ones and zeros produce a certain color, say green, then reversing the pattern to zeros and ones will produce the complementary color, purple. As in the low-resolution mode, each bit pattern corresponds to three and a half cycles of the color signal, so the phase relationship between the data bits and the color signal changes by a half cycle for each successive byte of data. Here, however, the bit patterns produced by the hardware are the same for adjacent bytes; the color compensation is performed by the high-resolution software, which uses different color masks for data being displayed in even and odd columns.

To produce other colors, bit patterns must have other timing relationships to the 3.58 MHz color signal. In high-resolution mode, the Apple IIe produces two more colors by delaying the output of the shift register by half a dot (70 ns), depending on the high-order bit of the data byte being displayed. (The high-order bit doesn't actually get displayed as a dot, because at 7 MHz there is only time to shift out seven of the eight bits.)

As each byte of data is sent from the character generator to the shift register, high-order data bit D7 is also sent to the PAL. If D7 is off, the PAL transmits shift-register timing signals LDPS' and VID7M normally. If D7 is on, the PAL delays LDPS' and VID7M by 70 nanoseconds, the time corresponding to half a dot. The bit pattern that formerly produced green now produces orange; the pattern for purple now produces blue.

A note about timing: For 80-column text, the shift register is clocked at twice normal speed. When 80-column text is used with graphics in mixed mode, the PAL controls shift-register timing signals LDPS' and VID7M so that the graphics portion of the display works correctly even when the text window is in 80-column mode.

Video Output Signals

The stream of video data generated by the display circuits described above goes to a linear summing circuit built around transistor Q1 where it is mixed with the sync signals and the color burst. Resistors R3, R5, R7, R10, R13, and R15 adjust the signals to the proper amplitudes, and a tank circuit (L3 and C32) resonant at 3.58 MHz conditions the color burst.

The resulting video signal is an NTSC-compatible composite-video signal that can be displayed on a standard video monitor. The signal is similar to the EIA (Electronic Industries Association) standard positive composite video (see Table 7-14). This signal is available in two places in the Apple IIe:

- At the phono jack on the back of the Apple IIe. The sleeve of this jack is connected to ground and the tip is connected to the video output through a resistor network that attenuates it to about 1 volt and matches its impedance to 75 ohms.
- At the internal video connector on the Apple IIe circuit board near the RCA jack, J13 in Figure 7-14c. It is made up of four Molex-type pins, 0.25 inches tall, on 0.10 inch centers. This connector carries the video signal, ground, and two power supplies, as shown in Table 7-14.

Table 7-14 Internal Video Connector Signals

Pin Number	Name	Description
1	GROUND	System common ground
2	VIDEO	NTSC-compatible positive composite video. White level is about 2.0 volts, black level is about 0.75 volts, and sync level is 0.0 volts. This output is not protected against short circuits.
3	−5V	−5 volt power supply
4	+12V	+12 volt power supply

Built-in I/O Circuits

The use of the Apple IIe's built-in I/O features is described in Chapter 2. This section describes the hardware implementation of all of those features except the video display described in the previous sections. The IOU (Input/Output Unit) generates the output signals for the speaker, the cassette interface, and the annunciators directly. The other I/O features are handled by smaller ICs, as described below.

The addresses of the built-in I/O features are described in Chapter 2 and listed in Table 2-2, Table 2-11, and Table 2-12. All of the built-in I/O features except the displays use memory locations between $C000 and $C070 (decimal 49152 and 49264). The I/O address decoding is performed by three ICs: a 74LS138, a 74LS154, and a 74LS251.

The 74LS138 decodes address lines A8, A9, A10, and A11 to select address pages on 256-byte boundaries starting at $C000 (decimal 49152). When it detects addresses between $C000 and $C0FF, it enables the IOU and the 74LS154. The 74LS154 in turn decodes address lines A4, A5, A6, and A7 to select 16-byte address areas between $C000 and $C0FF. Addresses between $C060 and $C06F enable the 74LS251 that multiplexes the hand control switches and paddles; addresses between $C070 and $C07F reset the NE558 quadruple timer that interfaces to the hand controls, as described below in the section "Game I/O Signals".

The Keyboard

The Apple IIe's keyboard is a matrix of keyswitches connected to an AY-3600-type keyboard decoder via a ribbon cable and a 26-pin connector. The AY-3600 scans the array of keys over and over to detect any keys pressed. The scanning rate is set by the external resistor-capacitor network made up of C70 and R32. The debounce time is also set externally, by C71.

The AY-3600's outputs include five bits of key code plus separate lines for [CONTROL], [SHIFT], any-key-down, and keyboard strobe. The any-key-down and keyboard-strobe lines are connected to the IOU, which addresses them as soft switches. The key-code lines, along with [CONTROL] and [SHIFT], are inputs to a separate 2316 ROM. The ROM translates them to the character codes that are enabled onto the data bus by signals named KBD' and ENKBD'. The KBD' signal is enabled by the MMU whenever a program reads location $C000, as described in Chapter 2.

Table 7-15 Keyboard Connector Signals

Pin Number	Name	Description
1, 2, 4, 6, 8, 10, 23, 25, 12, 22	Y0-Y9	Y-direction key-matrix connections
3	+5	+5 volt supply
5, 7, 9, 15	n. c.	
1	LCNTL'	Line from CONTROL key
13	GND	System common ground
14, 16, 20, 21, 19, 26, 17	X0-X7	X-direction key-matrix connections
24	LSHFT'	Line from SHIFT key

Connecting a Keypad

There is a smaller connector wired in parallel with the keyboard connector. You can connect a ten-key numeric pad to the Apple IIe via this connector.

Table 7-16 Keypad Connector Signals

Pin Number	Name	Description
1, 2, 5, 3, 4, 6	Y0-Y5	Y-direction key-matrix connections
7	n. c.	
9, 11, 10, 8	X4-X7	X-direction key-matrix connections

Cassette I/O

The two miniature phone jacks on the back of the Apple IIe are used to connect an audio cassette recorder for saving programs. The output signal to the cassette recorder comes from a pin on the IOU via resistor network R6 and R9, which attenuates the signal to a level appropriate for the recorder's microphone input. Input from the recorder is amplified and conditioned by a type 741 operational amplifier and sent to one of the inputs of the 74LS251 input multiplexer.

The signal specifications for cassette I/O are:

Input: 1 volt (nominal) from recorder Earphone or Monitor output. Input impedance is 12K ohms.

Output: 25 millivolts to recorder Microphone input. Output impedance is 100 ohms.

The Speaker

The Apple IIe's built-in loudspeaker is controlled by a single bit of output from the IOU (Input Output Unit). The signal from the IOU is AC coupled to Q5, an MPSA13 Darlington transistor amplifier. The speaker connector is a Molex KK100 connector, J18 in Figure 7-14b, with two square pins 0.25 inches tall and on 0.10-inch centers.

A light-emitting diode is connected in parallel across the speaker pins such that, when the speaker is not connected, the diode glows whenever the speaker signal is on. This diode is used as a diagnostic indicator during assembly and testing of the Apple IIe.

Table 7-17 Speaker Connector Signals

Pin Number	Name	Description
1	SPKR	Speaker signal. This line will deliver about 0.5 watts into an 8-ohm speaker.
2	+5	+5V power supply. Note that the speaker is not connected to system ground.

Game I/O Signals

Several I/O signals that are individually controlled via soft switches are collectively referred to as the game signals. Even though they are normally used for hand controls, these signals can be used for other simple I/O applications. There are five output signals: the four annunciators, numbered `A0` through `A3`, and one strobe output. There are three one-bit inputs, called switches and numbered `SW0` through `SW2`, and four analog inputs, called paddles and numbered `PDL0` through `PDL3`.

The annunciator outputs are driven directly by the IOU (Input Output Unit). These outputs can drive one TTL (transitor-transitor logic) load each; for heavier loads, you must use a transistor or a TTL buffer on these outputs. These signals are only available on the 16-pin internal connector (see Table 7-18).

The strobe output is a pulse transmitted any time a program reads or writes to location `$C040`. The strobe pin is connected to one output of the 74LS154 address decoder. This TTL signal is normally high; it goes low during $\phi 0$ of the instruction cycle that addresses location `$C040`. This signal is only available on the 16-pin internal connector (see Table 7-18).

The game inputs are multiplexed along with the cassette input signal by a 74LS251 eight-input multiplexer enabled by the `C06X'` signal from the 74LS154 I/O address decoder. Depending on the low-order address, the appropriate game input is connected to bit 7 of the data bus.

The switch inputs are standard low-power Shottky TTL inputs. To use them, connect each one to 220-ohm pull-up resistors connected to the +5-volt supply and through single-pole, momentary-contact pushbutton switches to ground.

The hand-control inputs are connected to the timing inputs of an NE558 quadruple 555-type analog timer. Addressing `$C07X` sends a signal from the 74LS154 that resets all four timers and causes their outputs to go to one (high). A variable resistance of up to 150K ohms connected between one of these inputs and the +5V supply controls the charging time of one of four 0.022-microfarad capacitors. When the voltage on the capacitor passes a certain threshold, the output of the NE558 changes back to zero (low).

Programs can determine the setting of a variable resistor by resetting the timers and then counting time until the selected timer input changes from high to low. The resulting count is proportional to the resistance.

The game I/O signals are all available on a 16-pin DIP socket labelled GAME I/O on the main circuit board inside the case. The switches and the paddles are also available on a D-type miniature connector on the back of the Apple IIe; see J8 and J15 in Figure 7-14d.

Table 7-18 Game I/O Connector Signals

Internal-Connector Pin Number	Back-panel-Connector Pin Number	Signal Name	Description
1	2	+5V	+5 power supply. Total current drain from this pin must not exceed 100mA.
2, 3, 4	7, 1, 6	PB0-PB2	Switch inputs. These are standard 74LS inputs.
5	—	STROBE'	Strobe output. This line goes low during $\phi 0$ of a read or write instruction to location $C040.
6, 10, 7, 11	5, 8, 4, 9	PDL0-PDL3	Hand control inputs. Each of these should be connected to a 150K-ohm variable resistor connected to +5V.
8	3	GND	System ground.
15, 14, 13, 12	—	AN0-AN3	Annunciators. These are standard 74LS TTL outputs and must be buffered to drive other than TTL inputs.
9, 16	—	n. c.	Nothing is connected to these pins.

Expanding the Apple IIe

The main circuit board of the Apple IIe has eight empty card connectors or slots on it. These slots make it possible to add features to the Apple IIe by plugging in peripheral cards with additional hardware. Chapter 6 describes the standards for programming peripheral cards for the Apple IIe. This section describes the hardware that supports them, including all of the signals available on the expansion slots.

The Expansion Slots

The seven connectors lined up across the back part of the Apple IIe's main circuit board are the expansion slots, also called peripheral slots or simply slots, numbered from 1 to 7. They are 50-pin PC-card edge connectors with pins on 0.10-inch centers. A PC card plugged into one of these connectors has access to all of the signals necessary to perform input and output and to execute programs in RAM or ROM on the card. These signals are described briefly in Tables 7-19a, 7-19b, and 7-19c. The following paragraphs describe the signals in general and mention a few points that are often overlooked. For further details, refer to the schematic diagram in Figures 7-14a, 7-14b, 7-14c, and 7-14d.

The Peripheral Address Bus

The 6502's address bus is buffered by two 74LS244 octal three-state buffers. These buffers, along with a buffer in the 6502's R/W' line, are enabled by a signal derived from the DMA' daisy-chain on the expansion slots. Pulling the peripheral line DMA' low disables the address and R/W' buffers so that peripheral DMA circuitry can control the address bus. The DMA address and R/W' signals supplied by a accessory card must be stable all during ϕo of the instruction cycle, as shown in Figure 7-13.

Another signal that can be used to disable normal operation of the Apple IIe is INH'. Pulling INH' low disables all of the memory in the Apple IIe except the part in the I/O space from $C000 to $CFFF. A peripheral card that uses either INH' or DMA' must observe proper timing; in order to disable RAM and ROM cleanly, the disabling signal must be stable all during ϕo of the instruction cycle (refer to the timing diagram in Figure 7-13).

The peripheral devices should use I/O SELECT' and DEVICE SELECT' as enables. Most peripheral ICs require their

enable signals to be present for a certain length of time before data is strobed into or out of the device. Remember that I/O SELECT' and DEVICE SELECT' are only asserted during ϕ0 high.

The Peripheral Data Bus

The Apple IIe has two versions of the 6502 data bus: an internal bus, MD0-MD7, connected directly to the 6502; and an external bus, D0-D7, driven by a 74LS245 octal bidirectional bus buffer. The 6502 is fabricated with MOS circuitry, so it can drive capacitive loads of up to about 130 pF. If peripheral cards are installed in all seven slots, the loading on the data bus can be as high as 500 pF, so the 74LS245 drives the data bus for the peripheral cards. The same argument applies if you use MOS devices on peripheral cards: they don't have enough drive for the fully-loaded bus, so you should add buffers.

Loading and Driving Rules

Tables 7-19a, 7-19b, and 7-19c show the drive requirements and loading limits for each pin on the expansion slots. The address bus, the data bus, and the R/W' line should be driven by three-state buffers. Remember that there is considerable distributed capacitance on these busses and that you should plan on tolerating the added load of up to six additional peripheral cards. MOS devices such as PIAs and ACIAs cannot switch such heavy capacitive loads. Connecting such devices directly to the bus will lead to possible timing and level errors.

Interrupt and DMA Daisy Chains

The interrupt requests (IRQ' and NMI') and the direct-memory access (DMA') signal are available at all seven expansion slots. A peripheral card requests an interrupt or a DMA transfer by pulling the appropriate output line low (active). If two peripheral cards request an interrupt or a DMA transfer at the same time, they will contend for the data and address busses. To prevent this, two pairs of pins on each connector are wired as a priority daisy chain. The daisy-chain pins for interrupts are INT IN and INT OUT, and the pins for DMA are DMA IN and DMA OUT, as shown for J1-J7 in Figure 7-14d.

Each daisy chain works like this: the output from each connector goes to the input of the next higher numbered one. For these signals to be useful for cards in lower numbered connectors, all of the higher numbered connectors must have cards in them, and all of those cards must connect DMA IN to DMA OUT and INT IN to INT OUT. Whenever an accessory card uses pin DMA', it must do so

only if its DMA IN line is active, and it must disable its DMA OUT line while it is using DMA'. The INT IN and INT OUT lines must be used the same way: enable the card's interrupt circuits with INT IN, and disable INT OUT whenever IRQ' or NMI' is being used.

Figure 7-13 Peripheral-signal Timing

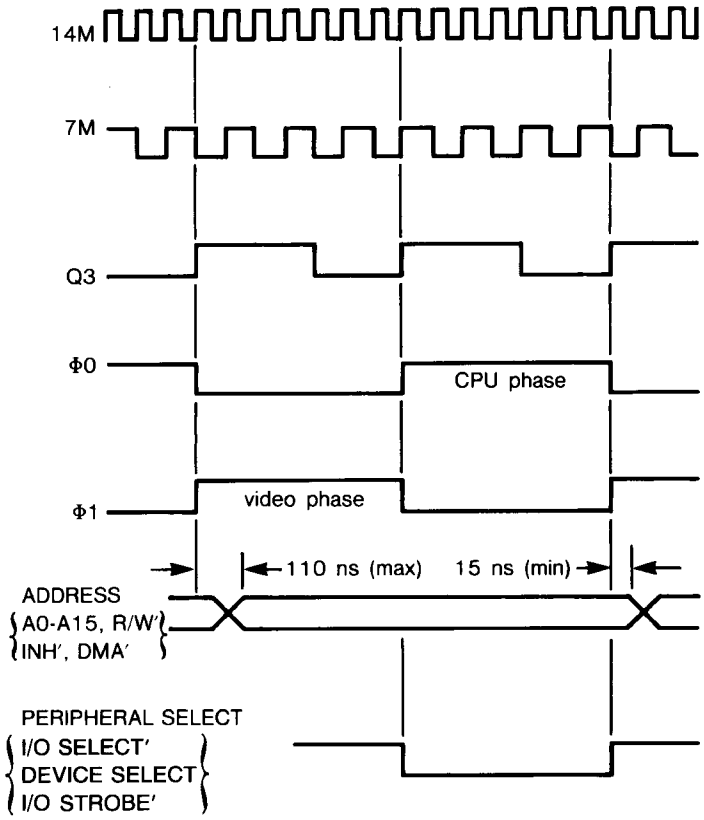


Table 7-19a Expansion Slot Signals

*Loading limits are for each card.

Pin Number	Name	Description
1	I/O SELECT	Normally high; goes low during ϕ0 when the 6502 addresses location $CnXX, where n is the connector number. This line can drive 10 LS TTL loads.*
2-17	A0-A15	Three-state address bus. The address becomes valid during ϕ1 and remains valid during ϕ0. Each address line can drive 5 LS TTL loads.*
18	R/W'	Three-state read/write line. Valid at the same time as the address bus; high during a read cycle, low during a write cycle. It can drive 2 LS TTL loads.*
19	SYNC'	Composite horizontal and vertical sync, on expansion slot 7 ONLY. This line can drive 2 LS TTL loads.*
20	I/O STROBE'	Normally high; goes low during ϕ0 when the 6502 addresses a location between $C800 and $CFFF. This line can drive 4 LS TTL loads.
21	RDY	Input to the 6502. Pulling this line low during ϕ1 halts the 6502 with the address bus holding the address of the location currently being fetched. This line has a 3300 ohm pullup resistor to +5V.
22	DMA'	Input to the address bus buffers. Pulling this line low during ϕ1 disconnects the 6502 from the address bus. This line has a 3300 ohm pullup resistor to +5V
23	INT OUT	Interrupt priority daisy-chain output. Usually connected to pin 28 (INT IN). (Note: on slot 7 ONLY, this pin can be connected to the graphics-mode signal GR: see text for details.)
24	DMA OUT	DMA priority daisy-chain output. Usually connected to pin 22 (DMA IN).
25	+5V	+5-volt power supply. A total of 500mA is available for all accessory cards.
26	GND	System common ground.

Table 7-19b Expansion Slot Signals, continued

*Loading limits are for each card.

Pin Number	Name	Description
27	DMA IN	DMA priority daisy-chain input. Usually connected to pin 24 (DMA OUT).
28	INT IN	Interrupt priority daisy-chain input. Usually connected to pin 23 (INT OUT).
29	NMI'	Non-maskable interrupt to 6502. Pulling this line low starts an interrupt cycle with the interrupt-handling routine at location $03FB. This line has a 3300 ohm pullup resistor to +5V.
30	IRQ'	Interrupt request to 6502. Pulling this line low starts an interrupt cycle only if the interrupt-disable (I) flag in the 6502 is not set. Uses the interrupt-handling routine at location $03FE. This line has a 3300 ohm pullup resistor to +5V.
31	RES'	Pulling this line low initiates a reset routine, as described in Chapter 4.
32	INH'	Pulling this line low during ϕ1 inhibits (disables) the memory on the main circuit board. This line has a 3300 ohm pullup resistor to +5V.
33	−12V	−12 volt power supply. A total of 200mA is available for all accessory cards.
34	−5V	−5 volt power supply. A total of 200mA is available for all accessory cards.
35	3.58M	3.58 MHz color reference signal, on slot 7 **only**. This line can drive 2 LS TTL loads.*
36	7M	System 7 MHz clock. This line can drive 2 LS TTL loads.*
37	Q3	System 2 MHz asymmetrical clock. This line can drive 2 LS TTL loads.*
38	ϕ1	6502 phase 1 clock. This line can drive 2 LS TTL loads.*

Table 7-19c Expansion Slot Signals, continued

*Loading limits are for each card.

Pin Number	Name	Description
39	μPSYNC	The 6502 signals an operand fetch by driving this line high during the first read cycle of each instruction.
40	φ0	6502 phase 0 clock. This line can drive 2 LS TTL loads.*
41	DEVICE SELECT'	Normally high; goes low during φ0 when the 6502 addresses location $C0nX, where n is the connector number plus 8. This line can drive 10 LS TTL loads.*
42-49	D0-D7	Three-state buffered bi-directional data bus. Data becomes valid during φ0 high and remains valid until φ0 goes low. Each data line can drive one LS TTL load.*
50	+12V	+12 volt power supply. A total of 250mA is available for all accessory cards.

Video Signals on Slot 7

The video signals are available only on the auxiliary slot and not on the numbered expansion slots, except for slot 7. The video signals available on expansion slot 7 are SYNC', the composite horizontal and video sync signal, on pin 19, and 3.58M, the color reference signal, on pin 35. Early production Apple IIes, identified by a circuit board part number ending in -A, do not have this feature.

The signal that enables the graphics modes, named GR, is not normally available on the numbered expansion slots. You can make it available on pin 23 of slot 7 by completing the circuit at location X7 on the main circuit board. Remember to turn off the power before changing anything inside the Apple IIe. Also remember that changes such as this are at your own risk and may void the warranty.

The Auxiliary Slot

The large connector at the left side of the Apple IIe's main circuit board is the auxiliary slot. It is a 60-pin PC-card edge connector with pins on 0.10-inch centers. A PC card plugged into this connector has access to all of the signals used in producing the video display. These signals are described briefly in Tables

7-20a, 7-20b, and 7-20c. For further details, refer to the schematic diagram in Figures 7-14a, 7-14b, 7-14c, and 7-14d.

Many of the internal signals that are not available on the expansion slots are on the auxiliary slot. By using both kinds of connectors, manufacturing and repair personnel can gain access to most of the signals needed for diagnosing problems in the Apple IIe.

80-column Display Signals

The additional memory needed for producing an 80-column text display is on the 80-column text card, along with the buffers that transfer the data to the video data bus, as described above in the section "Text Displays". The signals that control the 80-column text data include the system clocks ϕo and ϕ1, the multiplexed RAM address RA0-RA7, the RAM address-strobe signals PRAS' and PCAS', and the auxiliary-RAM enable signals, EN80' and R/W80. The EN80' enable signal is controlled by the 80STORE soft switch described in Chapter 4. Data is sent to the auxiliary memory via the internal data bus MD0-MD7; the data is transferred to the video generator via the video data bus VID0-VID7.

Table 7-20a Auxiliary Slot Signals

Pin Number	Name	Description
1	3.58M	3.58 MHz video color reference signal. This line can drive two LS TTL loads.
2	VID7M	Clocks the video dots out of the 74166 parallel-to-serial shift register. This line can drive two LS TTL loads.
3	SYNC'	Video horizontal and vertical sync signal. This line can drive two LS TTL loads.
4	PRAS'	Multiplexed RAM row-address strobe. This line can drive two LS TTL loads.
5	VC	Third low-order vertical-counter bit. This line can drive two LS TTL loads.
6	C07X'	Hand-control reset signal. This line can drive two LS TTL loads.
7	WNDW'	Video non-blank window. This line can drive two LS TTL loads.
8	SEGA	First low-order vertical counter bit. This line can drive two LS TTL loads.
51, 10, 49, 48, 13, 14, 46, 9	RA0-RA7	Multiplexed RAM-address bus. This line can drive two LS TTL loads.
11, 12	ROMEN1, ROMEN2	Enable signals for the ROMs on the main circuit board.
15	R/W'	Read/write signal from 6502. This line can drive two LS TTL loads.
44, 43, 40, 39, 21, 20, 17, 16	MD0-MD7	Internal (unbuffered) data bus. This line can drive two LS TTL loads.

Table 7-20b Auxiliary Slot Signals, continued

Pin Number	Name	Description
45, 42, 41, 38, 22, 19, 18, 15	VID0-VID7	Video data bus. This three-state bus carries video data to the character generator.
23	ϕ0	6502 clock phase 0. This line can drive two LS TTL loads.
24	CLRGAT'	Color-burst gating signal. This line can drive two LS TTL loads.
25	80VID'	Enables 80-column display timing. This line can drive two LS TTL loads.
26	EN80'	Enable for auxiliary RAM. This line can drive two LS TTL loads.
27	ALTVID'	Alternative video output to the video summing amplifier.
28	SEROUT'	Video serial output from 74166 parallel-to-serial shift register.
29	ENVID'	Normally low; driving this line high disables the character generator such that the video dots from the shift register are all high (white), and alternative video can be sent out via ALTVID'. This line has a 1000 ohm pulldown resistor to ground.
30	+5	+5 volt power supply.
31	GND	System common ground.
32	14M	14.3 MHz master clock signal. This line can drive two LS TTL loads.
33	PCAS'	Multiplexed column-address strobe. This line can drive two LS TTL loads.
34	LDPS'	Strobe to video parallel-to-serial shift register. This signal goes low to load the contents of the video data bus into the shift register. This line can drive two LS TTL loads.

Table 7-20c Auxiliary Slot Signals, continued

Pin Number	Name	Description
35	R/W80	Read/write signal for RAM on the 80-column text card. This line can drive two LS TTL loads.
36	ϕ1	6502 clock phase 1. This line can drive two LS TTL loads.
37	CASEN'	Column-address enable. This signal is disabled (held high) during accesses to memory on the auxiliary card. This line can drive two LS TTL loads.
47	H0	Low-order horizontal byte counter. This line can drive two LS TTL loads.
50	AN3	Output of annunciator number 3. This line can drive two LS TTL loads.
52	R/W'	6502 read/write signal. This line can drive two LS TTL loads.
53	Q3	2 MHz asymmetrical clock. This line can drive two LS TTL loads.
54	SEGB	Second low-order vertical-counter bit. This line can drive two LS TTL loads.
55	ENFIRM	Normally high; pulling this line low disables ROM1 and ROM2 on the main circuit board. This line has a 3300-ohm pullup resistor to +5V.
56, 57	RA9', RA10'	Character-generator control signals from the IOU. This line can drive two LS TTL loads.
58	GR	Graphics-mode enable signal. This line can drive two LS TTL loads.
59	7M	7 MHz timing signal. This line can drive two LS TTL loads.
60	ENTMG'	Normally low; pulling this line high disables the master timing from the PAL. This line has a 1000 ohm pulldown resistor to ground.

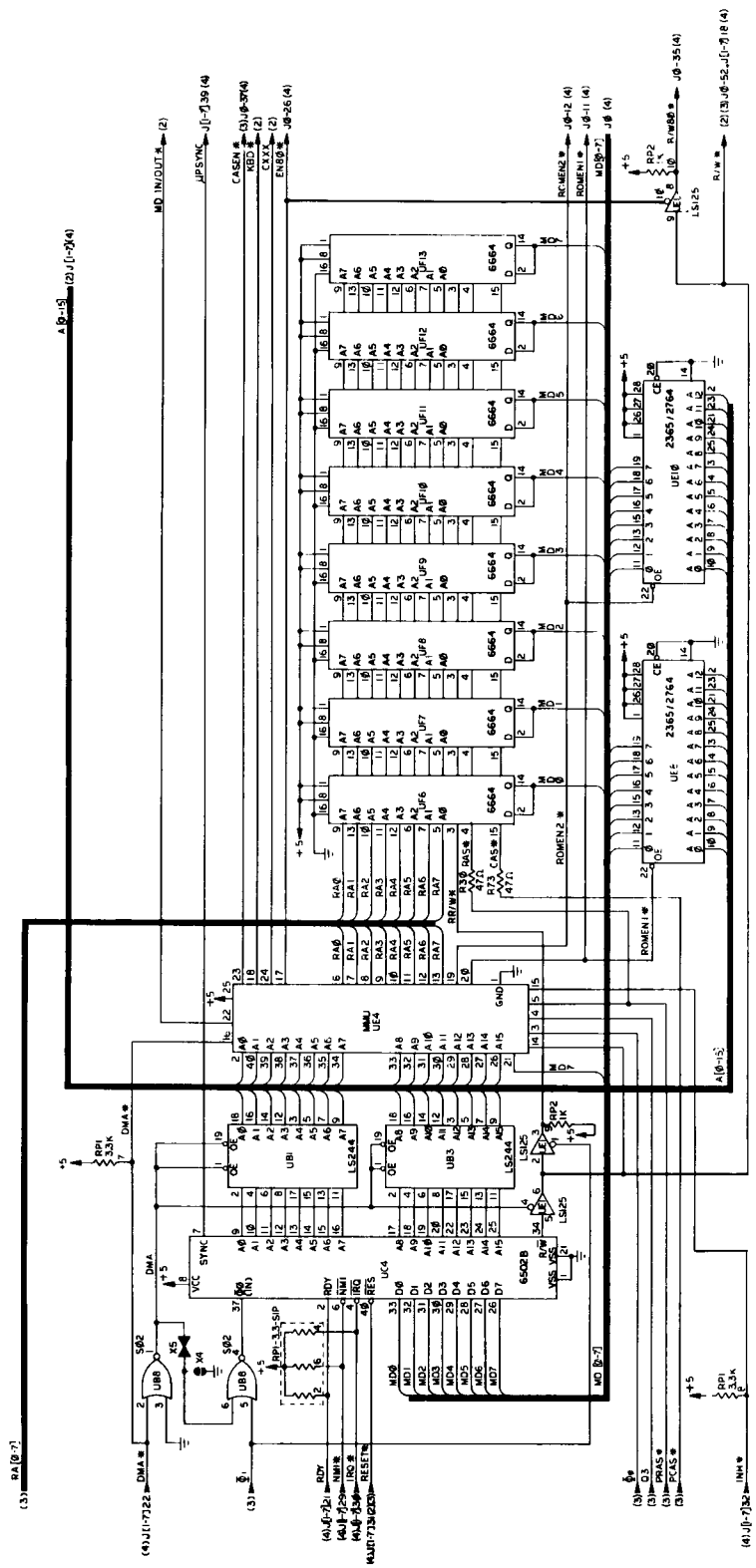


Figure 7-14a Schematic Diagram, part 1

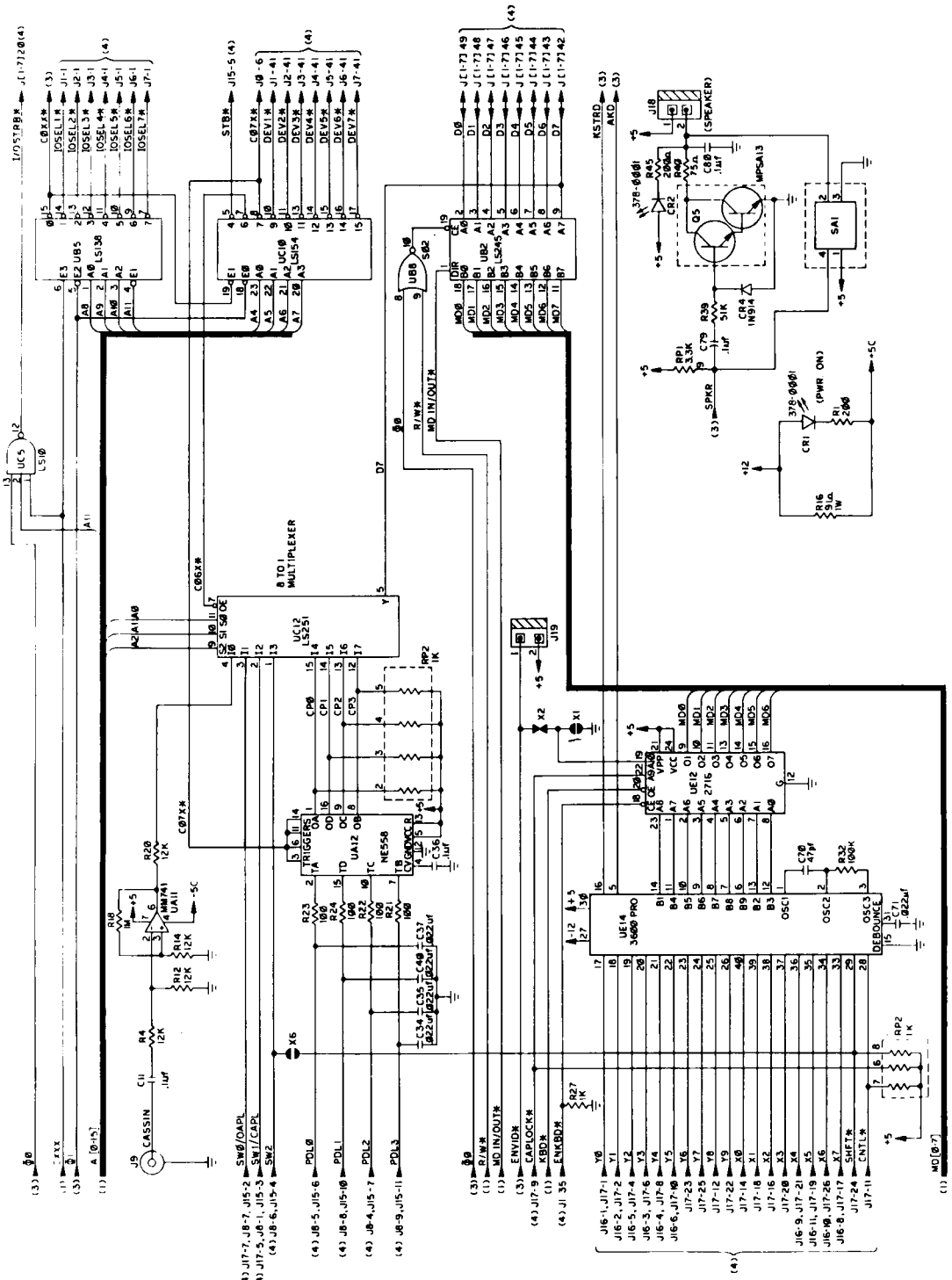


Figure 7-14b Schematic Diagram, part 2

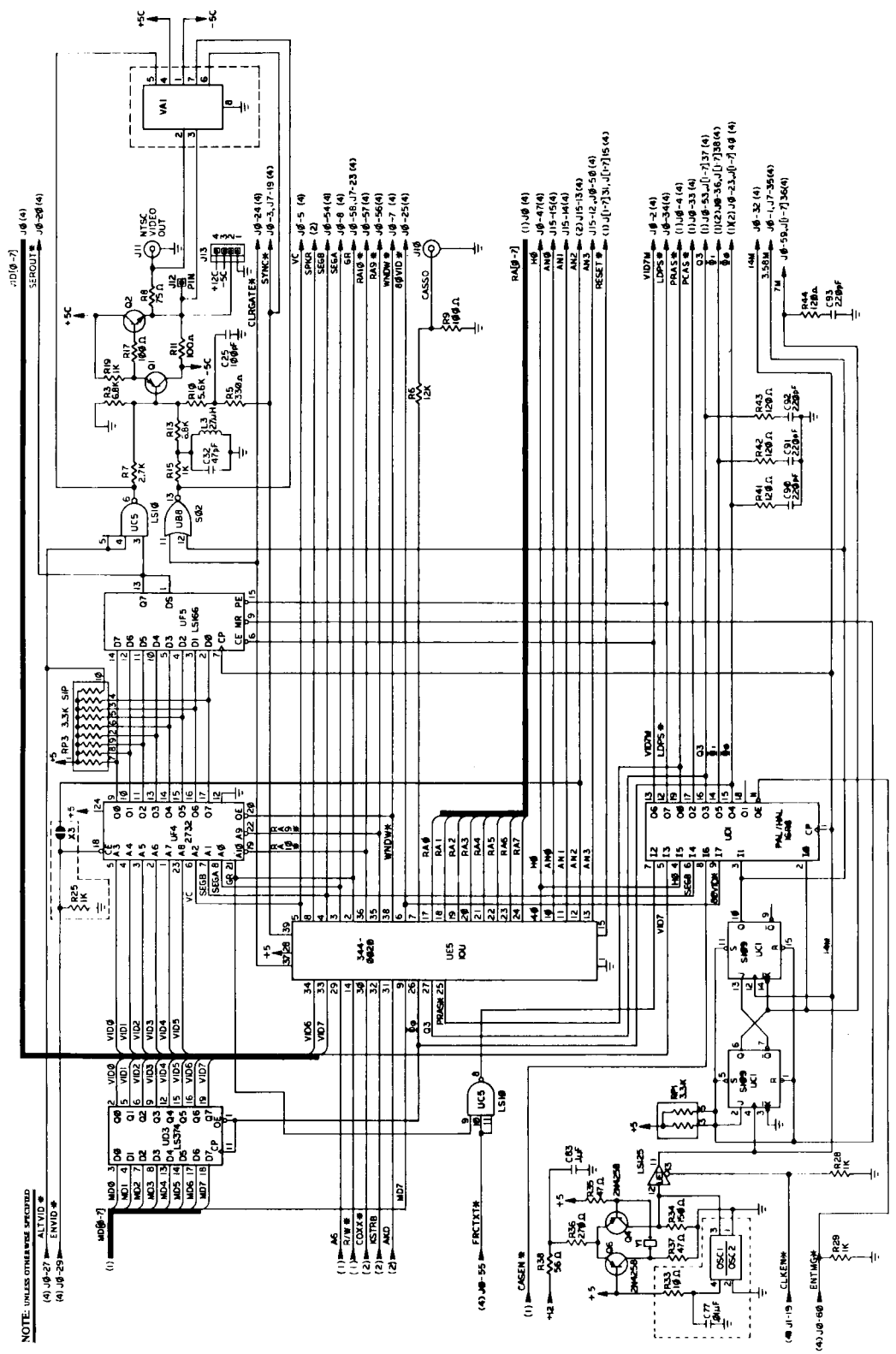


Figure 7-14c Schematic Diagram, part 3

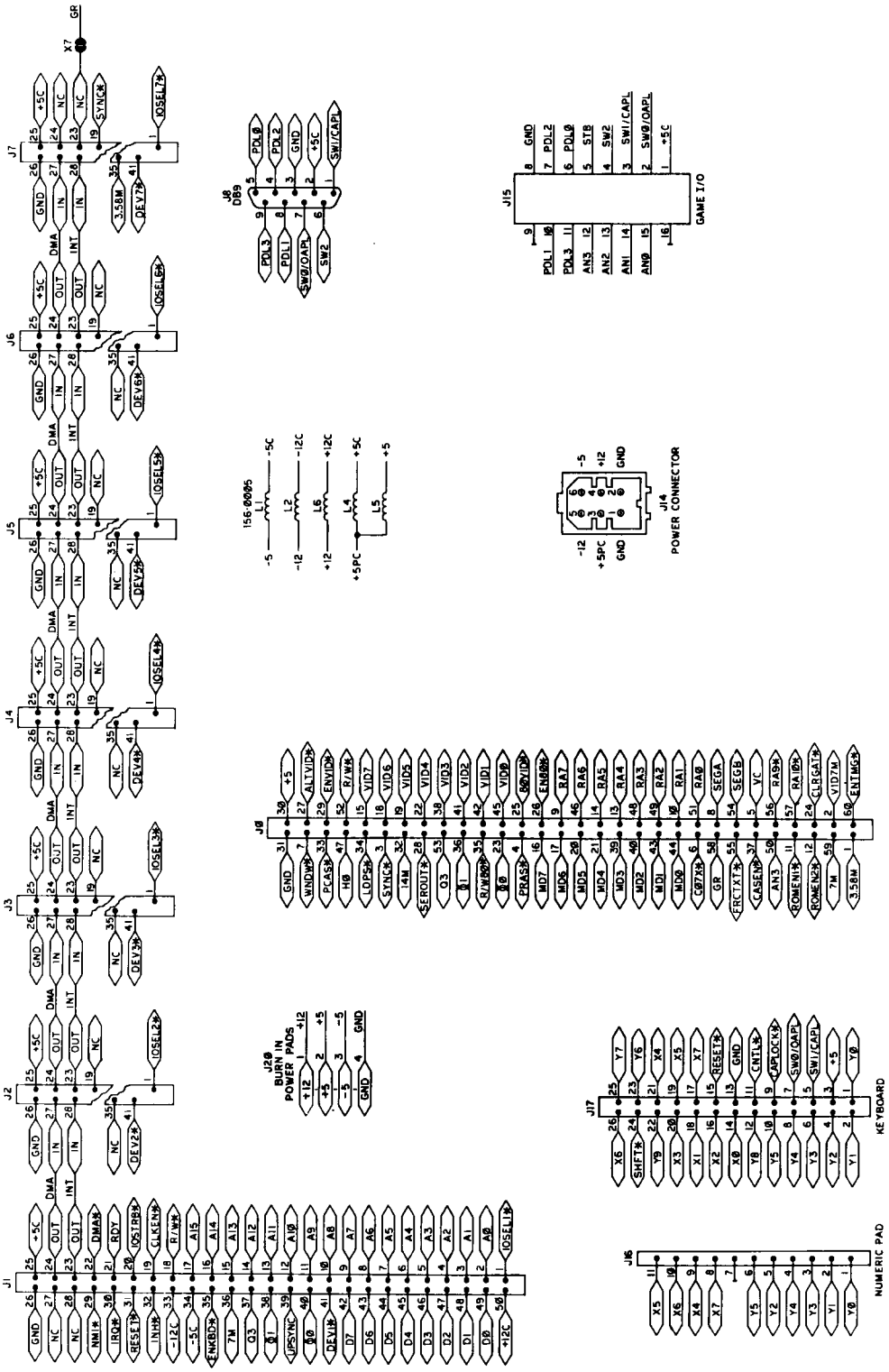
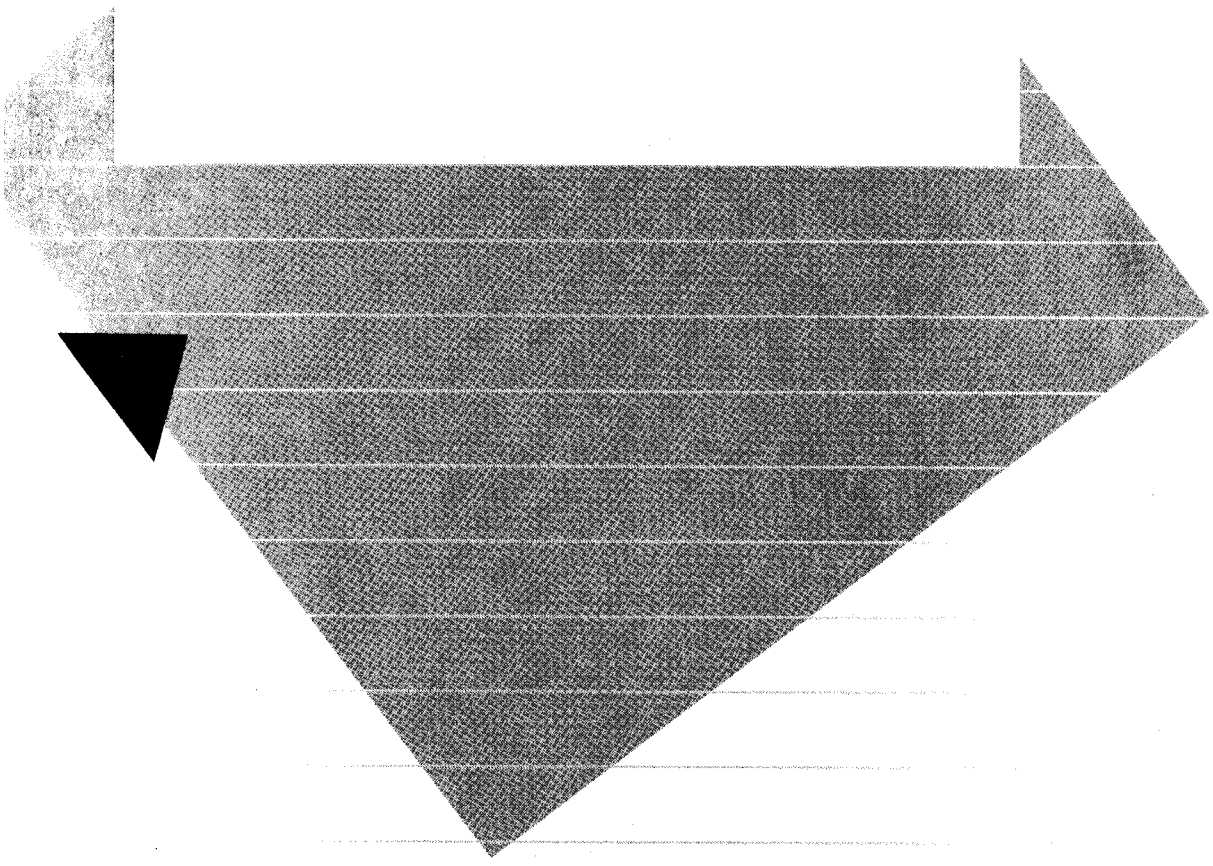


Figure 7-14d Schematic Diagram, part 4

Appendices



Appendix A

The 6502 Instruction Set

The Following Notation Applies to This Summary:

A	Accumulator
X, Y	Index Registers
M	Memory
$\overline{C}$	Borrow
P	Processor Status Register
S	Stack Pointer
✓	Change
—	No Change
+	Add
∧	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

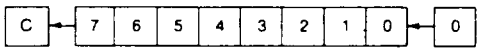


FIGURE 2. ROTATE ONE BIT LEFT (MEMORY OR ACCUMULATOR)

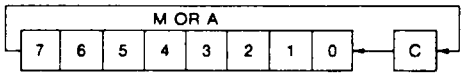
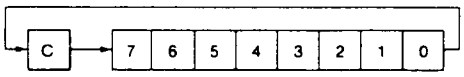


FIGURE 3. ROTATE ONE BIT RIGHT (MEMORY OR ACCUMULATOR)



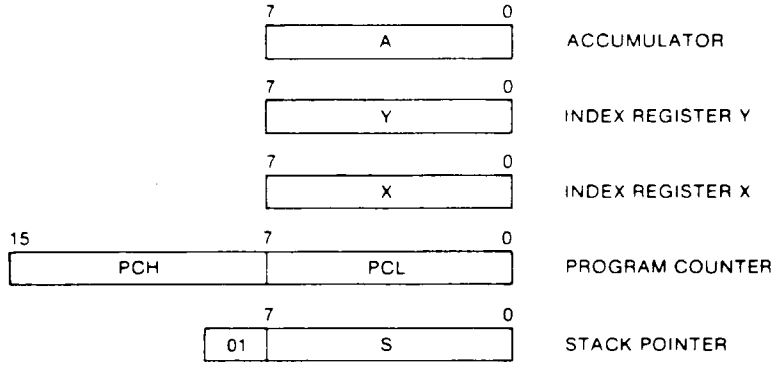
NOTE 1: BIT — TEST BITS

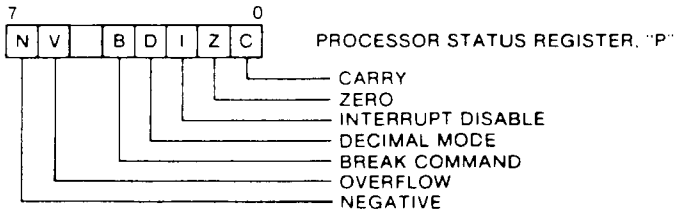
Bit 6 and 7 are transferred to the status register. If the result of A ∧ M is zero then Z=1, otherwise Z=0.

6502 Microprocessor Instructions

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

Programming Model





Instruction Codes

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg. N Z C I D V
ADC Add memory to accumulator with carry	A+M+C → A,C	Immediate	ADC #Oper	69	2	√√√– –√
		Zero Page	ADC Oper	65	2	
		Zero Page,X	ADC Oper,X	75	2	
		Absolute	ADC Oper	6D	3	
		Absolute,X	ADC Oper,X	7D	3	
		Absolute,Y	ADC Oper,Y	79	3	
		(Indirect,X)	ADC (Oper,X)	61	2	
		(Indirect),Y	ADC (Oper),Y	71	2	
AND "AND" memory with accumulator	A∧M → A	Immediate	AND #Oper	29	2	√√– – – –
		Zero Page	AND Oper	25	2	
		Zero Page,X	AND Oper,X	35	2	
		Absolute	AND Oper	2D	3	
		Absolute,X	AND Oper,X	3D	3	
		Absolute,Y	AND Oper,Y	39	3	
		(Indirect,X)	AND (Oper,X)	21	2	
		(Indirect),Y	AND (Oper),Y	31	2	
ASL Shift left one bit (Memory or Accumulator)	(See Figure 1)	Accumulator	ASL A	0A	1	√√√– – –
		Zero Page	ASL Oper	06	2	
		Zero Page,X	ASL Oper,X	16	2	
		Absolute	ASL Oper	0E	3	
		Absolute,X	ASL Oper,X	1E	3	
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∧M, M_7 → N, M_6 → V	Zero Page	BIT[1] Oper	24	2	M_7√– – – M_6
		Absolute	BIT[1] Oper	2C	3	
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 ↓ P ↓	Implied	BRK[2]	00	1	– – – 1 – –
BVC Branch on overflow clear	Branch on V=0	Relative	BVC Oper	50	2	– – – – – –

Note 1: Bits 6 and 7 are transferred to the status register. If the result of A AND M is zero, then Z = 1; otherwise Z = 0.

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

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" 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	√√
DEY Decrement index Y by one	Y — 1 → Y	Implied	DEY	88	1	√√

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	EOR #Oper	49	2	√√ ---
		Zero Page	EOR Oper	45	2	
		Zero Page,X	EOR Oper,X	55	2	
		Absolute	EOR Oper	4D	3	
		Absolute,X	EOR Oper,X	5D	3	
		Absolute,Y	EOR Oper,Y	59	3	
		(Indirect,X)	EOR (Oper,X)	41	2	
		(Indirect),Y	EOR (Oper),Y	51	2	
INC						
Increment memory by one	M + 1 → M	Zero Page	INC Oper	E6	2	√√----
		Zero Page,X	INC Oper,X	F6	2	
		Absolute	INC Oper	EE	3	
		Absolute,X	INC Oper,X	FE	3	
INX						
Increment index X by one	X + 1 → X	Implied	INX	E8	1	√√----
INY						
Increment index Y by one	Y + 1 → Y	Implied	INY	C8	1	√√----
JMP						
Jump to new location	(PC+1) → PCL	Absolute	JMP Oper	4C	3	------
	(PC+2) → PCH	Indirect	JMP (Oper)	6C	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	LDA #Oper	A9	2	√√----
		Zero Page	LDA Oper	A5	2	
		Zero Page,X	LDA Oper,X	B5	2	
		Absolute	LDA Oper	AD	3	
		Absolute,X	LDA Oper,X	BD	3	
		Absolute,Y	LDA Oper,Y	B9	3	
		(Indirect,X)	LDA (Oper,X)	A1	2	
		(Indirect),Y	LDA (Oper),Y	B1	2	
LDX						
Load index X with memory	M → X	Immediate	LDX #Oper	A2	2	√√----
		Zero Page	LDX Oper	A6	2	
		Zero Page,Y	LDX Oper,Y	B6	2	
		Absolute	LDX Oper	AE	3	
		Absolute,Y	LDX Oper,Y	BE	3	
LDY						
Load index Y with memory	M → Y	Immediate	LDY #Oper	A0	2	√√----
		Zero Page	LDY Oper	A4	2	
		Zero Page,X	LDY Oper,X	B4	2	
		Absolute	LDY Oper	AC	3	
		Absolute,X	LDY Oper,X	BC	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 (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 0D 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	√√√---

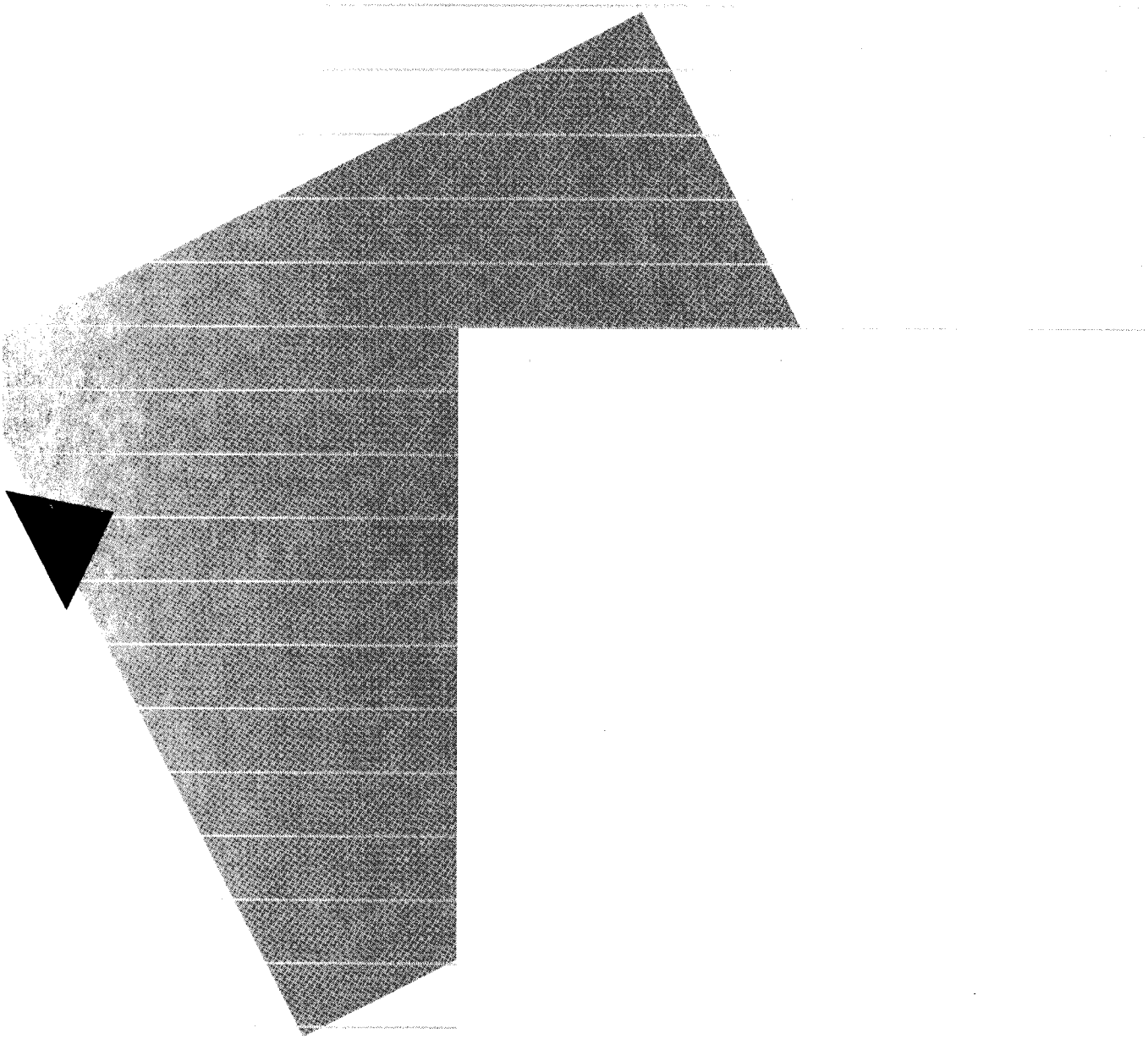
Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg. N Z C I D V
RTI Return from interrupt	P↑ PC↑	Implied	RTI	40	1	From Stack
RTS Return from subroutine	PC↑, PC+1 → PC	Implied	RTS	60	1	------
SBC Subtract memory from accumulator with borrow	A - M - $\overline{C}$ → A	Immediate	SBC #Oper	E9	2	√√√--√
		Zero Page	SBC Oper	E5	2	
		Zero Page,X	SBC Oper,X	F5	2	
		Absolute	SBC Oper	ED	3	
		Absolute,X	SBC Oper,X	FD	3	
		Absolute,Y	SBC Oper,Y	F9	3	
		(Indirect,X)	SBC (Oper,X)	E1	2	
		(Indirect),Y	SBC (Oper),Y	F1	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	STA Oper	85	2	------
		Zero Page,X	STA Oper,X	95	2	
		Absolute	STA Oper	8D	3	
		Absolute,X	STA Oper,X	9D	3	
		Absolute,Y	STA Oper,Y	99	3	
		(Indirect,X)	STA (Oper,X)	81	2	
		(Indirect),Y	STA (Oper),Y	91	2	
STX Store index X in memory	X → M	Zero Page	STX Oper	86	2	------
		Zero Page,Y	STX Oper,Y	96	2	
		Absolute	STX Oper	8E	3	
STY Store index Y in memory	Y → M	Zero Page	STY Oper	84	2	------
		Zero Page,X	STY Oper,X	94	2	
		Absolute	STY Oper	8C	3	
TAX Transfer accumulator to index X	A → X	Implied	TAX	AA	1	√√----
TAY Transfer accumulator to index Y	A → Y	Implied	TAY	A8	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 → A	Implied	TXA	8A	1	√√ – –
TXS Transfer index X to stack pointer	X → S	Implied	TXS	9A	1	– – – – – –
TYA Transfer index Y to accumulator	Y → A	Implied	TYA	98	1	√√– – – –

Hex Operation Codes

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

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



Appendix B

Tables

This appendix contains copies of the tables you will need to refer to a lot, for example, ASCII codes and soft-switch locations. The figures all have their original figure numbers so you can refer to the relevant sections in the text.

Table 2-2 Keyboard Memory Locations

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

Table 2-3a Keys and ASCII Codes

Codes are shown here in hexadecimal; to find the decimal equivalents, use Table 2-4.

Key	Normal	Control	Shift	Both
DELETE	7F	7F	7F	7F
LEFT-ARROW	08	08	08	08
TAB	09	09	09	09
DOWN-ARROW	0A	0A	0A	0A
UP-ARROW	0B	0B	0B	0B
RETURN	0D	0D	0D	0D
RIGHT-ARROW	15	15	15	15
ESC	1B	1B	1B	1B
space	20	20	20	20
' "	27	27	22	22
, <	2C	2C	3C	3C
- _	2D	2D	5F	1F
. >	2E	2E	3E	3E
/?	2F	2F	3F	3F
0)	30	30	29	29
1!	31	31	21	21
2@	32	00	40	00
3#	33	33	23	23
4$	34	34	24	24
5%	35	35	25	25
6^	36	1E	5E	1E
7&	37	37	26	26
8*	38	38	2A	2A
9(	39	39	28	28
: ;	3B	3B	3A	3A
= +	3D	3D	2B	2B
[{	5B	1B	7B	1B
\ |	5C	1C	7C	1C

Table 2-3b Keys and ASCII Codes, continued

Codes are shown here in hexadecimal; to find the decimal equivalents, use Table 2-4.

Key	Normal	Control	Shift	Both
]}	5D	1D	7D	1D
`~	60	60	7E	7E
A	61	01	41	01
B	62	02	42	02
C	63	03	43	03
D	64	04	44	04
E	65	05	45	05
F	66	06	46	06
G	67	07	47	07
H	68	08	48	08
I	69	09	49	09
J	6A	0A	4A	0A
K	6B	0B	4B	0B
L	6C	0C	4C	0C
M	6D	0D	4D	0D
N	6E	0E	4E	0E
O	6F	0F	4F	0F
P	70	10	50	10
Q	71	11	51	11
R	72	12	52	12
S	73	13	53	13
T	74	14	54	14
U	75	15	55	15
V	76	16	56	16
W	77	17	57	17
X	78	18	58	18
Y	79	19	59	19
Z	7A	1A	5A	1A

Table 2-4 The ASCII Character Set

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Table 2-6 The Display Character Sets

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

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

Table 2-7 Low-resolution Graphics Colors

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

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

Table 2-8 High-resolution Graphics Colors

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

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

Table 2-9 Video Display Page Locations

*Note: 80-column mode uses the 1024-byte page-1 locations in both main and auxiliary memory. The PAGE2 switch is used to select one or the other for storing data (see the section "Display Mode Switching").

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

Table 2-10 Display Soft Switches

(1) This mode is only effective when graphics-mode switch is ON.
(2) This switch has a different function when the 80-column text card's auxiliary text page is enabled for writing. Refer to the next section, "Addressing Display Pages Directly."
(3) This switch changes the function of the PAGE2 switch for addressing the auxiliary text memory on the extended 80-column text card. The next section describes how to do this.
(4) Reading this location returns the state of the vertical blanking signal VBL . The function of VBL is described in Chapter 7 in the section "Video Output Signals."

Name	Function	Location Hex	Location Decimal	Notes
ALTCHARSET	Alternate char. set on	$C00F	49167 -16369	Write
	Alternate char. set off	$C00E	49166 -16370	Write
	Read ALTCHARSET switch	$C01E	49182 -16354	Read
TEXT	Text mode on	$C051	49233 -16303	
	Text mode off (graphics)	$C050	49232 -16304	
	Read TEXT switch	$C01A	49178 -16358	Read
MIXED	Mixed-mode on	$C053	49235 -16301	1
	Mixed-mode off	$C052	49234 -16302	1
	Read MIXED switch	$C01B	49179 -16357	Read
PAGE2	Page 2 on	$C055	49237 -16299	2
	Page 2 off (Page 1)	$C054	49236 -16300	2
	Read PAGE2 switch	$C01C	49180 -16356	Read
HIRES	Hi-res mode on	$C057	49239 -16297	1
	Hi-res mode off	$C056	49238 -16298	1
	Read HIRES switch	$C01D	49181 -16355	Read
80COL	80-column display on	$C00D	49165 -16371	Write
	80-column display off	$C00C	49164 -16372	Write
	Read 80COL switch	$C01F	49183 -16353	Read
80STORE	Store in auxiliary memory	$C001	49153 -16383	Write, 3
	Store in main memory	$C000	49152 -16384	Write, 3
	Read 80STORE switch	$C018	49176 -16360	Read
VBL	Read vertical blanking	$C019	49177 -16359	Read, 4

Table 2-11 Annunciator Memory Locations

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

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

Table 2-12 Secondary I/O Memory Locations

For connector identification and pin numbers, refer to Tables 7-17 and 7-18.

Function	Address Decimal	Hex	Notes
Speaker	49200 -16336	$C030	Read
Cassette Out	49184 -16352	$C020	Read
Cassette In	49248 -16288	$C060	Read
Annunciator 0 On	49241 -16295	$C059	
Annunciator 0 Off	49240 -16296	$C058	
Annunciator 1 On	49243 -16293	$C05B	
Annunciator 1 Off	49242 -16294	$C05A	
Annunciator 2 On	49245 -16291	$C05D	
Annunciator 2 Off	49244 -16292	$C05C	
Annunciator 3 On	49247 -16289	$C05F	
Annunciator 3 Off	49246 -16290	$C05E	
Strobe Output	49216 -16320	$C040	Read
Switch Input 0	49249 -16287	$C061	Read
(OPEN-APPLE key)			
Switch Input 1	49250 -16286	$C062	Read
(SOLID-APPLE key)			
Switch Input 2	49251 -16285	$C063	Read
Analog Input Reset	49264 -16272	$C070	
Analog Input 0	49252 -16284	$C064	Read
Analog Input 1	49253 -16283	$C065	Read
Analog Input 2	49254 -16282	$C066	Read
Analog Input 3	49255 -16281	$C067	Read

Table 3-3a Control Characters with COUT1

(1) Only available when 80-column firmware is active.
(2) Only works from the keyboard.
(3) Doesn't work from the keyboard.

Control Character	ASCII Name	Apple IIe Name	Action Taken by COUT1	Notes
CONTROL - G	(BEL)	bell	Produces a 1000 Hz tone for 0.1 second.	
CONTROL - H	(BS)	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.	
CONTROL - J	(LF)	line feed	Moves cursor position down to next line in window; scrolls if needed.	
CONTROL - K	(VT)	clear EOS	Clears from cursor position to the end of the window.	1
CONTROL - L	(FF)	clear	Moves cursor position to upper left corner of window and clears window.	1
CONTROL - M	(CR)	return	Moves cursor position to left end of next line in window; scrolls if needed.	
CONTROL - N	(SO)	normal	Sets display format normal.	1, 3
CONTROL - O	(SI)	inverse	Sets display format inverse.	1, 3
CONTROL - Q	(DC1)	40-column	Sets display to 40-column.	1
CONTROL - R	(DC2)	80-column	Sets display to 80-column.	1
CONTROL - S	(DS3)	stop-list	Stops sending characters to the display, until a key is pressed.	1, 2

Table 3-3b Control Characters with COUT1, continued

(1) Only available when 80-column firmware is active.
(2) gotoXY is not supported under BASIC: see the *Apple Pascal Operating System Reference Manual.*

Control Character	ASCII Name	Apple IIe Name	Action Taken by COUT1	Notes
CONTROL - U	(NAK)	quit	Deactivates 80-column firmware, homes cursor, and clears screen.	1
CONTROL - V	(SYN)	scroll	Scrolls the display down one line, leaving the cursor in the current position.	1
CONTROL - W	(ETB)	scroll-up	Scrolls the display up one line, leaving the cursor in the current position.	1
CONTROL - Y	(EM)	home	Moves cursor position to upper left corner of window (but doesn't clear).	1
CONTROL - Z	(SUB)	clear line	Clears the line the cursor position is on.	1
CONTROL - \	(FS)	fwd. space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below.	1
CONTROL -]	(GS)	clear EOL	Clears line from cursor position to the right edge of the window.	1
CONTROL - ^	(RS)	gotoXY	Using the next two characters, minus 32, as one-byte X and Y values, moves the cursor position to CH=X, CV=Y.	1, 2

Table 3-4 Text Window Memory Locations

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

Table 3-6 Escape Codes

(1) Old-style cursor-control key: see text.
(2) Cursor-control key: see text.
(3) This code functions only when the 80-column firmware is active.

Escape Code	Function	Notes
ESC @	Clears the window and homes the cursor	
ESC A	Moves the cursor up one line	1
ESC B	Moves the cursor right one space	1
ESC C	Moves the cursor left one space	1
ESC D	Moves the cursor down one line	1
ESC E	Clears to the end of the line	
ESC F	Clears to the bottom of the window	
ESC I ESC ↑	Moves the cursor up one line and turns on escape mode	2
ESC J ESC ←	Moves the cursor left one space and turns on escape mode	2
ESC K ESC →	Moves the cursor right one space and turns on escape mode	2
ESC M ESC ↓	Moves the cursor down one line and turns on escape mode	2
ESC R	Turns on restricted-case mode	3
ESC T	Turns off restricted-case mode	3
ESC 4	Switches to 40-column mode, homes the cursor, and clears the screen	3
ESC 8	Switches to 80-column mode, homes the cursor, and clears the screen	3
ESC CONTROL - Q	Deactivates the 80-column firmware	3

Table 4-5 Bank Select Switches

(1) This switch write-enables RAM and read-enables ROM.
(2) Two successive reads to this switch enables RAM both for reading and writing.

Switch Address	Write RAM	Read RAM	Read ROM	4K RAM Bank: First	4K RAM Bank: Second	Notes
$C080		•			•	
$C081	•		•		•	1
$C082			•		•	
$C083	•	•			•	2
$C084		•			•	
$C085	•		•		•	1
$C086			•		•	
$C087	•	•			•	2
$C088		•		•		
$C089	•		•	•		1
$C08A			•	•		
$C08B	•	•		•		2
$C08C		•		•		
$C08D	•		•	•		1
$C08E			•	•		
$C08F	•	•		•		2

Table 4-6 Auxiliary-memory Select Switches

(1) When 80STORE is on, the PAGE2 switch selects main or auxiliary display memory.
(2) When 80STORE is on, the HIRES switch enables you to use the PAGE2 switch to switch between the high-resolution page-1 area in main memory or auxiliary memory.

Name	Function	Location Hex	Location Decimal	Notes
RAMRD	Read auxiliary memory	$C003	49155 -16381	Write
	Read main memory	$C002	49154 -16382	Write
	Read RAMRD switch	$C013	49171 -16365	Read
RAMWRT	Write auxiliary memory	$C005	49157 -16379	Write
	Write main memory	$C004	49156 -16380	Write
	Read RAMWRT switch	$C014	49172 -16354	Read
80STORE	On: access display page	$C001	49153 -16383	Write
	Off: use RAMRD, RAMWRT	$C000	49152 -16384	Write
	Read 80STORE switch	$C018	49176 -16360	Read
PAGE2	Page 2 on (Aux. memory)	$C055	49237 -16299	1
	Page 2 off (Main memory)	$C054	49236 -16300	1
	Read PAGE2 switch	$C01C	49180 -16356	Read
HIRES	On: access hi-res pages	$C057	49239 -16297	2
	Off: use RAMRD, RAMWRT	$C056	49238 -16298	2
	Read HIRES switch	$C01D	49181 -16355	Read
ALTZP	Auxiliary stack & z. p.	$C009	49161 -16373	Write
	Main stack & zero page	$C008	49160 -16374	Write
	Read ALTZP switch	$C016	49174 -16352	Read

Table 4-10 Page 3 Vectors

Vector address Decimal	Hex	Vector function
1008 1009	$3F0 $3F1	Address of the subroutine that handles BRK requests (normally $59, $FA).
1010 1011	$3F2 $3F3	Reset vector (see text).
1012	$3F4	Power-up byte (see text).
1013 1014 1015	$3F5 $3F6 $3F7	Jump instruction to the subroutine that handles Applesoft "&" commands (normally $4C, $58, $FF).
1016 1017 1018	$3F8 $3F9 $3FA	Jump instruction to the subroutine that handles user ([CONTROL] - Y) commands.
1019 1020 1021	$3FB $3FC $3FD	Jump instruction to the subroutine that handles non-maskable interrupts.
1022 1023	$3FE $3FF	Interrupt vector (address of the subroutine that handles interrupt requests).

Table 6-1 Peripheral-card I/O Memory Locations

Note: The enabling signal is marked with a prime, to indicate that it is an active-low signal.

Slot	Locations	Enabled by
1	$C090-$C09F	DEVICE SELECT'
2	$C0A0-$C0AF	DEVICE SELECT'
3	$C0B0-$C0BF	DEVICE SELECT'
4	$C0C0-$C0CF	DEVICE SELECT'
5	$C0D0-$C0DF	DEVICE SELECT'
6	$C0E0-$C0EF	DEVICE SELECT'
7	$C0F0-$C0FF	DEVICE SELECT'

Table 6-2 Peripheral-card ROM Memory Locations

Note: The enabling signal is marked with a prime, to indicate that it is an active-low signal.

Slot	Locations	Enabled by
1	$C100-$C1FF	I/O SELECT'
2	$C200-$C2FF	I/O SELECT'
3	$C300-$C3FF	I/O SELECT'
4	$C400-$C4FF	I/O SELECT'
5	$C500-$C5FF	I/O SELECT'
6	$C600-$C6FF	I/O SELECT'
7	$C700-$C7FF	I/O SELECT'

Table 6-3 Peripheral-card RAM Memory Locations

*Note: The RAM locations normally allocated to slot 3 are taken over by any card installed in the auxiliary slot.

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

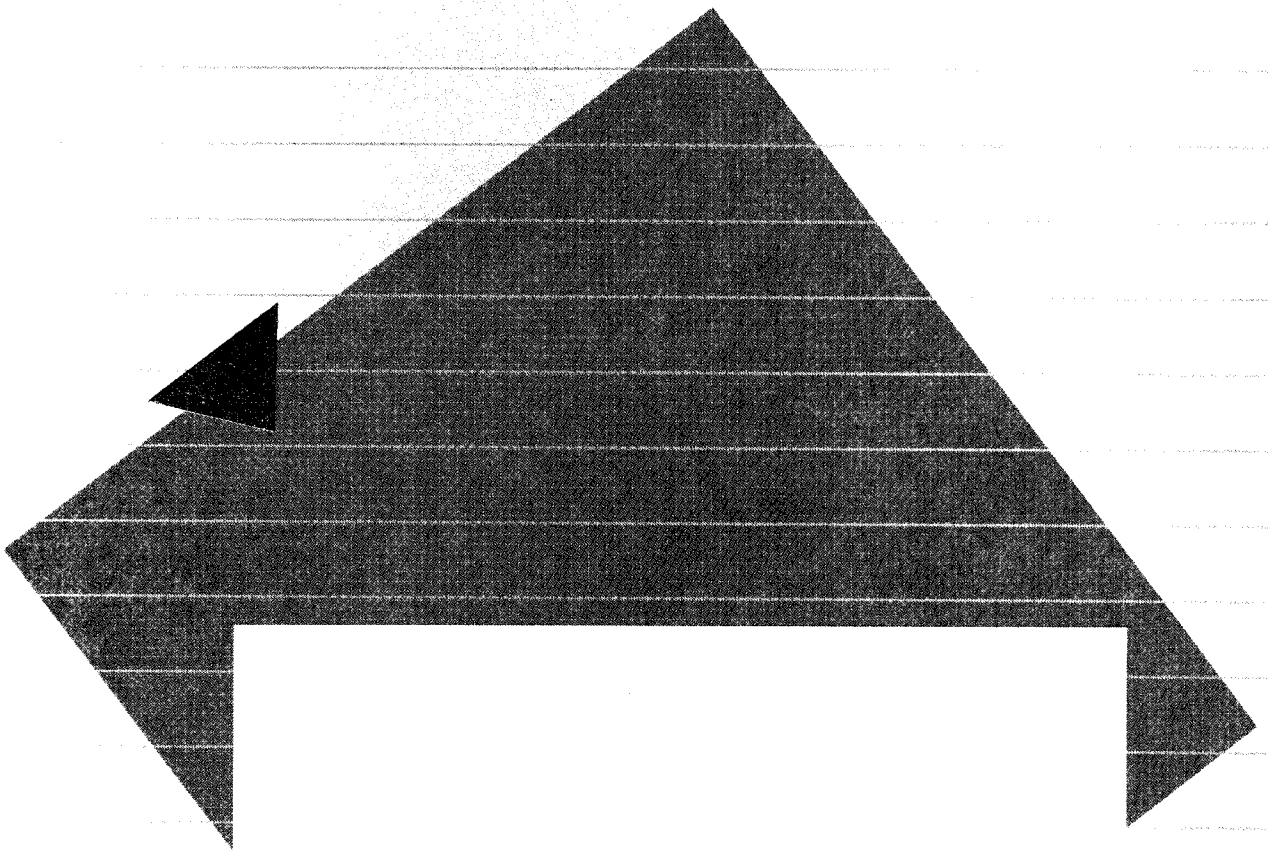
Table 6-4 Peripheral-card I/O Base Addresses

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

Table 6-5 I/O Memory Switches

Name	Function	Location Hex	Decimal		Notes
SLOTC3ROM	Slot ROM at $C300	$C00B	49163	−16373	Write
	Internal ROM at $C300	$C00A	49162	−16374	Write
	Read SLOTC3ROM switch	$C017	49175	−16361	Read
SLOTCXROM	Slot ROM at $Cx00	$C007	49159	−16377	Write
	Internal ROM at $Cx00	$C006	49158	−16378	Write
	Read SLOTCXROM switch	$C015	49173	−16363	Read

BLANK PAGE



Appendix C

Directory of Built-in Subroutines

Here is a list of useful subroutines in the Apple IIe's Monitor. To use these subroutines from machine-language programs, store data into the specified memory locations or 6502 registers as required by the subroutine and execute a JSR to the subroutine's starting address. After the subroutine performs its function, it returns with the 6502's registers changed as described.

Warning

For the sake of compatability between the Apple II Plus and the Apple IIe, do not jump into the middle of Monitor subroutines. The starting addresses are the same for all models of the Apple II, but the actual code is different.

BELL Output a bell character $FF3A

BELL writes a bell (CONTROL-G) character to the current output device. It leaves the accumulator holding $87.

BELL1 Send a beep to the speaker $FBDD

BELL1 generates a 1 kHz tone in the Apple IIe's speaker for 0.1 second. It scrambles the A and X registers.

CLREOL Clear to end of line $FC9C

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

CLEOLZ Clear to end of line $FC9E

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

CLREOP Clear to end of window $FC42

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

CLRSCR Clear the low-resolution screen $F832

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

CLRTOP Clear the low-resolution screen $F836

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

COUT Output a character $FDED

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

COUT1 Output to screen $FDF0

COUT1 displays the character in the accumulator on the Apple IIe'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, backspace, and bell. It returns with all registers intact.

CROUT Generate a RETURN $FD8E

CROUT sends a RETURN character to the current output device.

CROUT1 RETURN with clear $FD8B

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

GETLN Get an input line with prompt $FD6A

GETLN is the standard input subroutine for entire lines of characters, as described in Chapter 3. Your program calls GETLN

with the prompt character in location $33; GETLN returns with the input line in the input buffer (beginning at location $200) and the X register holding the length of the input line.

GETLNZ Get an input line $FD67

GETLNZ is an alternate entry point for GETLN that sends a carriage return to the standard output, then continues into GETLN.

GETLN1 Get an input line, no prompt $FD6F

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

HLINE Draw a horizontal line of blocks $F819

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

HOME Home cursor and clear $FC58

HOME clears the display and puts the cursor in the home position: the upper left corner of the screen.

IOREST Restore all registers $FF3F

IOREST loads the 6502's internal registers with the contents of memory locations $45 through $49.

IOSAVE Save all registers $FF4A

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

KEYIN Read the keyboard $FD1B

KEYIN is the keyboard input subroutine. It reads the Apple IIe's keyboard, waits for a keypress, and randomizes the random

number seed at $4E-$4F. When a key is pressed, KEYIN removes the blinking cursor from the display and returns with the keycode in the accumulator. KEYIN is described in Chapter 3.

MOVE Move a block of memory $FE2C

MOVE copies the contents of memory from one range of locations to another. This subroutine is the same as the MOVE command in the Monitor, except it takes its arguments from pairs of locations in memory, low-byte first. The destination address must be in A4 ($42-$43), the starting source address in A1 ($3C-$3D), and the ending source address in A2 ($3E-$3F) when your program calls MOVE.

NEXTCOL Increment color by 3 $F85F

NEXTCOL adds 3 to the current color (set by SETCOL) used for low-resolution graphics.

PLOT Plot on the low-resolution screen $F800

PLOT puts a single block of the color value set by SETCOL on the low-resolution display screen. 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 intact.

PRBLNK Print 3 spaces $F948

PRBLNK outputs three blank spaces to the standard output device. On return, the accumulator usually contains $A0, the X register contains 0.

PRBL2 Print many blank spaces $F94A

PRBL2 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.

PRBYTE Print a hexadecimal byte $FDDA

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

PREAD Read a hand control $FB1E

PREAD returns a number that represents the position of a hand control. You pass the number of the hand control in the X register. If this number is not valid (not equal to 0, 1, 2, or 3), strange things may happen. PREAD returns with a number from $00 to $FF in the Y register. The accumulator is scrambled.

PRERR Print ERR $FF2D

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

PRHEX Print a hexadecimal digit $FDE3

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

PRNTAX Print A and X in hexadecimal $F941

PRNTAX prints 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. On return, the contents of the accumulator are scrambled.

RDCHAR Get an input character or ESC code $FD35

RDCHAR is an alternate input subroutine that gets characters from the standard input subroutine, and also interprets the escape codes listed in Chapter 3.

RDKEY Get an input character $FD0C

RDKEY is the character input subroutine. It places a blinking cursor on the display at the cursor position and jumps to the subroutine whose address is stored in KSW (locations $38 and $39), usually the standard input subroutine KEYIN, which returns with a character in the accumulator.

READ Read a record from a cassette $FEFD

READ reads a series of tones at the cassette input port, converts them to data bytes, and stores the data in a specified range of

memory locations. Before calling READ, the address of the first byte must be in A1 ($3C-$3D) and the address of the last byte in A2 ($3E-3F.)

READ keeps a running Exclusive OR of the data bytes in CHKSUM ($2E). When the last location has been filled, READ reads one more byte and compares it with CHKSUM. If equal, READ sends out a beep, and returns; if not, it sends "ERR" through COUT, sends the beep, and returns.

SCRN Read the low-resolution graphics screen $F871

SCRN returns the color value of a single block on the low-resolution graphics display. Call it with the vertical position of the block in the accumulator and the horizontal position in the Y register. Call it as you would call PLOT (above). The color of the block will be returned in the accumulator. No other registers are changed.

SETCOL Set low-resolution graphics color $F864

SETCOL sets the color used for plotting in low-resolution graphics to the value passed in the accumulator. The colors and their values are listed in Table 2-7.

SETINV Set Inverse mode $FE80

SETINV sets the dislay format to inverse. COUT1 will then display all output characters as black dots on a white background. The Y register is set to $3F, all others are unchanged.

SETNORM Set Normal mode $FE84

SETNORM sets the display format to normal. COUT1 will then display all output characters as white dots on a black background. On return, the Y register is set to $FF, all others are unchanged.

VERIFY Compare two blocks of memory $FE36

VERIFY compares the contents of one range of memory to another. This subroutine is the same as the VERIFY command in the Monitor, except it takes its arguments from pairs of locations in memory, low-byte first. The destination address must be in A4 ($42-$43), the starting source address in A1 ($3C-$3D), and the ending source address in A2 ($3E-$3F) when your program calls VERIFY.

VLINE Draw a vertical line of blocks $F828

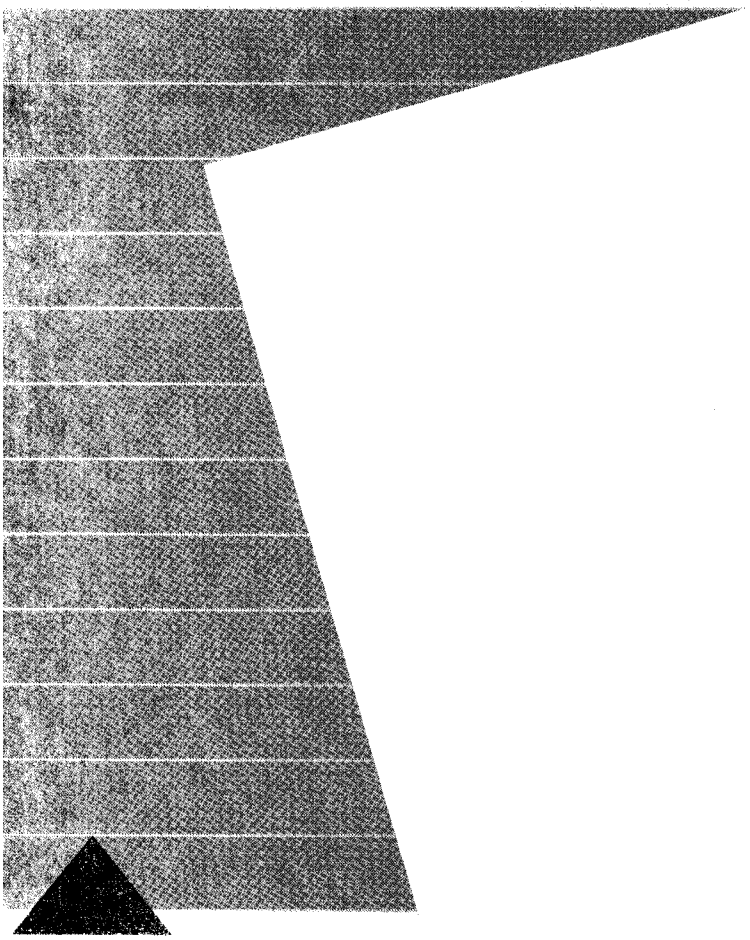
VLINE draws a vertical line of blocks of the color set by SETCOL on the low-resolution display. You should call VLINE with the horizontal coordinate of the line in the Y register, the top vertical coordinate in the accumulator, and the bottom vertical coordinate in location $2D. VLINE will return with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

WAIT Delay $FCA8

WAIT delays for a specific amount of time, then returns to the program that called it. The amount of delay is specified by the contents of the accumulator. With A the contents of the accumulator, the delay is 1/2(26+27A+5A^2) microseconds. WAIT returns with the accumulator zeroed and the X and Y registers undisturbed.

WRITE Write a record on a cassette $FECD

WRITE converts the data in a range of memory to a series of tones at the cassette output port. Before calling WRITE, the address of the first data byte must be in A1 ($3C-$3D) and the address of the last byte in A2 ($3E-$3F). The subroutine writes a ten-second continuous tone as a header, then writes the data followed by a one-byte checksum.



Appendix D

Differences Between the Apple IIe and the Apple II Plus

The Apple IIe is the latest model Apple II, and it includes several improvements over the older models. The improvements and other differences are listed here in approximately the order you are likely to encounter them: obvious differences first, technical details later. Each entry in the list includes references to the chapters in this manual where the item is described.

Full Keyboard

The Apple IIe has a full 62-key uppercase and lowercase keyboard. The keyboard includes fully-operational SHIFT and CAPS LOCK keys. It also includes four directional arrow keys for moving the cursor. Chapter 2 includes a description of the keyboard. The cursor-motion keys are described in Chapter 3.

Apple Keys

The keyboard of the Apple IIe has two keys marked with the Apple logo. These keys, called the OPEN-APPLE and SOLID-APPLE keys, are used with the RESET key to select special reset functions. They are connected to the buttons on the hand controls, so they can be used for special functions in programs.

Lowercase Display

The Apple IIe can display the full ASCII character set, uppercase and lowercase. For compatibility with older Apple II's, the standard display character set includes flashing uppercase instead of inverse-format lowercase; you can also switch to an alternate character set with inverse lowercase and uppercase, but no flashing. Chapter 2 includes a description of the display character sets. Chapter 3 tells you how to switch display formats.

Optional 80-column Display

With the addition of an 80-column text card, the Apple IIe can display 80 columns of text. The 80-column display is completely compatable with both graphics modes — you can even use it in mixed mode. (If you prefer, you can use an old-style 80-column card in an expansion slot instead.) Chapter 2 includes a description of the 80-column display.

Additional Escape and Control Keys

The display features mentioned above (and many others not mentioned) can be controlled from the keyboard by escape sequences and from programs by control characters. Chapter 3 includes descriptions of those escape codes and control characters.

Built-in Language Card

The 16K bytes of RAM you add to the Apple II Plus by installing the Language Card is built into the Apple IIe, giving it a standard memory size of 64K bytes. In the Apple IIe, this 16K-byte block of memory is called the Bank-switched Memory. It is described in Chapter 4.

Optional Auxiliary Memory

By installing an extended 80-column text card, you can add an alternate 64K bytes of RAM to the Apple IIe. Chapter 4 tells you how to use the additional memory. (Compatibility note: the extended 80-column text card also provides the 80-column display option.)

Auxiliary Slot

In addition to the normal expansion slots there is a special slot that is used either for the optional 80-column text card or for the extended 80-column text card. This slot is identified in Chapter 1 and described in Chapter 7.

Back Panel and Connectors

The Apple IIe has a metal back panel with space for several D-type connectors. Each peripheral card you add comes with a connector that you install in the back panel. Chapter 1 includes a description of the back panel; for details, see the installation instructions supplied with the peripheral cards.

Additional Soft Switches — Readable, Too

The additional display and memory features of the Apple IIe are controlled by soft switches like the ones on the Apple II Plus. On the Apple IIe, programs can also read the settings of the soft switches. Chapter 2 describes the soft switches that control the display features, and Chapter 4 describes the soft switches that control the memory features.

Built-in Self Test

The Apple IIe has additional built-in firmware that includes a self-test routine. The self-test is intended primarily for testing during manufacturing, but you can run it to be sure the Apple IIe is working correctly. The self-test is described in Chapter 4.

Forced Reset

Some programs on the Apple II Plus take control of the reset function to keep users from stopping the machine and copying the program. The Apple IIe has a forced reset that writes over the program in memory. By using the forced reset, you can restart the Apple IIe without turning power off and on and causing unnecessary stress on the circuits. The forced reset is described in Chapter 4.

Interrupt Handling

Even though most application programs don't use interrupts, the Apple IIe provides for interrupt-driven programs. For example, the 80-column firmware periodically enables interrupts while it is clearing the display (normally a long time to have interrupts locked out). Interrupts are discussed in Chapter 6.

Vertical Sync for Animators

Programs with animation can now stay in step with the display and avoid flickering objects in their displays. Chapter 7 includes a description of the video generation and the vertical sync.

Apple IIe Signature Byte

A program can find out whether it's running on an Apple IIe or on an older model Apple II by reading the byte at location $FBB3 in the System Monitor. In the Apple IIe Monitor, this byte's value is $06; in the Autostart Monitor (the standard Monitor on the Apple II Plus), its value is $EA. (Note: if you start up with DOS

and switch to Integer BASIC, the Autostart Monitor is active and the value at location $FBB3 is $EA, even on an Apple IIe.) Obviously, there are lots of other locations that have different values in the different versions of the Monitor; location $FBB3 was chosen because it will have the value $06 even in future revisions of the Apple IIe Monitor.

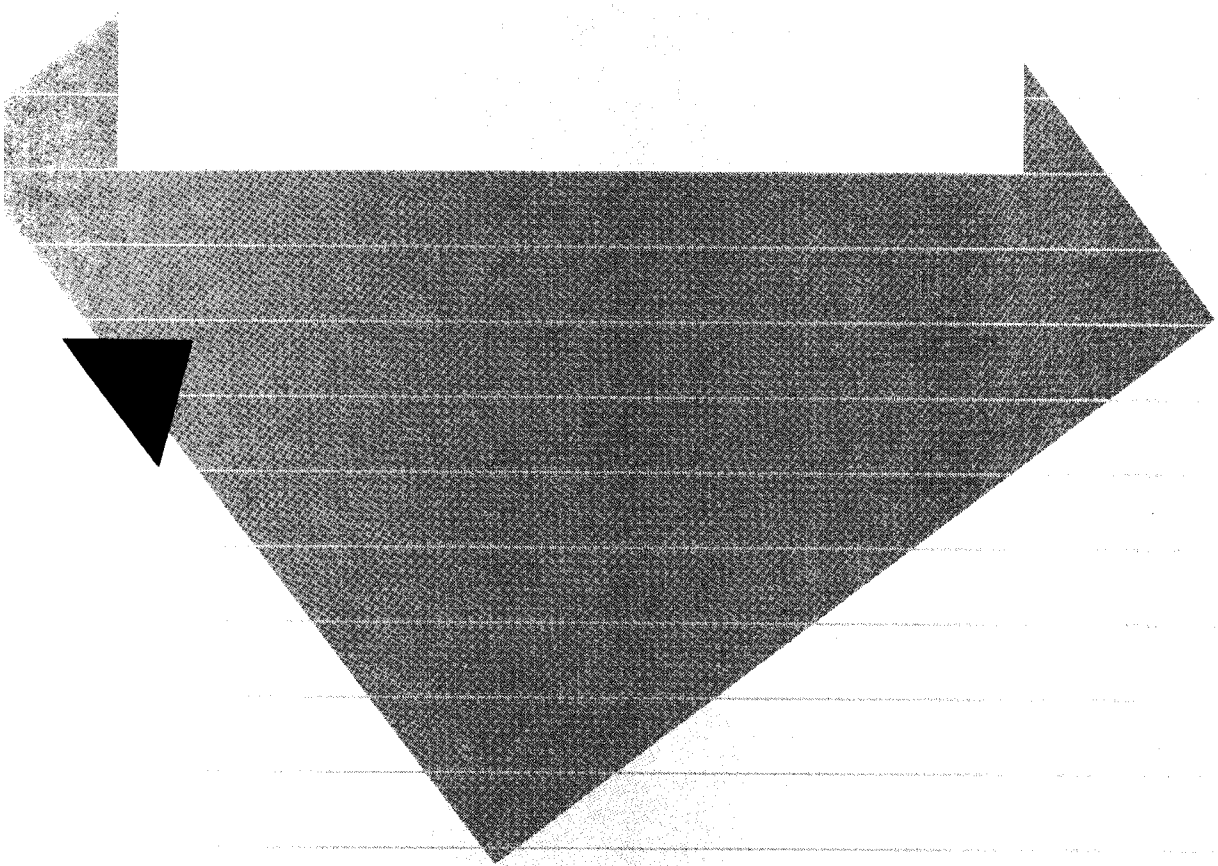
Hardware Implementation

The hardware implementation of the Apple IIe is radically different from the Apple II and Apple II Plus. Three of the more important differences are

- The custom ICs: the IOU and MMU;
- The video hardware, which uses ROM to generate both text and graphics;
- The peripheral data bus, which is fully buffered.

All of these features are described in Chapter 7.

BLANK PAGE



"A2_030-0357-B_1982_1-230.pict" 657 KB 2002-10-27 dpi: 600h x 600v pix: 4064h x 5287v

Glossary

6502: The microprocessor used in the Apple IIe computer.

accumulator: The register in the 6502 microprocessor where most computations are performed.

acronym: A word formed from the initial letters of a name or phrase, such as laser, from Light Amplification by Stimulated Emission of Radiation.

ADC: See **analog-to-digital converter**.

address: A number used to identify something, such as a location in the computer's memory.

analog: Represented in terms of a physical quantity, such as a voltage, frequency, length, or position, that can vary smoothly and continuously over a range of values. For example, a conventional 12-hour clock face (remember those?) is an analog device that represents the time of day in terms of the angles of the clock's hands. Compare **digital**.

analog-to-digital converter: A device that converts quantities from analog to digital form. For example, the Apple IIe's hand control converts the position of the control dial (an analog quantity) into a discrete number (a digital quantity) that changes abruptly even when the dial is turned smoothly.

AND: A logical operator that produces a true result if both of its operands are true, a false result if either or both of its operands are false; compare **OR**, **exclusive OR**, **NOT**.

Apple IIe: A personal computer in the Apple II family, manufactured and sold by Apple Computer.

Apple IIe 80-Column Text Card: A peripheral card made and sold by Apple Computer that plugs into the Apple IIe's auxiliary slot and converts the computer's display of text from 40- to 80-column width.

Apple IIe Extended 80-Column Text Card: A peripheral card made and sold by Apple Computer that plugs into the Apple IIe's auxiliary slot and converts the computer's display of text from 40- to 80-column width while extending its memory capacity by 64K bytes.

Applesoft: An extended version of the BASIC programming language used with the Apple IIe computer. An interpreter for creating and executing programs in Applesoft is built into the Apple IIe system in firmware.

ASCII: American Standard Code for Information Interchange; a code in which the numbers from `0` to `127` stand for text characters, used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

assembler: A language translator that converts a program written in assembly language into an equivalent program in machine language.

assembly language: A low-level programming language in which individual machine-language instructions are written in a symbolic form more easily understood by a human programmer than machine language itself.

auxiliary slot: The special expansion slot inside the Apple IIe used for the Apple 80-Column Text Card or Extended 80-Column Text Card.

back panel: The rear face of the Apple IIe computer, which includes the power switch, the power connector, and connectors for a video display device, a cassette tape recorder, and other peripheral devices.

bandwidth: A measure of the range of frequencies a device can handle. In the case of a video monitor, greater bandwidth enables it to display more information; to display 80 columns of text, a monitor should have a bandwidth of at least 12 MHz.

base address: In indexed addressing, the fixed component of an address.

binary: The representation of numbers in terms of powers of two, using the two digits `0` and `1`. Commonly used in computers, since the values `0` and `1` can easily be represented in physical form in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the display screen.

binary operator: An operator that combines two operands to produce a result; for example, `OR` is a binary logical operator. Compare **unary operator**.

bit: A binary digit (`0` or `1`); the smallest possible unit of information, consisting of a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing.

bit bucket: The final resting place of all information; see **write-only memory**.

board: See **printed-circuit board**.

boot: To start up a computer by loading a program into memory from an external storage medium such as a disk. Often accomplished by first loading a small program whose purpose is to read the larger program into memory. The program is said to "pull itself in by its own bootstraps"; hence the term *bootstrapping* or *booting*.

bootstrap: See **boot**.

buffer: An area of the computer's memory reserved for a specific purpose, such as to hold graphical information to be displayed on the screen or text characters being read from some peripheral device. Often used as an intermediary "holding area" for transferring information between devices operating at different speeds, such as the computer's processor and a printer or disk drive. Information can be stored into the buffer by one device and then read out by the other at a different speed.

bus: A group of wires that transmit related information, such as the bits of an address, from one part of a computer system to another.

byte: A unit of information consisting of a fixed number of bits; on the Apple IIe, one byte consists of eight bits and can hold any value from `0` to `255`.

card: See **peripheral card**.

carrier: A broadcast radio signal that is modulated in order to transmit information.

carry flag: A status bit in the 6502 microprocessor, used in addition and subtraction to hold the high-order bit (the *carry bit*).

cathode-ray tube: An electronic device, such as a television picture tube, that produces images on a screen coated with phosphors that emit light when struck by a focused beam of electrons.

central processing unit: See **processor**.

character: A letter, digit, punctuation mark, or other written symbol used in printing or displaying information in a form readable by humans.

character code: A number used to represent a text character for processing by a computer system.

chip: The small piece of semiconducting material (usually silicon) on which an integrated circuit is fabricated. The word *chip* properly refers only to the piece of silicon itself, but is often used for an integrated circuit and its package; see **integrated circuit**.

code: (1) A number or symbol used to represent some piece of information in a compact or easily processed form. (2) The statements or instructions making up a program.

cold start: The process of starting up the Apple IIe when the power is first turned on (or as if the power had just been turned on) by loading the operating system into main memory, then loading and running a program. Compare **warm start**.

command: A communication from the user to a computer system (usually typed from the keyboard) directing it to perform some immediate action.

compiler: A language translator that converts a program written in a high-level programming language into an equivalent program in some lower-level language (such as machine language) for later execution. Compare **interpreter**.

component: A part; in particular, a part of a computer system.

composite video: A video signal that includes both display information and the synchronization (and other) signals needed to display it.

computer: An electronic device for performing predefined (programmed) computations at high speed and with great accuracy.

computer system: A computer and its associated hardware, firmware, and software.

connector: a physical device such as a plug, socket, or jack, used to connect one hardware component of a system to another.

control character: A character that controls or modifies the way information is printed or displayed. Control characters have ASCII codes between 0 and 31 and are typed from the Apple IIe keyboard by holding down the CONTROL key while typing some other character. For example, the character CONTROL -M (ASCII code 13) means "return to the beginning of the line" and is equivalent to the RETURN key.

controller card: A peripheral card that connects a device such as a printer or disk drive to the Apple IIe and controls the operation of the device.

CPU: Central processing unit; see **processor**.

crash: To cease operating unexpectedly, possibly damaging or destroying information in the process.

CRT: See **cathode-ray tube**.

cursor: A marker or symbol displayed on the screen that marks where the user's next action will take effect or where the next character typed from the keyboard will appear.

DAC: See **digital-to-analog converter**.

data: Information; especially information used or operated on by a program.

debug: To locate and correct an error or the cause of a problem or malfunction in a computer system. Typically used to refer to software-related problems; compare **troubleshoot**.

decimal: The common form of number representation used in everyday life, in which numbers are expressed in terms of powers of ten, using the ten digits 0 to 9.

default: A value, action, or setting that is automatically used by a computer system when no other explicit information has been given. For example, if a command to run a program from a disk does not identify which disk drive to use, the Disk Operating System will automatically use the same drive that was used in the last operation.

defenestration: The act of throwing something through, from, or out of a window. Not recommended handling of an Apple IIe.

demodulate: To recover the information being transmitted by a modulated signal; for example, a conventional radio receiver demodulates an incoming broadcast signal to convert it into sound emitted by a speaker.

device: (1) A physical apparatus for performing a particular task or achieving a particular purpose. (2) In particular, a hardware component of a computer system.

device handler: See **device driver**.

device driver: A program that manages the transfer of information between the computer and a peripheral device.

digit: (1) One of the characters 0 to 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary or 0 to 9 and A to F in hexadecimal.

digital: Represented in a discrete (noncontinuous) form, such as numerical digits. For example, contemporary digital clocks display the time in numerical form (such as 2:57) instead of using the positions of a pair of hands on a clock face. Compare **analog**.

digital-to-analog converter: A device that converts quantities from digital to analog form.

DIP: See **dual in-line package**.

disassembler: A language translator that converts a machine-language program into an equivalent program in assembly language, more easily understood by a human programmer. The opposite of an assembler.

disk: An information storage medium consisting of a flat, circular magnetic surface on which information can be recorded in the form of small magnetized spots, similarly to the way sounds are recorded on tape.

disk controller card: A peripheral card that connects one or two disk drives to the Apple IIe and controls their operation.

disk drive: A peripheral device that writes and reads information on the surface of a magnetic disk.

diskette: A term sometimes used for the small (5-1/4-inch) flexible disks used with the Apple Disk II drive.

Disk II drive: A model of disk drive made and sold by Apple Computer for use with the Apple IIe computer; uses 5-1/4-inch flexible ("floppy") disks.

Disk Operating System: An optional software system for the Apple IIe that enables the computer to control and communicate with one or more Disk II drives.

display: (1) Information exhibited visually, especially on the screen of a display device. (2) To exhibit information visually. (3) A display device.

display device: A device that exhibits information visually, such as a television receiver or video monitor.

display screen: The glass or plastic panel on the front of a display device, on which images are displayed.

DOS: See **Disk Operating System**.

dual in-line package: An integrated circuit packaged in a narrow rectangular box with a row of metal pins along each side; similar in appearance to an armored centipede.

edit: To change or modify; for example, to insert, remove, replace, or move text in a document.

editor: A program that enables the user to create and edit information of a particular form; for example, a *text editor* or a *graphics editor*.

effective address: In machine-language programming, the address of the memory location on which a particular instruction actually operates, which may be arrived at by indexed addressing or some other addressing method.

error message: A message displayed or printed to notify the user of an error or problem in the execution of a program.

escape mode: A state of the Apple IIe computer, entered by pressing the ESC key, in which certain keys on the keyboard take on special meanings for positioning the cursor and controlling the display of text on the screen.

escape sequence: A sequence of keystrokes beginning with the ESC key, used for positioning the cursor and controlling the display of text on the screen.

exclusive OR: A logical operator that produces a true result if one of its operands is true and the other false, a false result if its operands are both true or both false; compare OR, AND, NOT.

execute: To perform or carry out a specified action or sequence of actions, such as those described by a program.

expansion slot: A connector inside the Apple IIe computer in which a peripheral card can be installed; sometimes called *peripheral slot*.

firmware: Those components of a computer system consisting of programs stored permanently in read-only memory. Such programs (for example, the Applesoft interpreter and the Apple IIe Monitor program) are built into the computer at the factory; they can be executed at any time but cannot be modified or erased from main memory. Compare **hardware**, **software**.

fixed-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position within the number. Typically, the point is considered to lie at the right end of the number, so that the number is interpreted as an integer. Compare **floating-point**.

flexible disk: A disk made of flexible plastic; often called a "floppy" disk. Compare **rigid disk**.

floating-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is permitted to "float" to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Compare **fixed-point**.

GAME I/O connector: A special 16-pin connector inside the Apple IIe, originally designed for connecting hand controls to the computer, but also used for connecting some other peripheral devices. Compare **hand control connector**.

graphics: (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text**.

hand control: An optional peripheral device that can be connected to the Apple IIe's hand control connector and has a rotating dial and a pushbutton; typically used to control game-playing programs, but can be used in more serious applications as well.

hand control connector: A 9-pin connector on the Apple IIe's back panel, used for connecting hand controls to the computer. Compare **GAME I/O connector**.

hardware: Those components of a computer system consisting of physical (electronic or mechanical) devices. Compare **software, firmware**.

hertz: The unit of frequency of vibration or oscillation, also called cycles per second; named for the physicişt Heinrich Hertz and abbreviated Hz. The current provided by a standard power outlet alternates at a rate of 60 hertz; that is , it changes polarity 60 times each second. The Apple IIe's 6502 microprocessor operates at a clock frequency of 1 million hertz, or 1 megahertz (MHz).

hexadecimal: The representation of numbers in terms of powers of sixteen, using the sixteen digits 0 to 9 and A to F. Hexadecimal numbers are easier for humans to read and understand than binary numbers, but can be converted easily and directly to binary form: each hexadecimal digit corresponds to a sequence of four binary digits, or bits.

high-level language: A programming language that is relatively easy for humans to understand. A single statement in a high-level language typically corresponds to several instructions of machine language.

high-order byte: The more significant half of a memory address or other two-byte quantity. In the Apple IIe's 6502 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

high-resolution graphics: The display of graphics on the Apple IIe's display screen as a six-color array of points, 280 columns wide and 192 rows high.

hold time: In computer circuits, the amount of time a signal must remain valid after some related signal has been turned off; compare setup time.

Hz: See **hertz**.

IC: See **integrated circuit**.

information: Facts, concepts, or instructions represented in an organized form.

index: (1) A number used to identify a member of a list or table by its sequential position. (2) A list or table whose entries are identified by sequential position. (3) In machine-language programming, the variable component of an indexed address, contained in an index register and added to the base address to form the effective address.

indexed addressing: A method of specifying memory addresses used in machine-language programming.

index register: A register in a computer processor that holds an index for use in indexed addressing. The Apple IIe's 6502 microprocessor has two index registers, called the X register and the Y register.

input: (1) Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem. (2) The act or process of transferring such information.

instruction: A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

integer: A whole number, with no fractional part; represented inside the computer in fixed-point form. Compare **real number**.

integrated circuit: An electronic component consisting of many circuit elements fabricated on a single piece of semiconducting material, such as silicon; see **chip**.

interface: The devices, rules, or conventions by which one component of a system communicates with another.

interface card: A peripheral card that implements a particular interface (such as a parallel or serial interface) by which the computer can communicate with a peripheral device such as a printer or modem.

interpreter: A language translator that reads a program written in a particular programming language and immediately carries out the actions that the program describes. Compare **compiler**.

interrupt: A temporary suspension in the execution of a program by a computer in order to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

inverse video: The display of text on the computer's display screen in the form of black dots on a white (or other single phosphor color) background, instead of the usual white dots on a black background.

I/O: Input/output; the transfer of information into and out of a computer. See **input**, **output**.

I/O device: Input/output device; a device that transfers information into or out of a computer. See **input**, **output**, **peripheral device**.

I/O link: A fixed location that contains the address of an input/output subroutine in the Apple IIe Monitor program.

K: Two to the tenth power, or 1024 (from the Greek root *kilo*, meaning one thousand); for example, 64K equals 64 times 1024, or 65,536.

keyboard: The set of keys built into the Apple IIe computer, similar to a typewriter keyboard, for typing information to the computer.

keystroke: The act of pressing a single key or a combination of keys (such as CONTROL-C) on the Apple IIe keyboard.

kilobyte: A unit of information consisting of 1K (1024) bytes, or 8K (8192) bits; see K.

KSW: The symbolic name of the location in the Apple IIe's memory where the standard input link is stored; stands for "keyboard switch." See **I/O link**.

language: See **programming language**.

language translator: A system program that reads a program written in a particular programming language and either executes it directly or converts it into some other language (such as machine language) for later execution. See **interpreter**, **compiler**, **assembler**.

load: To transfer information from a peripheral storage medium (such as a disk) into main memory for use; for example, to transfer a program into memory for execution.

location: See **memory location**.

logical operator: An operator, such as AND, that combines logical values to produce a logical result.

low-level language: A programming language that is relatively close to the form that the computer's processor can execute directly. Low-level languages available for the Apple IIe include 6502 machine language and 6502 assembly language.

low-order byte: The less significant half of a memory address or other two-byte quantity. In the Apple IIe's 6502 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

low-power Shottkey: A type of TTL integrated circuit having lower power and higher speed than a conventional TTL integrated circuit.

low-resolution graphics: The display of graphics on the Apple IIe's display screen as a sixteen-color array of blocks, 40 columns wide and 48 rows high.

LS: See **low-power Shottkey**.

machine language: The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor (such as the 6502 microprocessor used in the Apple IIe) has its own form of machine language.

main memory: The memory component of a computer system that is built into the computer itself and whose contents are directly accessible to the processor.

memory: A hardware component of a computer system that can store information for later retrieval; see **main memory**, **random-access memory**, **read-only memory**, **read-write memory**, **write-only memory**.

memory location: A unit of main memory that is identified by an address and can hold a single item of information of a fixed size; in the Apple IIe, a memory location holds one byte, or eight bits, of information.

memory-resident: (1) Stored permanently in main memory, as firmware. (2) Held continually in main memory even while not in use, as the Disk Operating System.

MHz: Megahertz; one million hertz. See **hertz**.

microcomputer: A computer, such as the Apple IIe, whose processor is a microprocessor.

microprocessor: A computer processor contained in a single integrated circuit, such as the 6502 microprocessor used in the Apple IIe.

microsecond: One millionth of a second; abbreviated μs.

millisecond: One thousandth of a second; abbreviated ms.

mode: A state of a computer or system that determines its behavior.

modem: Modulator/demodulator; a peripheral device that enables the computer to transmit and receive information over a telephone line.

modulate: To modify or alter a signal so as to transmit information; for example, conventional broadcast radio transmits sound by modulating the amplitude (amplitude modulation, or AM) or the frequency (frequency modulation, or FM) of a carrier signal.

monitor: See **video monitor.**

Monitor program: A system program built into the Apple IIe in firmware, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level.

nanosecond: One billionth (in British usage, one thousand-millionth) of a second; abbreviated ns.

network: A collection of interconnected, individually controlled computers, together with the hardware and software used to connect them.

nibble: A unit of information equal to half a byte, four bits, or fifty cents; can hold any value from 0 to 15. Sometimes spelled *nybble*.

NOT: A unary logical operator that produces a true result if its operand is false, a false result if its operand is true; compare AND, OR, exclusive OR.

NTSC: (1) National Television Standards Committee; the committee that defined the standard format used for transmitting broadcast video signals in the United States. (2) The standard video format defined by the NTSC.

object code: See **object program**.

object program: The translated form of a program produced by a language translator such as a compiler or assembler; also called *object code*. Compare **source program**.

op code: See **operation code**.

operand: A value to which an operator is applied.

operating system: A software system that organizes the computer's resources and capabilities and makes them available to the user or to application programs running on the computer.

operation code: The part of a machine-language instruction that specifies the operation to be performed; often called *op code*.

operator: A symbol or sequence of characters, such as + or AND, specifying an operation to be performed on one or more values (the operands) to produce a result.

OR: A logical operator that produces a true result if either or both of its operands are true, a false result if both of its operands are false; compare exclusive OR, AND, NOT.

output: (1) Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem. (2) The act or process of transferring such information.

page: (1) A screenful of information on a video display, consisting on the Apple IIe of 24 lines of 40 or 80 characters each. (2) An area of main memory containing text or graphical information being displayed on the screen. (3) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256 bytes.

page zero: See **zero page**.

parallel interface: An interface in which many bits of information (typically eight bits, or one byte) are transmitted simultaneously over different wires or channels. Compare **serial interface**.

PC board: See **printed-circuit board**.

peripheral: At or outside the boundaries of the computer itself, either physically (as a *peripheral device*) or in a logical sense (as a peripheral card).

peripheral bus: The bus used for transmitting information between the Apple IIe computer and peripheral devices connected to the computer's expansion slots.

peripheral card: A removable printed-circuit board that plugs into one of the Apple IIe's expansion slots and expands or modifies the computer's capabilities by connnecting a peripheral device or performing some subsidiary or peripheral function.

peripheral device: A device, such as a video monitor, disk drive, printer, or modem, used in conjunction with a computer. Often (but not necessarily) physically separate from the computer and connected to it by wires, cables, or some other form of interface, typically by means of a peripheral card.

peripheral slot: See **expansion slot**.

phase: (1) A stage in a periodic process; a point in a cycle; for example, the 6502 microprocessor uses a clock cycle consisting of two phases called $\phi 0$ and $\phi 1$. (2) The relationship between two periodic signals or processes; for example, in NTSC color video, the color of a point on the screen is expressed by the instantaneous phase of the video signal relative to the color reference signal.

pipelining: A feature of a processor that enables it to begin fetching the next instruction before it has finished executing the current instruction. All other things equal, processors that have this feature run faster than those without it.

pointer: An item of information consisting of the memory address of some other item.

pop: To remove the top entry from a stack.

port: The point of connection, usually a physical connector, between a computer and a peripheral device, another computer, or a network.

power supply: The hardware component of a computer that draws electrical power from a power outlet and converts it to the forms needed by other hardware components.

printed-circuit board: A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly fiberglass, to which integrated circuits and other electronic components are connected.

printer: A peripheral device that writes information on paper in a form easily readable by humans or literate monkeys.

processor: The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory.

program: (1) A set of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language. (2) To write a program.

programming language: A set of rules or conventions for writing programs.

prompt: To remind or signal the user that some action is expected, typically by displaying a distinctive symbol, a reminder message, or a menu of choices on the display screen.

prompt character: A text character displayed on the screen to prompt the user for some action. Often also identifies the program or component of the system that is doing the prompting; for example, the prompt character `]` is used by the Applesoft BASIC interpreter, `>` by Integer BASIC, and `*` by the system Monitor program. Also called *prompting character.*

prompt message: A message displayed on the screen to prompt the user for some action. Also called *prompting message.*

push: To add an entry to the top of a stack.

radio-frequency modulator: A device for converting the video signals produced by a computer to a form that can be accepted by a television receiver.

RAM: See **random-access memory**.

random-access memory: Memory in which the contents of individual locations can be referred to in an arbitrary or random order. This term is often used incorrectly to refer to read-write memory, but strictly speaking both read-only and read-write

memory can be accessed in random order. This misuse of the term *random-access* is an attempt to confuse new users, creating a rite of passage and an excellent market for glossaries of computer terms. Compare **read-only memory**, **read-write memory**, **write-only memory**.

raster: The pattern of parallel lines making up the image on a video display screen. The image is produced by controlling the brightness of successive dots on the individual lines of the raster.

read: To transfer information into the computer's memory from a source external to the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

read-only memory: Memory whose contents can be read but not written; used for storing firmware. Information is written into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off, and can never be erased or changed. Compare **read-write memory**, **random-access memory**, **write-only memory**.

read-write memory: Memory whose contents can both be read and written; often misleadingly called *random-access memory*, or *RAM*. The information contained in read-write memory is erased when the computer's power is turned off, and is permanently lost unless it has been saved on a more permanent storage medium, such as a disk. Compare **read-only memory**, **random-access memory**, **write-only memory**.

real number: A number that may include a fractional part; represented inside the computer in floating-point form. Compare **integer**.

register: A location in a computer processor where an item of information, such as a byte, is held and modified under program control. Registers in the 6502 microprocessor include the accumulator (A), two index registers (X and Y), the stack pointer (S), the processor status register (P), and the program counter (PC). The PC register holds two bytes (sixteen bits); the other registers hold one byte (eight bits) each.

return address: The point in a program to which control returns on completion of a subroutine.

RF modulator: See **radio-frequency modulator**.

rigid disk: A disk made of a hard, nonflexible material. Compare **flexible disk**.

ROM: See **read-only memory**.

routine: A part of a program that accomplishes some task subordinate to the overall task of the program.

run: (1) To execute a program. (2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

save: To transfer information from main memory to a peripheral storage medium for later use.

screen: See **display screen**.

scroll: To change the contents of all or part of the display screen by shifting information out at one end (most often the top) to make room for new information appearing at the other end (most often the bottom), producing an effect like that of moving a scroll of paper past a fixed viewing window. See **viewport**, **window**.

serial interface: An interface in which information is transmitted sequentially, one bit at a time, over a single wire or channel. Compare **parallel interface**.

setup time: The amount of time a signal must be valid in advance of some event; compare **hold time**.

silicon: A non-metallic, semiconducting chemical element from which integrated circuits are made. Not to be confused with *silica* — that is, silicon dioxide, such as quartz, opal, or sand — or with *silicone*, any of a group of organic compounds containing silicon.

soft switch: A means of changing some feature of the Apple IIe from within a program; specifically, a location in memory that produces some special effect whenever its contents are read or written.

software: Those components of a computer system consisting of programs that determine or control the behavior of the computer. Compare **hardware**, **firmware**.

source code: See **source program**.

source program: The original form of a program given to a language translator such as a compiler or assembler for conversion into another form; sometimes called *source code*. Compare **object program**.

space character: A text character whose printed representation is a blank space, typed from the keyboard by pressing the SPACE bar.

stack: A list in which entries are added or removed at one end only (the top of the stack), causing them to be removed in LIFO (last-in-first-out) order.

string: An item of information consisting of a sequence of text characters.

strobe: (1) An event, such as a change in a signal, that triggers some action. (2) A signal whose change is used to trigger some action.

subroutine: A part of a program that can be executed on request from any point in the program, and which returns control to the point of the request on completion.

system: A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose.

television receiver: A display device capable of receiving broadcast video signals (such as commercial television) by means of an antenna. Can be used in combination with a radio-frequency modulator as a display device for the Apple IIe computer. Compare **video monitor**.

television set: See **television receiver**.

terminal: A device consisting of a typewriterlike keyboard and a display device, used for communicating between a computer system and a human user. Personal computers such as the Apple IIe typically have all or part of a terminal built into them.

text: (1) Information presented in the form of characters readable by humans. (2) The display of characters on the Apple IIe's display screen. Compare **graphics**.

text window: An area on the Apple IIe's display screen within which text is displayed and scrolled.

transistor-transistor logic: (1) A family of integrated circuits used in computers and related devices. (2) A standard for interconnecting such circuits that defines the voltages used to represent logical zeros and ones.

troubleshoot: To locate and correct the cause of a problem or malfunction in a computer system. Typically used to refer to hardware-related problems; compare **debug**.

TTL: See **transistor-transistor logic**.

unary operator: An operator that applies to a single operand; for example, the minus sign (-) in a negative number such as -6 is a unary arithmetic operator. Compare **binary operator**.

user: The person operating or controlling a computer system.

user interface: The rules and conventions by which a computer system communicates with the person operating it.

vector: (1) The starting address of a program segment, when used as a common point for transferring control from other programs. (2) A memory location used to hold a vector, or the address of such a location.

video: (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form. (3) An early space pioneer.

video monitor: A display device capable of receiving video signals by direct connection only, and which cannot receive broadcast signals such as commercial television. Can be connected directly to the Apple IIe computer as a display device. Compare **television receiver**.

viewport: All or part of the display screen, used by an application program to display a portion of the information (such as a document, picture, or worksheet) that the program is working on. Compare **window**.

warm start: The process of restarting the Apple IIe after the power is already on, without reloading the operating system into main memory and often without losing the program or information already in main memory. Compare **cold start**.

window: (1) The portion of a collection of information (such as a document, picture, or worksheet) that is visible in a viewport on the display screen; compare viewport. (2) A viewport. (3) A flat, rectangular panel, usually made of silica, used in many archaic structures as a human-to-nature interface.

word: A group of bits of a fixed size that is treated as a unit; the number of bits in a word is a characteristic of each particular computer.

wraparound: The automatic continuation of text from the end of one line to the beginning of the next, as on the display screen or a printer.

write: To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

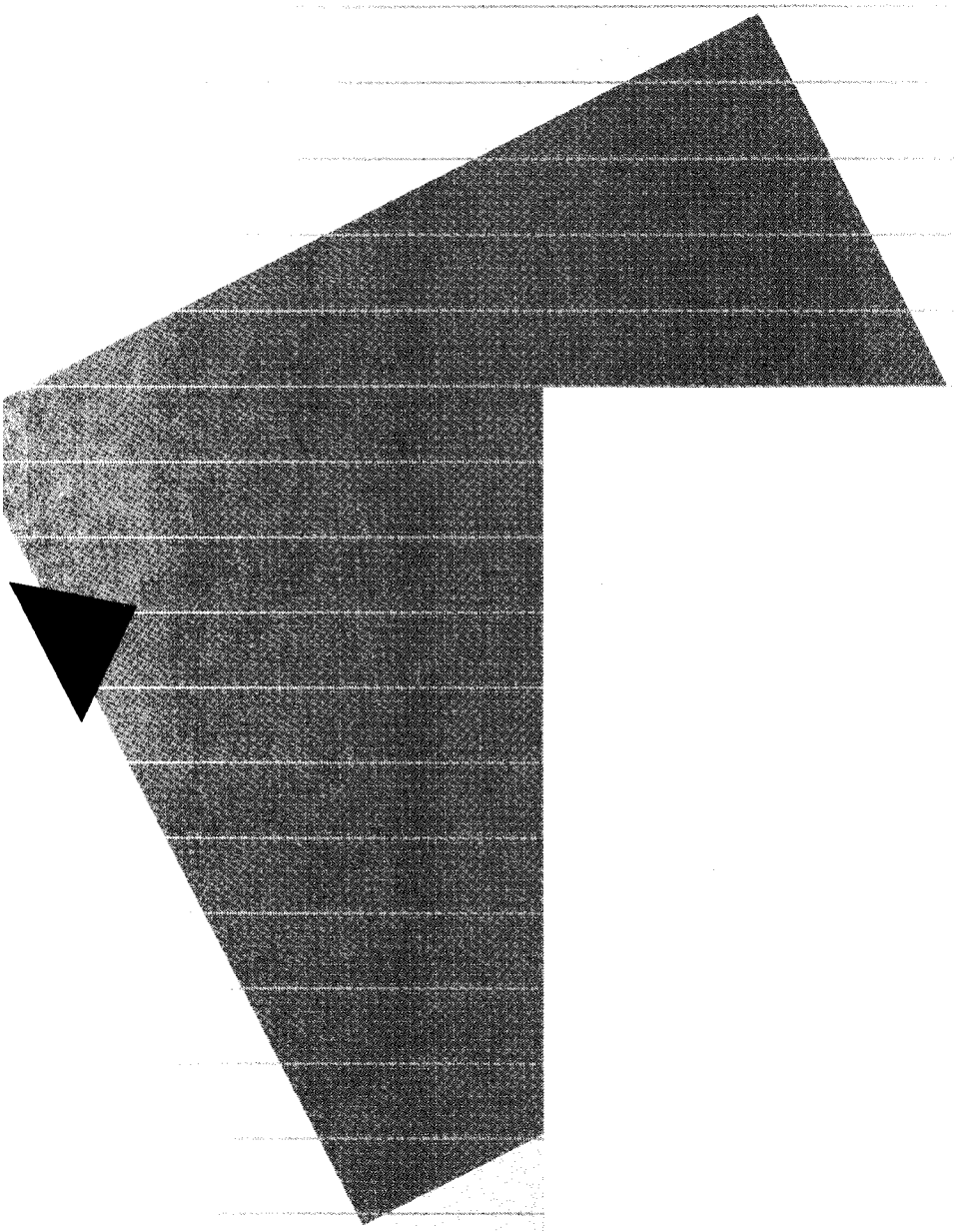
write-only memory: A form of computer memory into which information can be stored but never, ever retrieved, developed under government contract in 1975 by Professor Homberg T. Farnsfarfle. Farnsfarfle's original prototype, approximately one inch on each side, has so far been used to store more than 100 trillion words of surplus federal information. Farnsfarfle's critics have denounced his project as a six-million-dollar boondoggle, but his defenders point out that this excess information would have cost more than 250 billion dollars to store in conventional media. Compare **read-only memory**, **read-write memory**, **random-access memory**.

X **register:** One of the index registers in the 6502 microprocessor.

Y **register:** One of the index registers in the 6502 microprocessor.

zero page: The first page (256 bytes) of the Apple IIe's memory, also called *page zero*. Since the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.

BLANK PAGE



Bibliography

Apple Computer, Inc.: *Apple IIe Applesoft Reference Manual;* Apple Computer Inc., 1982, Cupertino, CA. Apple product number A2L2004

— *Apple IIe Applesoft Tutorial Manual;* Apple Computer Inc., 1982, Cupertino, CA. Apple product number A2L2003

— *Apple II BASIC Programming Manual;* Apple Computer Inc., 1978, Cupertino, CA. Apple product number A2L0005

— *Apple II Monitors Peeled;* Apple Computer Inc., 1978, Cupertino, California. Apple product number D2L0013

— *Apple IIe Owner's Manual;* Apple Computer Inc., 1982, Cupertino, CA. Apple product number A2L2001

— *Programmer's Aid #1 Installation and Operating Manual;* Apple Computer Inc., 1978, Cupertino, California. Apple product number A2L0011

Leventhal, Lance: *6502 Assembly Language Programming;* Osborne/McGraw-Hill, 1979, Berkeley, CA.

Synertek, Incorporated: *Hardware Manual*; Synertek Incorporated, 1976, Santa Clara, CA. Apple product number A2L0002

— *Programming Manual;* Synertek Incorporated, 1976, Santa Clara, CA. Apple product number A2L0003

Watson, Allen, III: "More Colors for Your Apple"; *Byte*, Vol. 4, No. 6, June, 1979. Byte Publications, Inc, Peterborough, NH

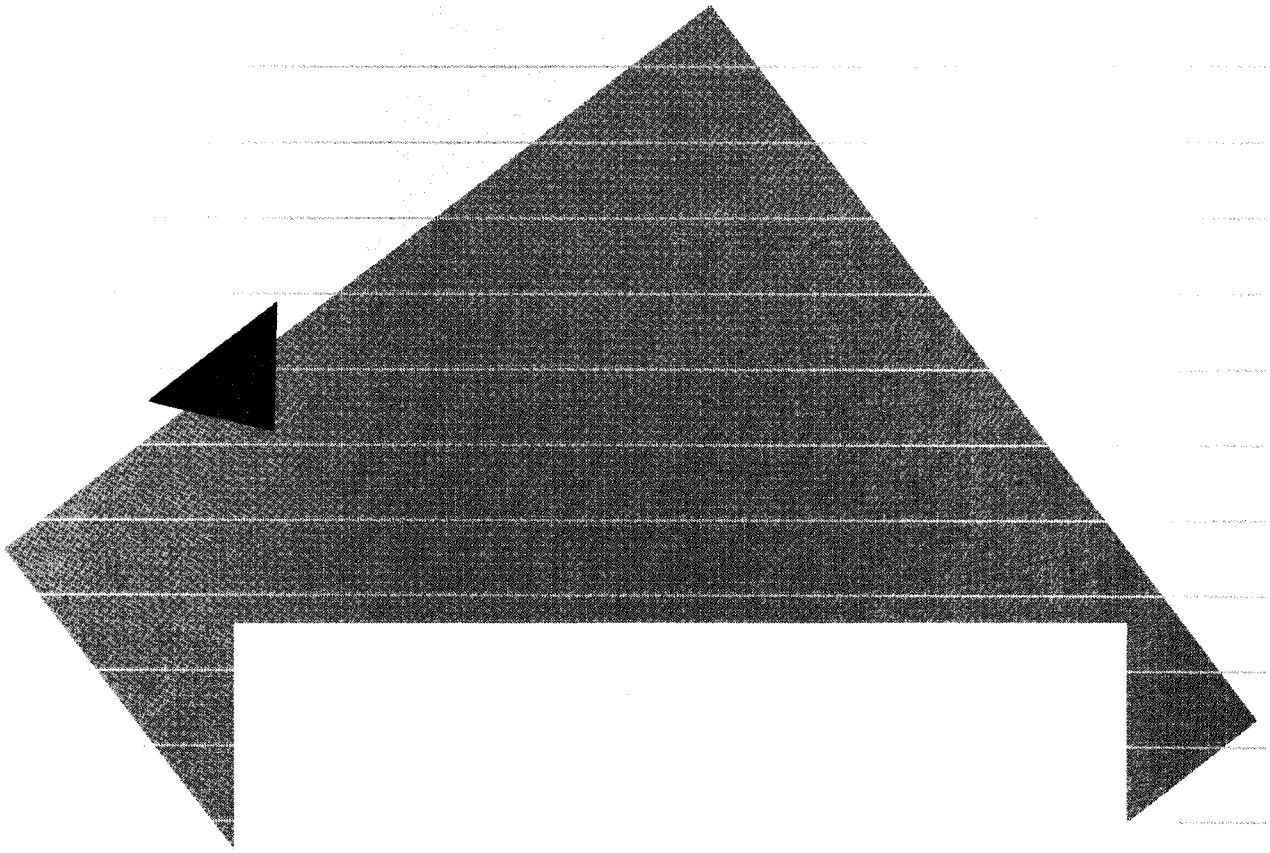
— "Simplified Theory of Video Graphics, Part I"; *Byte*, Vol. 5, No. 11, November, 1980. Byte Publications, Inc, Peterborough, NH

— "Simplified Theory of Video Graphics, Part II"; *Byte*, Vol. 5, No. 12, December, 1980. Byte Publications, Inc. Peterborough, NH

Wozniak, Steve: "System Description: The Apple II"; *Byte*, Vol. 2, No. 5, May, 1977. Byte Publications, Inc. Peterborough, NH

— "SWEET16: The 6502 Dream Machine"; *Byte*, Vol. 2, No. 10, October, 1977. Byte Publications, Inc, Peterborough, NH

BLANK PAGE

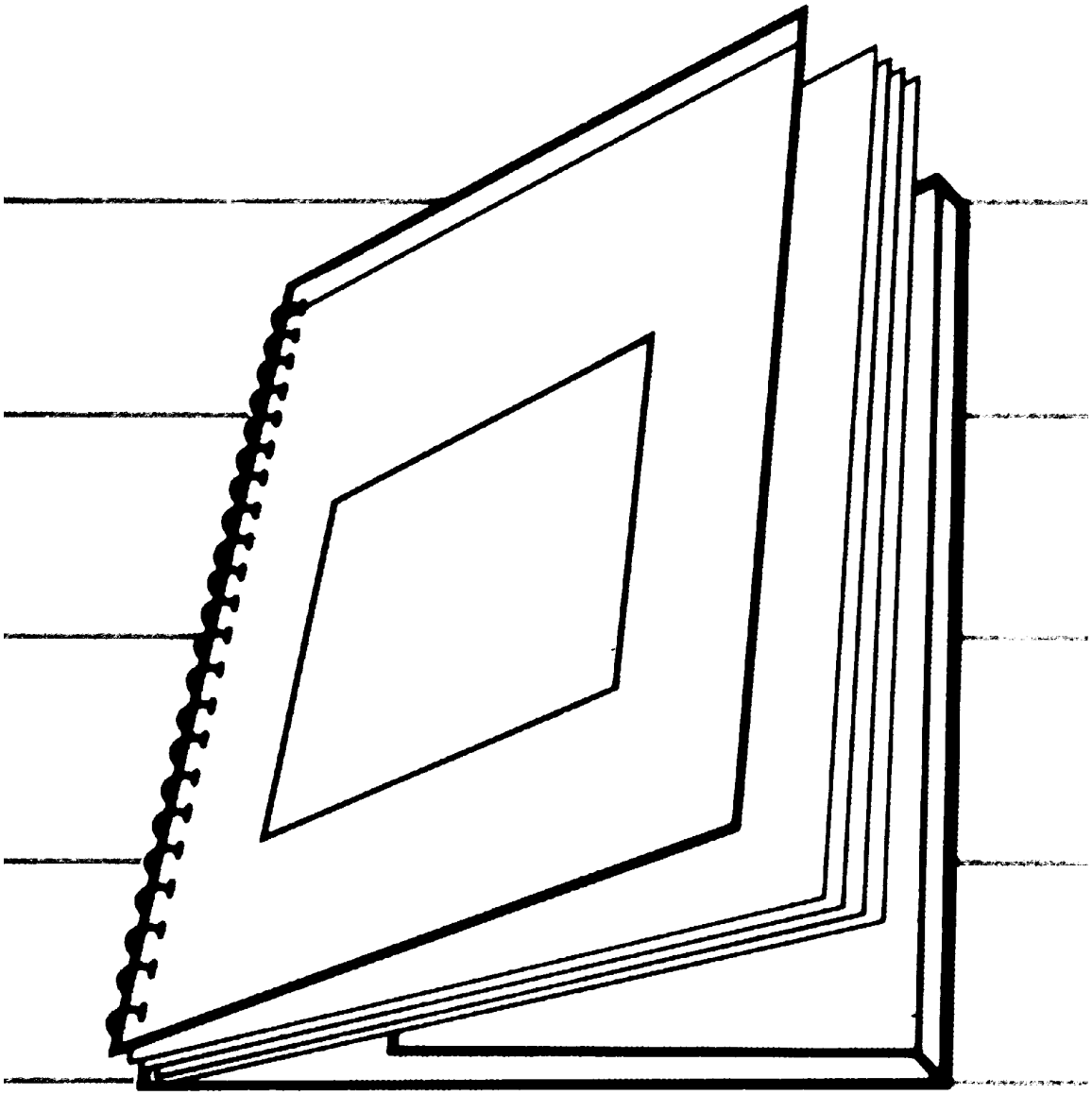


Index

D

030-0357-B 1982

Tuck end flap
inside back cover
when using manual.



"A2_030-0357-B_1982_9-1.pict" 134 KB 2002-10-27 dpi: 1000h x 1000v pix: 1866h x 2848v

"A2_030-0357-B_1982_9-2.pict" 2206 KB 2002-10-27 dpi: 600h x 600v pix: 300h x 2494v

030-0357-B 1982