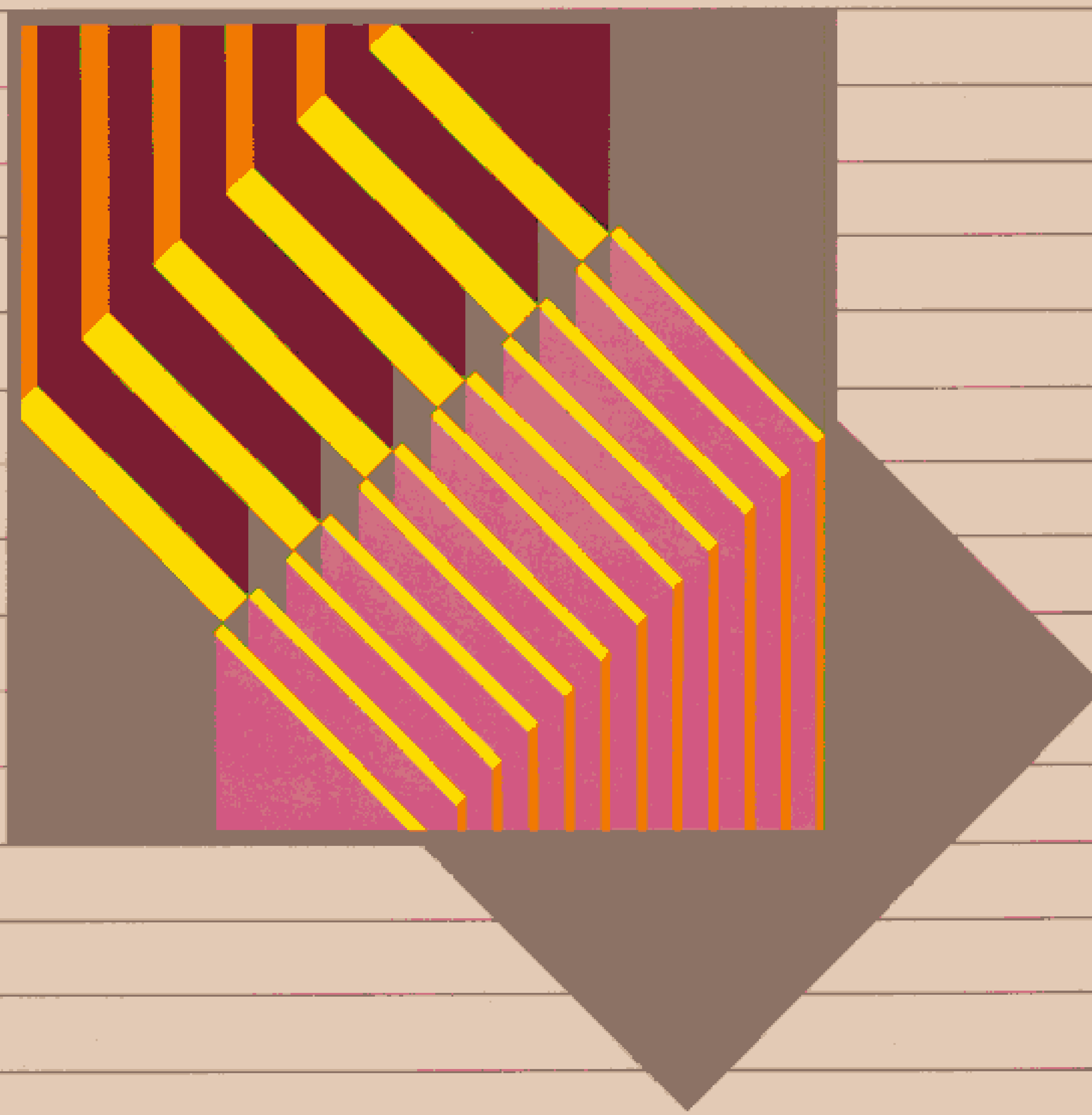
Apple II

Extended 80-Column Text Card Supplement

For *IIe* Only



Limited Warranty on Media and Replacement

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program.

While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for Program coverage and details. In some countries the replacement period may be different; check with your authorized Apple dealer.

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

Even though Apple has tested the software and reviewed the documentation, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD "AS IS," AND YOU THE PURCHASER ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, even if advised of the possibility of such damages. In particular, Apple shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering such programs or data.

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

Some states do 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. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Warning

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

Reorder Apple Product #A2L2007

Apple II

Extended 80-Column Text Card Supplement

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.

Table of Contents

Who Needs To Read This Supplement?

This supplement comes with the Apple IIe Extended 80-Column Text Card and describes the added features it has, compared to the 80-Column Text Card. Before reading this supplement, you should read the *Apple IIe 80-Column Text Card Manual.*

There are two ways you are likely to use the extended version of the 80-Column Text Card:

- As a user with application programs that take advantage of the extra memory on the card to give you more features or more storage for your data.
- As a developer creating a program, for yourself or for others, that will use the extra storage the extended card provides.

Users: A Card Is a Card

From the user's point of view, the Extended 80-Column Text Card is just like the standard 80-Column Text Card. Oh, it's a little bigger, and it costs more, but the technical differences between the two kinds of text cards are mostly hidden by software. Read Chapter 1 of this supplement for an introduction to the Apple IIe 80-Column Extended Text Card.

The extended text card is installed the same way as the standard 80-column card: read the *Apple IIe 80-Column Text Card Manual* for directions.

Most application programs run the same with either card—in fact, many of them don't even take advantage of the extra memory on the extended card; they simply use it to display 80 columns of text. Programs that do use the extra memory may do so automatically, without any action on your part, or they may let you select optional features or data storage. To find out how to use those programs with the extra memory, refer to their instruction manuals.

In short, if you just want to use this card for displaying 80 columns of text, and you aren't developing a program that uses the auxiliary memory, all you really need to know can be found in the *Apple IIe 80-Column Text Card Manual* and in the instructions for your application programs.

Developers: How To Use the Auxiliary Memory

The only difference between the Extended 80-Column Text Card and the standard 80-Column Text Card is the amount of memory they contain. The extended card has 64K bytes of auxiliary memory, while the standard card has only the additional 1K bytes necessary to display 80 columns of text on an Apple IIe.

The main purpose of this supplement is to provide you with enough information to use the auxiliary memory in your programs. Normally, programs used with the Apple IIe can only work with the 64K bytes of built-in main memory. To work with the auxiliary memory, a program must set special switches in the Apple IIe that substitute auxiliary memory for main memory. Neither DOS 3.3 nor Pascal 1.1—system programs for the Apple II—support this memory substitution, so for now your application programs have to handle it themselves.

Contents of This Supplement

This supplement contains the information you need to use the auxiliary memory for storing programs and data. Chapter 1 is a general introduction; it describes the functions of the Extended 80-Column Text Card.

Chapter 2 is a general description of the design of the Extended 80-Column Text Card; it explains how the card works with the Apple IIe hardware.

Chapter 3 contains directions for using the auxiliary memory with your programs. Most of the information in Chapter 3 is adapted from the *Apple IIe Reference Manual*. The reference manual is your main source of information about the internal operation of the Apple IIe.

Chapter 4 contains short programs that use the auxiliary memory. These examples are functional, but not general: you will probably want to modify them for use in the programs you write.

Symbols Used in This Supplement

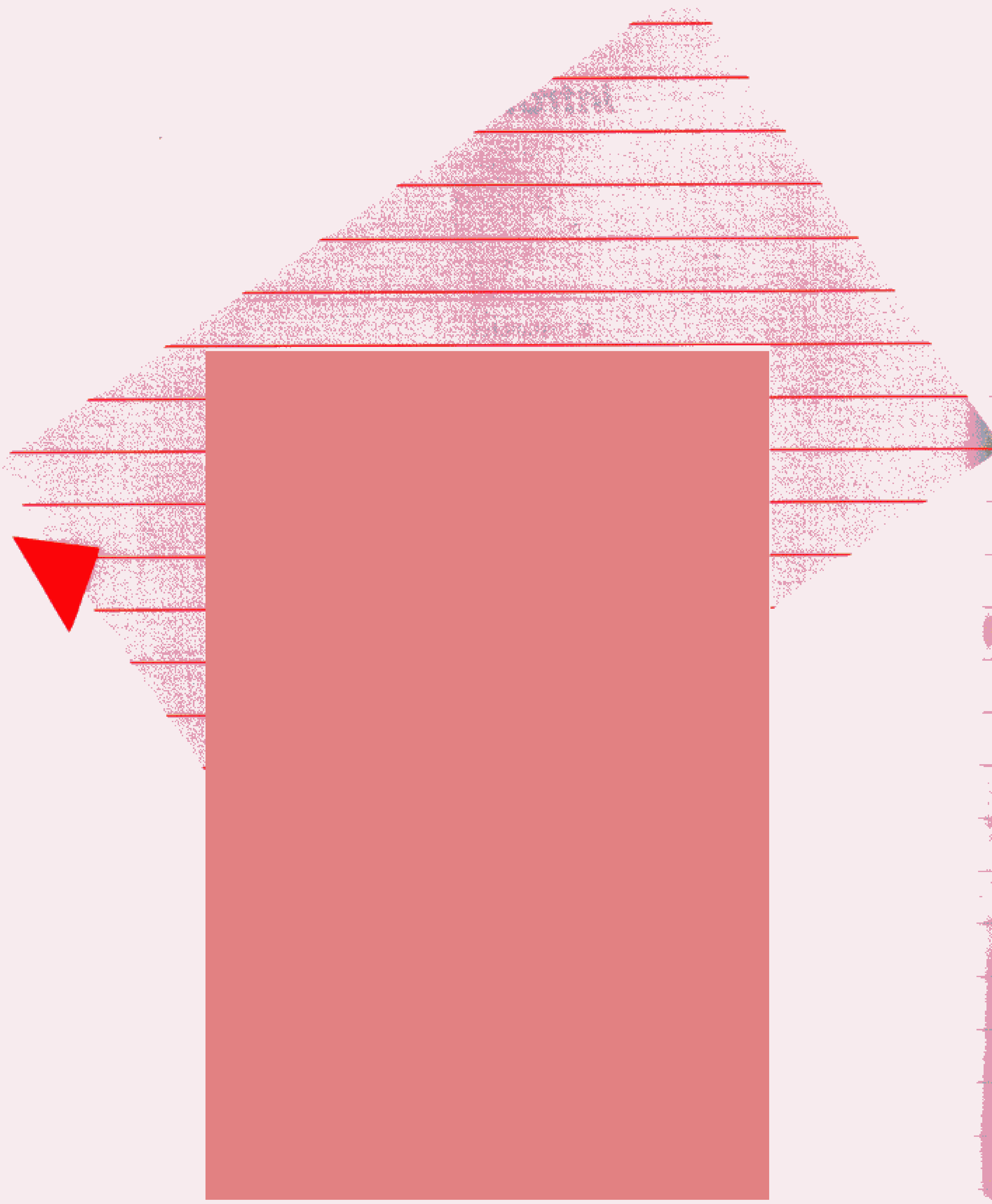
Special text in this manual is set off in different ways, as shown in these examples.

Warning
Important warnings appear in boxes like this.

Reminder: Information that is only incidental to the text appears in gray boxes like this. You may want to skip over such boxes and return to them later.

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

Introduction



Introduction

The design of the Apple IIe Extended 80-Column Text Card is the same as that of the standard Apple IIe 80-Column Text Card. The only difference is that the extended text card contains 64K bytes of auxiliary memory (programmable memory or RAM) while the standard card contains only 1K byte of RAM. The 80-column display requires only 1K byte of auxiliary memory, so it will work with either card. The firmware that supports the special features associated with the 80-column display is part of the Apple IIe itself, and works the same regardless of which card is present.

Installation

Installing the Extended 80-Column Text Card is easy: do it just the way you install the standard 80-Column Text Card. Either card fits into the auxiliary slot (labeled AUX. CONNECTOR) on the main logic board inside the Apple IIe. If you haven't installed the card yet, follow the directions given in the *Apple IIe 80-Column Text Card Manual*.

Warning
Never install or remove anything inside the Apple IIe with the power on. There is a small red lamp—an LED—toward the back of the main circuit board to remind you of this; if the red lamp is on, turn off the power before you do anything inside the Apple IIe.

80-Column Features

The built-in firmware that supports the 80-column display has other features in addition to the wider display. The *Apple IIe 80-Column Text Card Manual* tells you how to activate the built-in firmware and the 80-column display. That manual also describes many of the Apple IIe's features.

You can find more information about the Apple IIe in the *Apple IIe Reference Manual*. Chapter 2 includes a description of the different display modes and how to select them. Chapter 3 includes tables of the functions of the escape sequences and control keys in the Apple IIe.

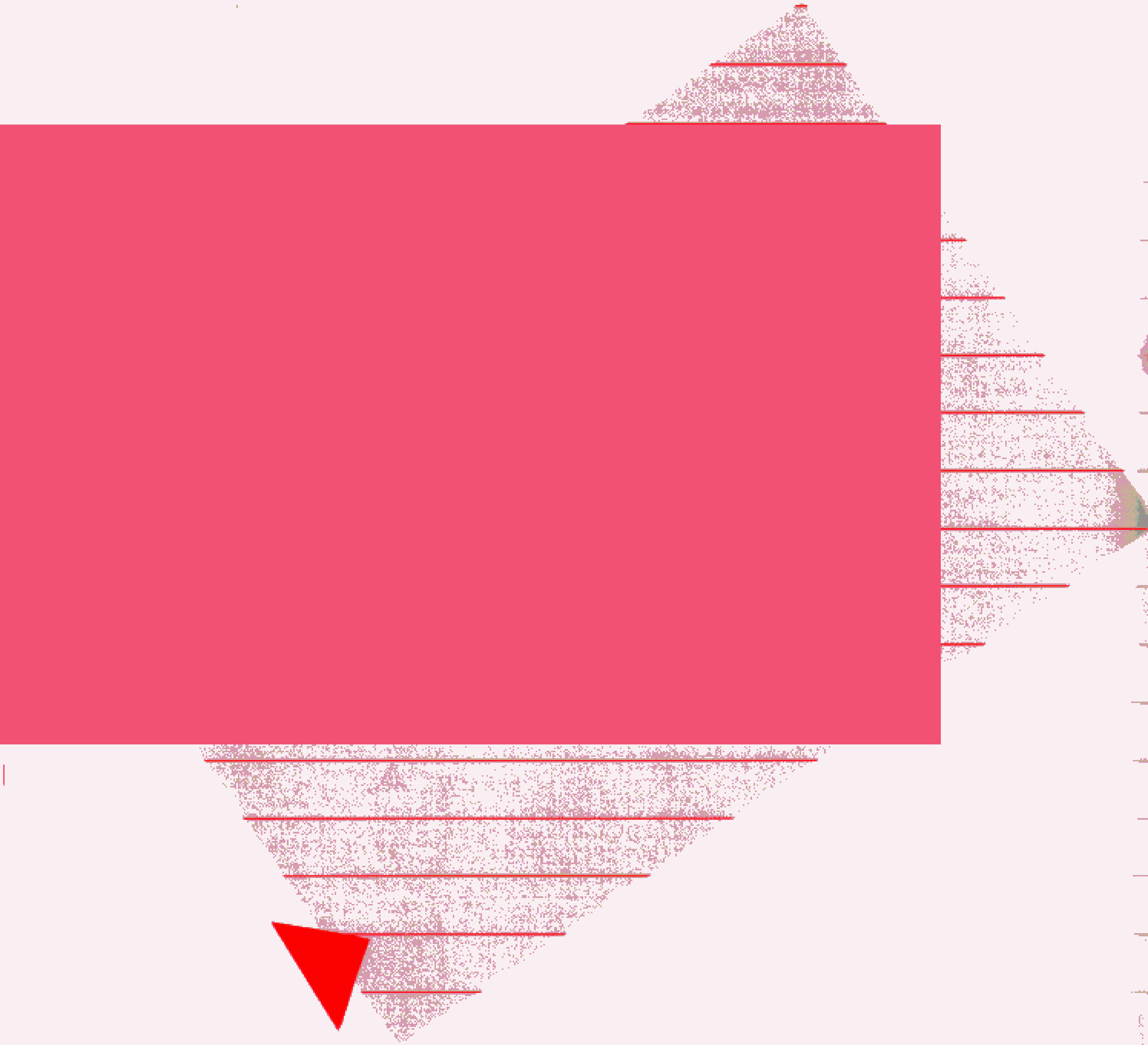
About the Auxiliary Memory

The Extended 80-Column Text Card has 64K bytes of additional RAM, usually referred to as auxiliary memory. 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 for auxiliary program and data storage. If you use only 40 columns for text display, all 64K bytes are available for programs and data.

The processor in the Apple IIe can only address 64K bytes of memory. The computer has special circuits that programs can switch to access auxiliary memory in place of main memory. At any one time, locations in the same 64K address space are in either main memory or auxiliary memory. In other words, even though an Apple IIe with an Extended 80-Column Text Card has a total of 128K bytes of programmable memory, it is not appropriate to call it an 128K-byte system. Rather, there are 64K bytes of auxiliary memory that can be swapped for main memory under program control.

Warning
Careless switching to the auxiliary memory is almost certain to crash your programs. If you want to use auxiliary memory in your own programs, be sure to study the rest of this supplement and the relevant information in the *Apple IIe Reference Manual*.

How the Auxiliary Memory Works



How the Auxiliary Memory Works

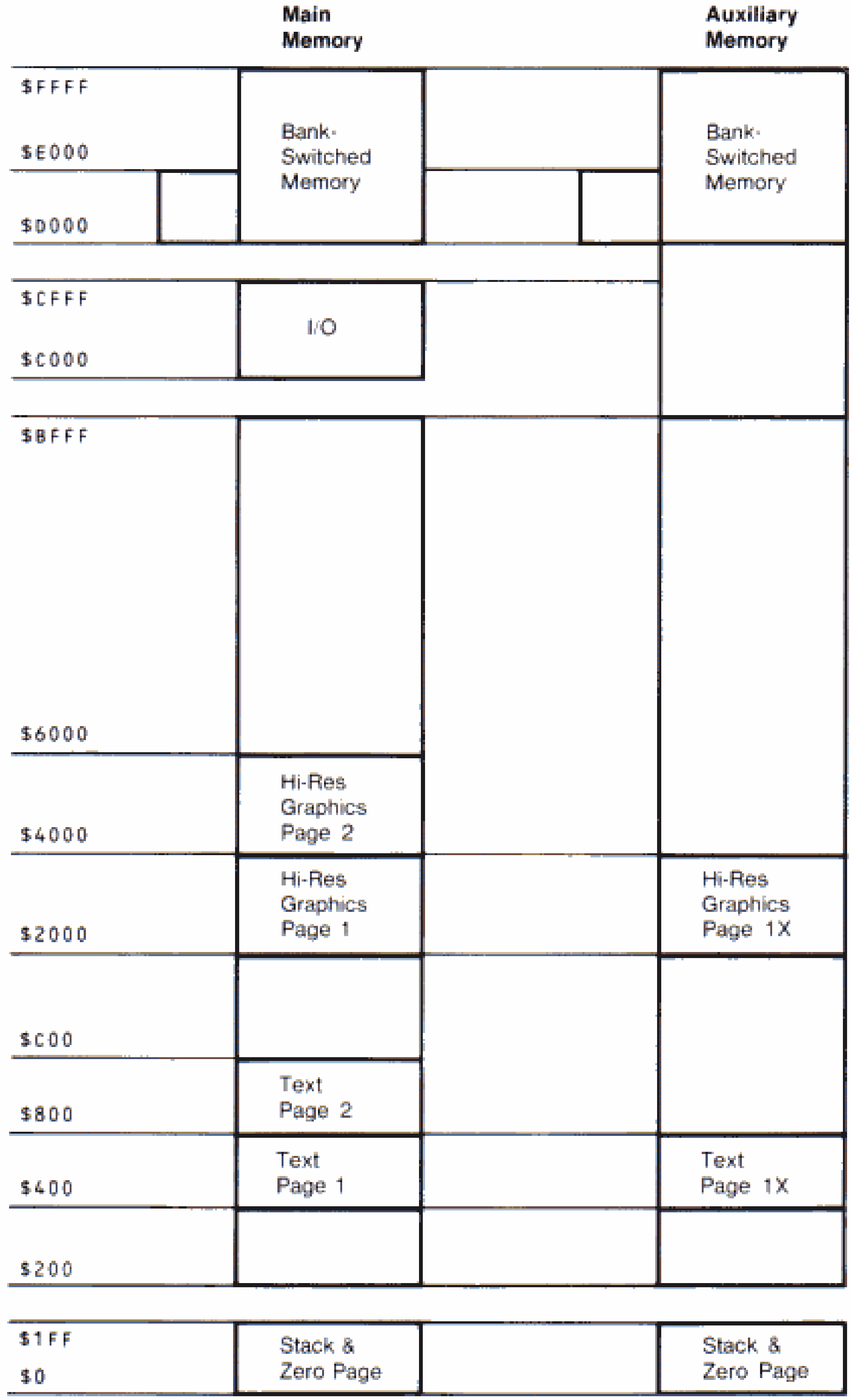
This chapter briefly outlines how the auxiliary memory operates. It will help you understand what happens when you use the auxiliary memory in your programs.

Addressing the Auxiliary Memory

The 6502 microprocessor can address 64K bytes of memory. In the Apple IIe the microprocessor's entire 64K memory space is taken up by main RAM (random-access memory), ROM (read-only memory), and I/O (input/output); there's no memory space available for the added memory on the extended text card. Instead, the address bus is connected to the auxiliary memory in parallel with the main memory. To use the auxiliary memory for program and data storage, the Apple IIe switches its data bus so that it reads and writes to the memory on the card instead of the main memory. To use the auxiliary memory to expand the display, the Apple IIe fetches data both from main memory and from auxiliary memory, as described in the section "How the 80-Column Display Works."

The bus switching for program and data storage is controlled by the Memory Management Unit (MMU), a custom integrated circuit designed for the Apple IIe (see Chapter 7 of the *Apple IIe Reference Manual*). The MMU contains the soft switches set by your programs along with the logic circuitry to monitor the address bus and to switch to auxiliary memory for the selected address ranges.

Figure 2-1. Memory Map with Auxiliary Memory



As you can see by studying the memory map in Figure 2-1, the auxiliary memory is divided into two large sections and one small one. The largest section is substituted for main memory addresses 512 to 49151 ($200 through $BFFF). This part of memory is sometimes referred to as the 48K memory space, and it is used for storing programs and data.

The other large section of auxiliary memory replaces main memory addresses 52K to 64K ($D000 through $FFFF). This memory space is called the bank-switched memory. If you plan to use this part of the auxiliary memory, read the section "Bank-switched Memory" in the *Apple IIe Reference Manual*. The switching for the ROM and the $D000 bank is independent of the auxiliary-RAM switching, so the bank switches have the same effect on the auxiliary RAM that they do on the main RAM.

When you switch to the auxiliary memory in the bank-switched memory space, you also get the first two pages of auxiliary memory, 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.

Warning
Remember that addresses in page zero and the 6502 stack switch to auxiliary memory any time you switch the bank-switched memory to auxiliary memory.

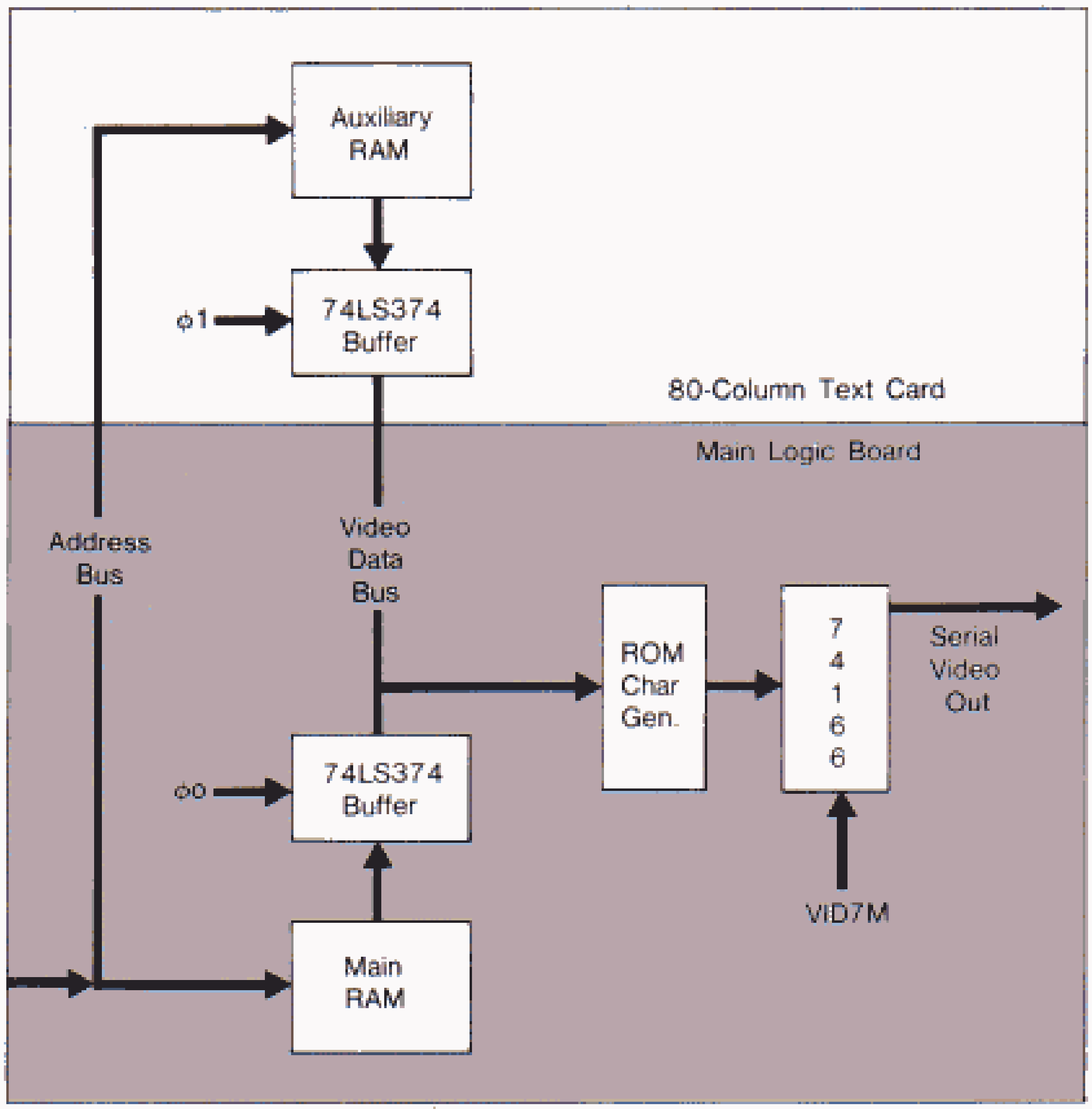
How the 80-Column Display Works

Half of the data for the 80-column display is stored in main memory in the normal text Page 1, and the other half is stored in auxiliary memory on the extended text card. The display circuitry fetches bytes of data from these two memory areas simultaneously and displays them as two adjacent characters.

Memory pages are 256 bytes long, but **display pages** are either 1024 bytes, e.g., text Page 1, or 8192 bytes, e.g., high-resolution graphics Page 1. See Chapters 2 and 4 of the *Apple IIe Reference Manual*.

The main memory and the auxiliary memory are connected to the address bus in parallel, so both are activated during the display cycle. The 40-column display uses every other clock cycle and fetches data only from main memory. The 80-column display uses the remaining clock cycles to process the additional display data from auxiliary memory.

Figure 2-2. Fetching Data for the 80-Column Display



The byte of display data from main memory goes to a buffer on the main logic board, and the display data from auxiliary memory goes to a buffer on the extended text card. When the 80-column display is on, the data bytes from these buffers are switched onto the video data bus on alternate clock cycles: first the byte from the auxiliary memory, then the byte from the main memory. The main memory provides the characters displayed in the odd columns of the display, and the auxiliary memory provides the characters in the even columns.

The 80-column display contains twice as many characters as the 40-column display does, so it has to put twice as many dots across the screen. This means that the dots are clocked out at 14MHz instead of 7MHz, making them narrower and therefore dimmer on a normal video monitor. On a television set, the dot patterns making up the characters are too close together to reproduce clearly. To produce a satisfactory 80-column display requires a monitor with a bandwidth of at least 14MHz.

RGB stands for red, green, and blue, and identifies a type of color monitor that uses independent inputs for the three primary colors.

Except for some expensive RGB-type color monitors, any video monitor with a bandwidth as high as 14MHz will be a monochrome monitor. Monochrome means one color: a monochrome video monitor can have a screen color of white, green, orange, or any other single color.

Note that this simultaneous-then-sequential fetching applies only to the video-display generation; reading and writing for data storage in auxiliary memory is done by switching the data bus to read only from the card, as described in the previous section. For more information about the way the Apple IIe handles its display memory, refer to Chapter 2 and Chapter 7 of the *Apple IIe Reference Manual*.

Double High-Resolution Graphics

When you select mixed-mode graphics with 80-column text, you would expect that the doubling of the data rate that produces the 80-column display would change the high-resolution graphics from 280 to 560 dots horizontally and cause the low-resolution graphics to malfunction. To prevent this, the logic that controls the display includes an extra circuit to force the graphics displays to be the same regardless of whether you have set the soft switches for 80-column text or for 40-column text. This feature is included so that you can use 80-column text in the mixed graphics and text modes.

For those who would like to have a graphics display with twice the horizontal resolution, there is a way to disable the circuit that forces normal graphics timing with 80-column text. There are two things you must do to obtain the double high-resolution display:

- Install a jumper to connect the two Molex-type pins on the Extended 80-Column Text Card.
- Turn on the Annunciator 3 soft switch along with the switches that select the 80-column display and high-resolution graphics.

This procedure works only on the Apple IIe with the Rev B (and later) main logic board, identified by a B as the last letter of the part number on the back part of the board. Connecting the pins on the Extended 80-Column Text Card completes a connection between pin 50 (AN3) and pin 55 (FRCTXT') on the auxiliary slot.

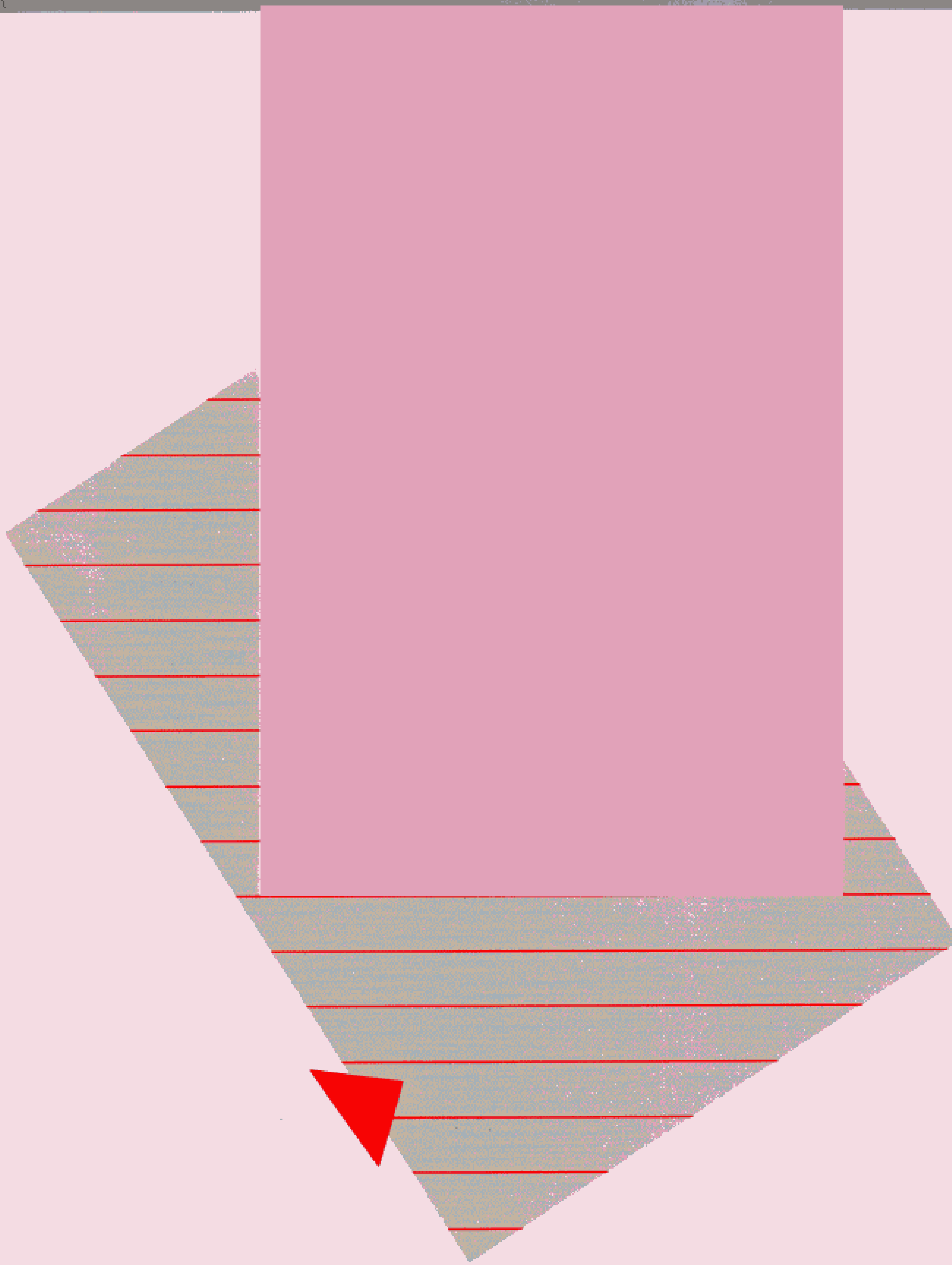
Warning
If you have a Rev A Apple IIe, using an extended text card with a jumper makes the computer inoperable. You cannot use the double high-resolution modification with a Rev A Apple IIe.

If you have an extended text card with a jumper installed in a Rev B (or later) Apple IIe, turning on Annunciator 3 and selecting high-resolution graphics and 80-column text at the same time generates a display using high-resolution Page 1 addresses in main memory and auxiliary memory at the same time.

The memory mapping for this graphics display is doubled by columns the same way as 80-column text, but it uses high-resolution graphics Page 1 instead of text Page 1. Where the 80-column text mode displays pairs of data bytes as pairs of characters, double high-resolution mode displays pairs of data bytes as 14 adjacent dots, seven from each byte. As in 80-column text mode, there are twice as many dots across the display screen, so the dots are only half as wide.

Existing Apple II graphics programs do not support this kind of display. Until new programs become available, you'll have to write your own plotting routines if you want to use 560-dot graphics.

How To Use the Auxiliary Memory



How To Use the Auxiliary Memory

This chapter describes soft switches and built-in subroutines that control the operation of the auxiliary memory. To take advantage of the additional memory, you must set up your programs to operate in one part of memory while they switch the other part between main and auxiliary RAM. Your program can perform the memory switching by means of the soft switches described in the section "Display Mode Switching" or by using the `AUXMOVE` and `XFER` subroutines described later in this chapter. Except for these subroutines, most existing Apple II system software (DOS 3.3, Pascal 1.1) doesn't support the auxiliary memory.

Although some high-level languages, such as BASIC, can set the soft switches directly, your programs must use assembly-language subroutines to control the auxiliary memory. Small assembly-language subroutines can be accessed from a BASIC program using a `CALL` statement, or they can be linked to a Pascal program as procedures or functions: see the examples in Chapter 4.

Warning
Do not attempt to use the auxiliary memory directly from a program in an interpreter language such as BASIC or Pascal. The interpreters that run such programs use several areas in main memory, including the stack and the zero page. If you switch to auxiliary memory in these pages, the interpreter crashes. When you reset the system to start over, your program and data are lost.

The Extended Display

The primary purpose of an 80-column text card is the generation of an 80-column display, so there is a complete set of switches just to control the display. Other switches are used for program and data storage in the auxiliary memory; they are described later.

Display Pages

The Apple IIe generates its video displays from data stored in specific areas in memory called display pages. The 40-column-text and low-resolution-graphics modes use text Page 1 and text Page 2, located at 1024-2047 (hexadecimal $400-$7FF) and 2048-3071 ($800-$BFF) in main memory.

The 80-column text display uses a combination of text Page 1 in main memory and the same page in the auxiliary memory, here called Page 1X. Text Page 1X occupies the same address space as text Page 1, but in auxiliary memory rather than main memory. To store data in Page 1X, you must use a soft switch (see the section "Display Mode Switching"). The built-in 80-column display routines described in Chapter 3 of the *Apple IIe Reference Manual* take care of this switching automatically; that is a good reason to use those routines for all your normal 80-column text output.

Table 3-1. Video Display Page Locations. ***Note:** These modes use 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		Notes
40-Column Text, Low-Resolution Graphics	1	$400	1024	$7FF	2047	
	2	$800	2048	$BFF	3071	
80-Column Text	1	$400	1024	$7FF	2047	*
Normal 280-Dot High-Resolution Graphics	1	$2000	8192	$3FFF	16383	
	2	$4000	16384	$5FFF	24575	
Optional 560-Dot High-Resolution Graphics	1	$2000	8192	$3FFF	16383	*

Display Mode Switching

You select the display mode that is appropriate for your application by reading or writing to soft switches. Most soft switches have three memory locations: one for turning the switch on, one for turning it off, and one for reading the state of the switch.

Table 3-2 shows the locations of the soft switches that control the display modes. 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.

For information about the **keyboard data** and **strobe functions**, see Chapter 2 of the *Apple IIe Reference Manual.*

Some of the soft switches in Table 3-2 are marked read or write. Those soft switches share their locations with the keyboard data and strobe functions. To perform the function shown in the table, use only 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 switch function occurs when you address the location, and the value is ignored.

Warning

Be sure to use only the indicated operations to manipulate the switches. If you read from a switch marked write, you won't get the correct data. If you write to a switch marked read, you won't set the switch you wanted, and you may change some other switch so as to cause your program to malfunction.

When you read a soft switch, you get a byte with the state of the switch in bit 7, the high-order bit. The other bits in the byte are unpredictable. If you are programming in machine language, this bit is the sign bit. 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.

Table 3-2. Display Soft Switches. (1) This mode is only effective when TEXT switch is off. (2) This switch has a different function when 80STORE is on: refer to the next section. (3) This switch changes the function of the PAGE2 switch for addressing the display memory on the extended text card: refer to the next section.

Name	Function	Location Hex	Decimal		Notes
TEXT	On: Display Text	$C051	49233	-16303	
	Off: Display Graphics	$C050	49232	-16304	
	Read TEXT Switch	$C01A	49178	-16358	Read
MIXED	On: Text With Graphics	$C053	49235	-16301	1
	Off: Full Graphics	$C052	49234	-16302	1
	Read MIXED Switch	$C01B	49179	-16357	Read
PAGE2	On: Display Page 2	$C055	49237	-16299	2
	Off: Display Page 1	$C054	49236	-16300	2
	Read PAGE2 Switch	$C01C	49180	-16356	Read
HIRES	On: Graphics = High-Resolution	$C057	49239	-16297	1
	Off: Graphics = Low-Resolution	$C056	49238	-16298	1
	Read HIRES Switch	$C01D	49181	-16355	Read
80COL	On: Display 80 Columns	$C00D	49165	-16371	Write
	Off: Display 40 Columns	$C00C	49164	-16372	Write
	Read 80COL Switch	$C01F	49183	-16353	Read
80STORE	On: Store in Auxiliary Page	$C001	49153	-16383	Write,3
	Off: Store in Main Page	$C000	49152	-16384	Write,3
	Read 80STORE Switch	$C018	49176	-16360	Read

Addressing the 80-Column Display Directly

Figure 3-1 is the map of the 80-column display. Half of the data is stored in text Page 1 in main memory, and the other half is stored in the same locations in auxiliary memory (here called Page 1X). The display circuitry fetches bytes from these two memory areas simultaneously and displays them sequentially: first the byte from the auxiliary memory, then the byte from the main memory. The main memory stores the characters in the odd columns of the display, and the auxiliary memory stores the characters in the even columns. For a full description of the way the Apple IIe handles its display memory, refer to Chapter 2 and Chapter 7 of the *Apple IIe Reference Manual*.

To store data directly into the display page on the Extended 80-Column Text Card, first turn on the 80STORE soft switch by writing to location 49153 (negative decimal -16383 or hexadecimal $C001). With 80STORE on, the page-select switch PAGE2 switches between the portion of the 80-column display stored in Page 1 of main memory and the portion stored in Page 1X in auxiliary memory. To select Page 1X, turn the PAGE2 soft switch on by reading or writing at location 49237 (-16299, $C055).

You'll have to write a short program to try out the 80STORE and PAGE2 soft switches. When you try to change these switches by using the Monitor program, it changes them back in the process of displaying the commands you type.

If you want to use the optional double-high-resolution display described in Chapter 2, you can store data directly into high-resolution graphics Page 1X in auxiliary memory in a similar fashion. Turn on both 80STORE and HIRES, then use PAGE2 to switch from Page 1 in main memory to Page 1X in auxiliary memory.

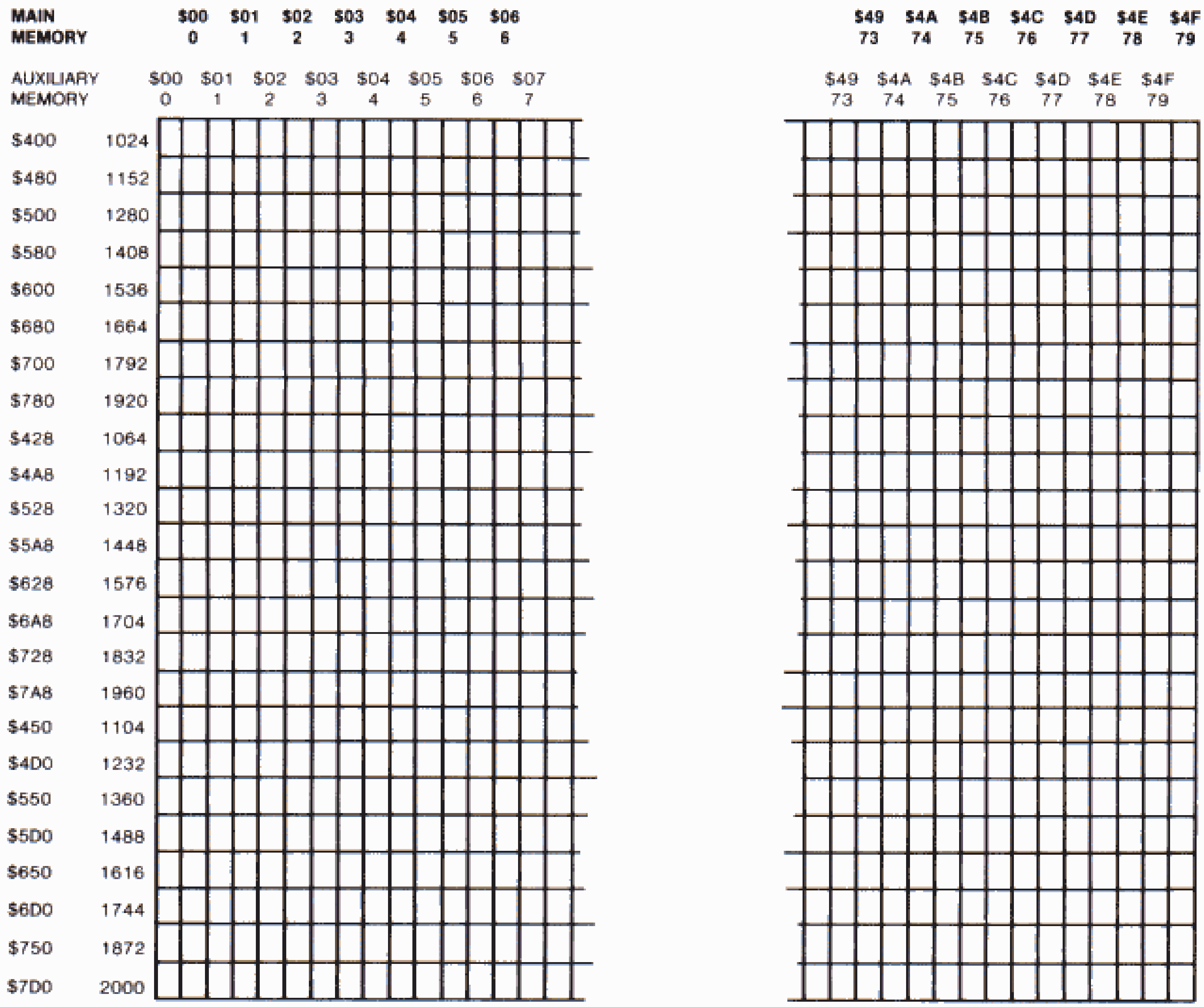
The memory mapping for double high-resolution graphics is similar to the normal high-resolution mapping described in Chapter 2 of the *Apple IIe Reference Manual*, with the addition of the column doubling produced by the 80-column display. Like the 80-column text mode, the double high-resolution graphics mode displays two bytes in the time normally required for one, but it uses high-resolution graphics Page 1 and Page 1X instead of text Page 1 and Page 1X.

For a description of the way the **high-order bit** acts as **color-select bit** in high-resolution displays, see Chapters 2 and 7 of the *Apple IIe Reference Manual*.

Double high-resolution graphics mode displays each pair of data bytes as 14 adjacent dots, seven from each byte. The high-order bit (color-select bit) of each byte is ignored. The auxiliary-memory byte is displayed first, so data from auxiliary memory appears in columns 0-6, 14-20, etc., up to columns 547-552. Data from main memory appears in columns 7-13, 21-27, and so on up to 553-559.

As in 80-column text, there are twice as many dots across the display screen, so the dots are only half as wide. On a TV set or low-bandwidth monitor, single dots will be dimmer than normal.

Figure 3-1. Map of 80-Column Text Display



Auxiliary Memory Switching

This section describes the switches used to access the auxiliary memory for storing programs and data.

Warning
The display soft switches `80STORE`, `PAGE2`, and `HIRES`, discussed here and in the previous section, are used primarily for addressing display data. These switches override the general-purpose switches described in this section, so you must set them correctly even if your program doesn't use them.

Switching the 48K Bank

Switching the 48K-byte section of memory is performed by two soft switches: `RAMRD` selects main or auxiliary memory for reading, and `RAMWRT` selects main or auxiliary memory for writing. As shown in Table 3-3, each switch has a pair of memory locations dedicated to it, one to select main memory, and the other to select auxiliary memory. Setting the read and write functions independently makes it possible for a program whose instructions are being fetched from one 48K-byte memory space to store data into the other 48K memory space.

Warning
Before using these switches, you must fully understand the effects of switching to auxiliary memory. For example, an application program running in the 48K bank of auxiliary memory that tries to use the built-in I/O routines by calling the standard I/O links will crash even though the main ROM, which contains the built-in I/O routines, has been selected. This happens because the standard links call DOS routines, and DOS is in the 48K bank of main memory, which is locked out while the application program is running in auxiliary memory.

When `RAMWRT` and `RAMRD` are on, auxiliary memory is used; when they are off, main memory is used.

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 48K bank by using RAMRD and RAMWRT. These areas in auxiliary memory can also be controlled separately by using the display-page switches 80STORE, PAGE2, and HIRES described in "Addressing the 80-Column Display Directly."

As shown in Table 3-3, 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 PAGE2 switch selects main or auxiliary memory in the text display Page 1, $0400 to $07FF; with HIRES on, the PAGE2 switch selects main or auxiliary memory in text Page 1 and high-resolution graphics Page 1, $2000 to $3FFF.

If you are using both the 48K-bank control switches and the 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.

You can find out the settings of these soft switches by reading from two other locations. The byte you read at location $C013 has its high bit (the sign bit) set to 1 if RAMRD is on (auxiliary memory is enabled for reading), or 0 if RAMRD is off (the 48K block of main memory is enabled for reading). The byte at location $C014 has its high bit set to 1 if RAMWRT is on (auxiliary memory is enabled for writing), or 0 if RAMWRT is off (the 48K block of main memory is enabled for writing).

Figure 3-2. Effect of Switching RAMRD and RAMWRT with 80STORE Off

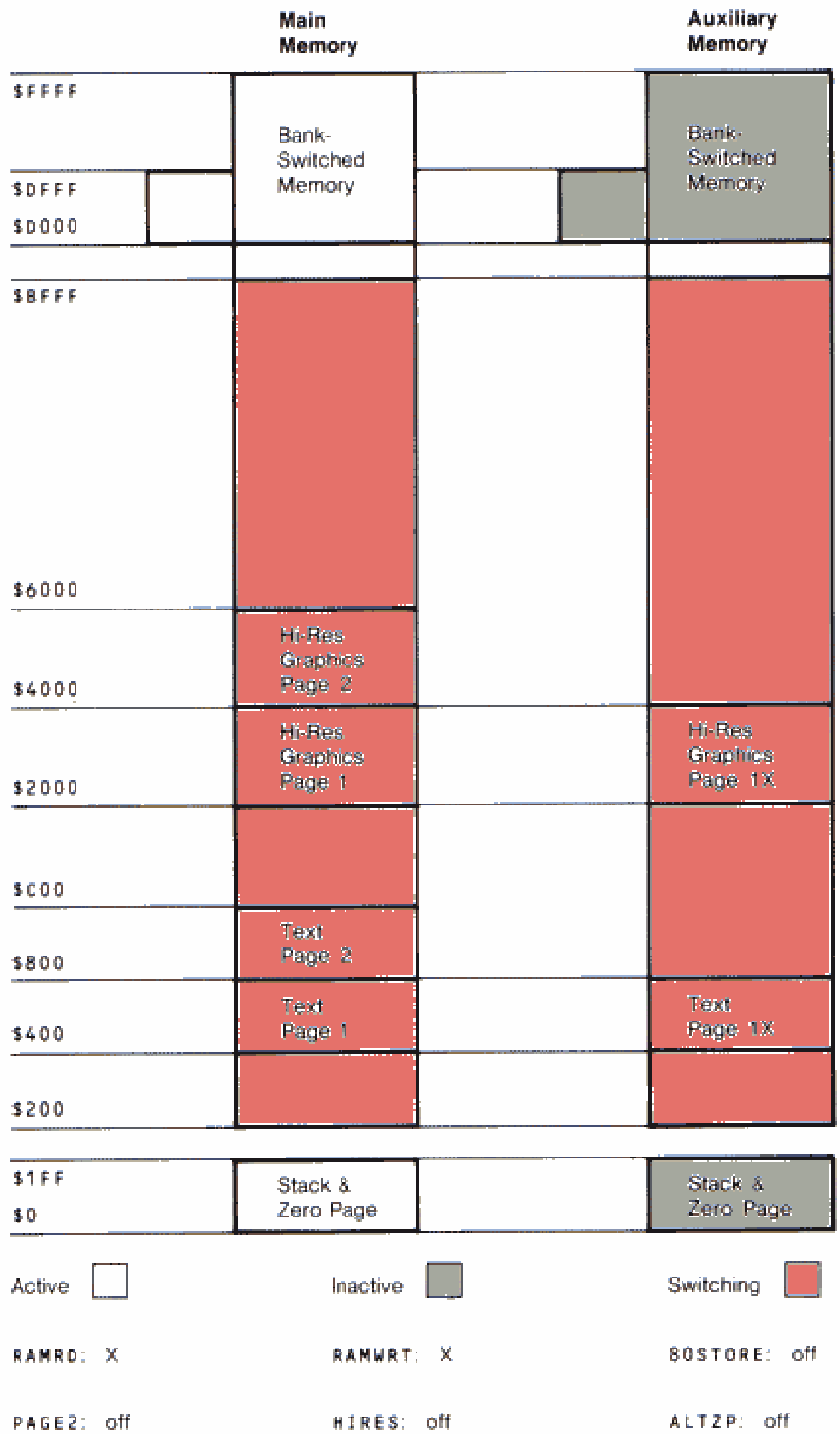
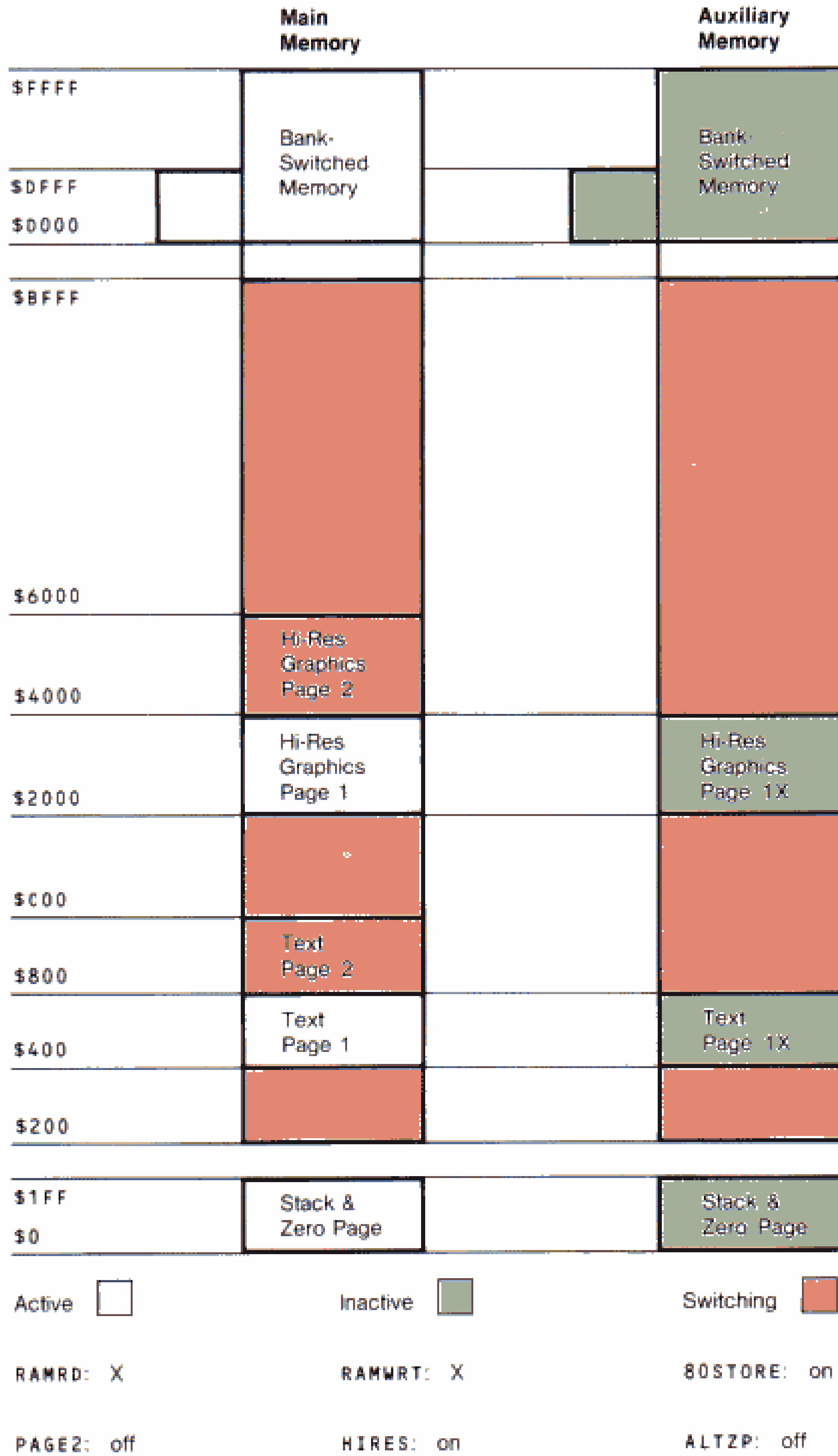


Figure 3-3. Effect of Switching `RAMRD` and `RAMWRT` with `80STORE` and `HIRES` On



Switching High Memory, Stack, and Zero Page

The single soft switch ALTZP (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 3-3, 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 reading and writing. The section "Auxiliary-Memory Subroutines" describes firmware that you can call to help you switch between main and auxiliary memory.

When the ALTZP soft switch is on, auxiliary memory is used; when it is off, main memory is used.

To find out the setting of this soft switch, read location $C016. The data byte you get has its high bit (the sign 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 (the same areas in main memory are selected).

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 3-3 share their memory locations with the keyboard functions listed in Chapter 2 of the *Apple IIe Reference Manual*. Whichever operation—read or write—is shown in Table 3-3 for controlling the auxiliary memory is the one that is **not** used for reading the keyboard and clearing the strobe.

Table 3-3. Auxiliary-Memory Select Switches. (1) When 80STORE is on, the PAGE2 switch works as shown; when 80STORE is off, PAGE2 doesn't affect the auxiliary memory. (2) When 80STORE is on, the HIRES switch enables you to use the PAGE2 switch to select between high-resolution Page 1 areas in main and auxiliary memory.

Name	Function	Location Hex	Decimal		Notes
RAMRD	On: Read Aux. 48K	$C003	49155	-16381	Write
	Off: Read Main 48K	$C002	49154	-16382	Write
	Read RAMRD Switch	$C013	49171	-16365	Read
RAMWRT	On: Write Aux. 48K	$C005	49157	-16379	Write
	Off: Write Main 48K	$C004	49156	-16380	Write
	Read RAMWRT Switch	$C014	49172	-16354	Read
ALTZP	On: Aux. Stack, Zero Page, and Bank-Switched Memory	$C009	49161	-16373	Write
	Off: Main Stack, Zero Page, and Bank-Switched Memory	$C008	49160	-16374	Write
	Read ALTZP Switch	$C016	49174	-16352	Read
80STORE	On: Access Page 1X	$C001	49153	-16383	Write
	Off: Use RAMRD, RAMWRT	$C000	49152	-16384	Write
	Read 80STORE Switch	$C018	49176	-16360	Read
PAGE2	On: Access Aux. Memory	$C055	49237	-16299	1
	Off: Access Main Memory	$C054	49236	-16300	1
	Read PAGE2 Switch	$C01C	49180	-16356	Read
HIRES	On: Access High-Resolution Page IX	$C057	49239	-16297	2
	Off: Use RAMRD, RAMWRT	$C056	49238	-16298	2
	Read HIRES Switch	$C01D	49181	-16355	Read

Figure 3-4. Effect of Switching ALTZP

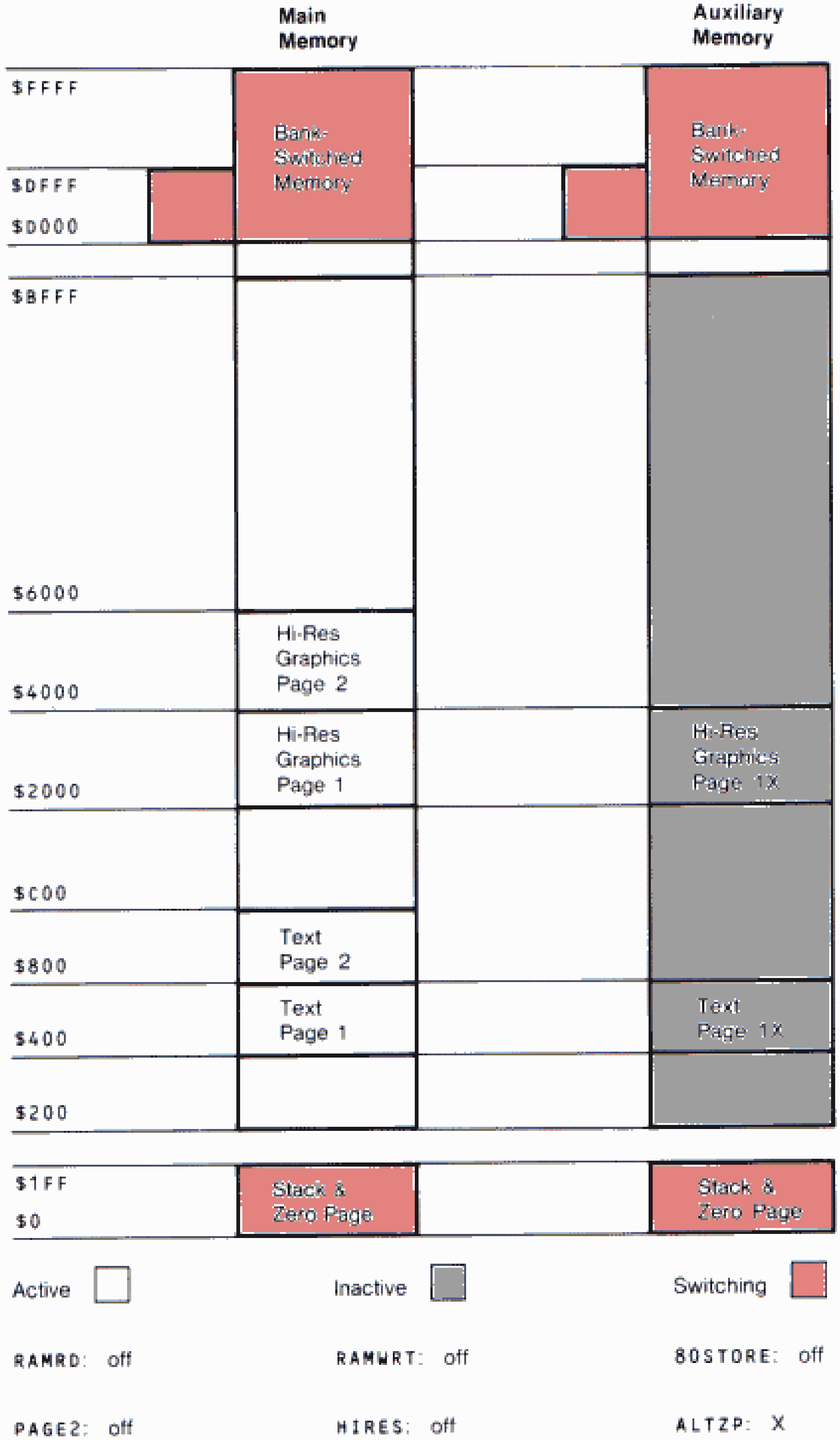
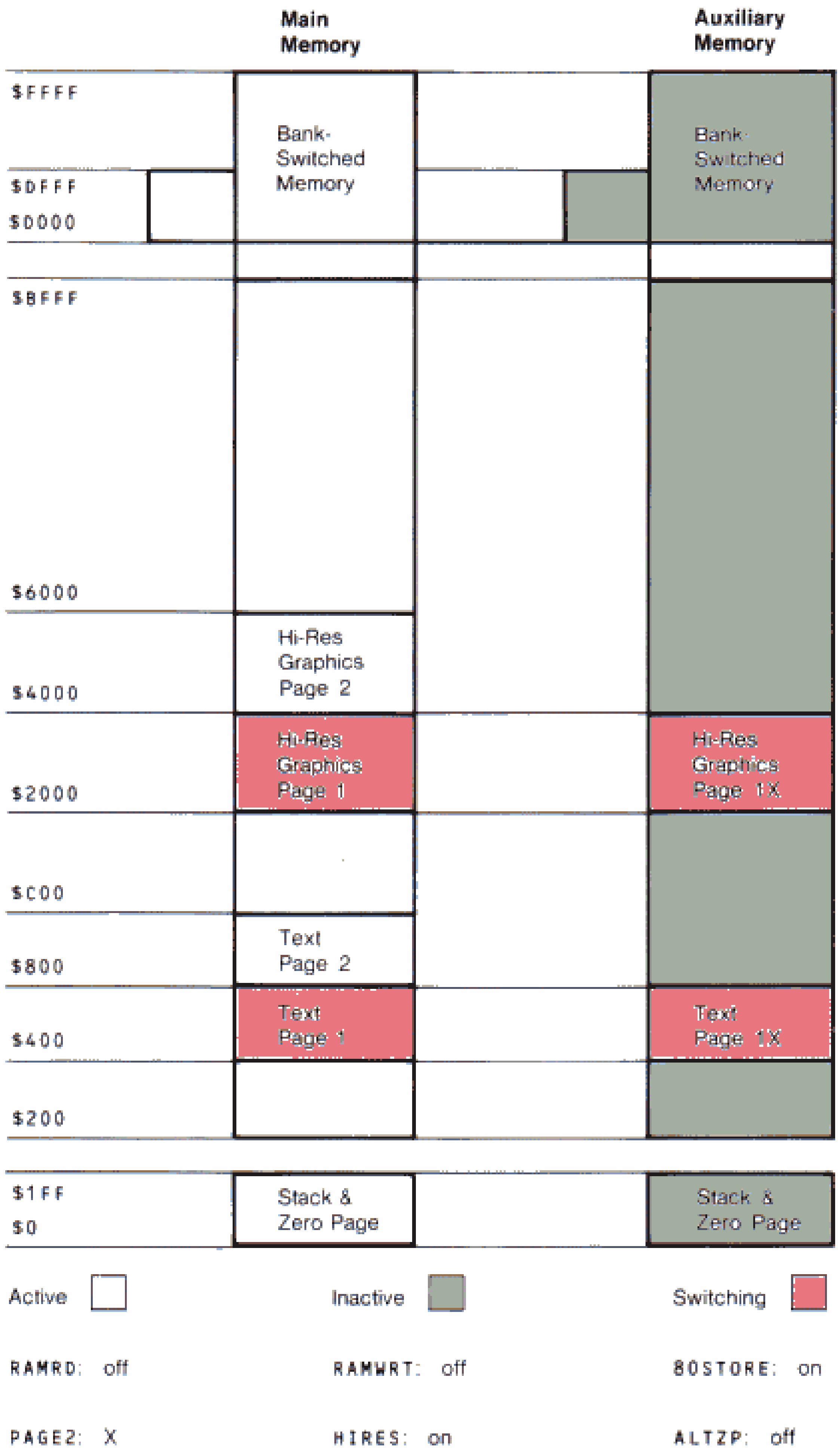


Figure 3-5. Effect of Switching PAGE2 with 80STORE and HIRES On



Auxiliary-Memory Subroutines

If you want to write assembly-language programs or procedures that use auxiliary memory, the built-in auxiliary-memory subroutines will be helpful. These subroutines make it possible to use the auxiliary memory without having to manipulate the soft switches already described.

The subroutines described in this section 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 inexplicable crashes.

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

Table 3-4. 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.

The **carry bit** is bit `0` in the processor status word; use the `SEC` instruction to set it, and `CLC` to clear it.

Warning

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

Remember that Pascal uses page zero too, so you can't use AUXMOVE from a Pascal procedure without saving the contents of page zero first, and restoring them afterward.

The pairs of bytes you use for passing addresses to this subroutine are called A1, A2, and A4; they are used for passing parameters to several of the Apple IIe's built-in routines. The addresses of these byte pairs are shown in Table 3-5.

Table 3-5. 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 (SEC); to copy data from auxiliary memory to main memory, clear the carry bit (CLC).

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 3-6. 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

The **overflow bit** is bit 6 in the processor status word; use the CLV instruction to clear it. To set it, force an overflow by adding two numbers that total more than 127.

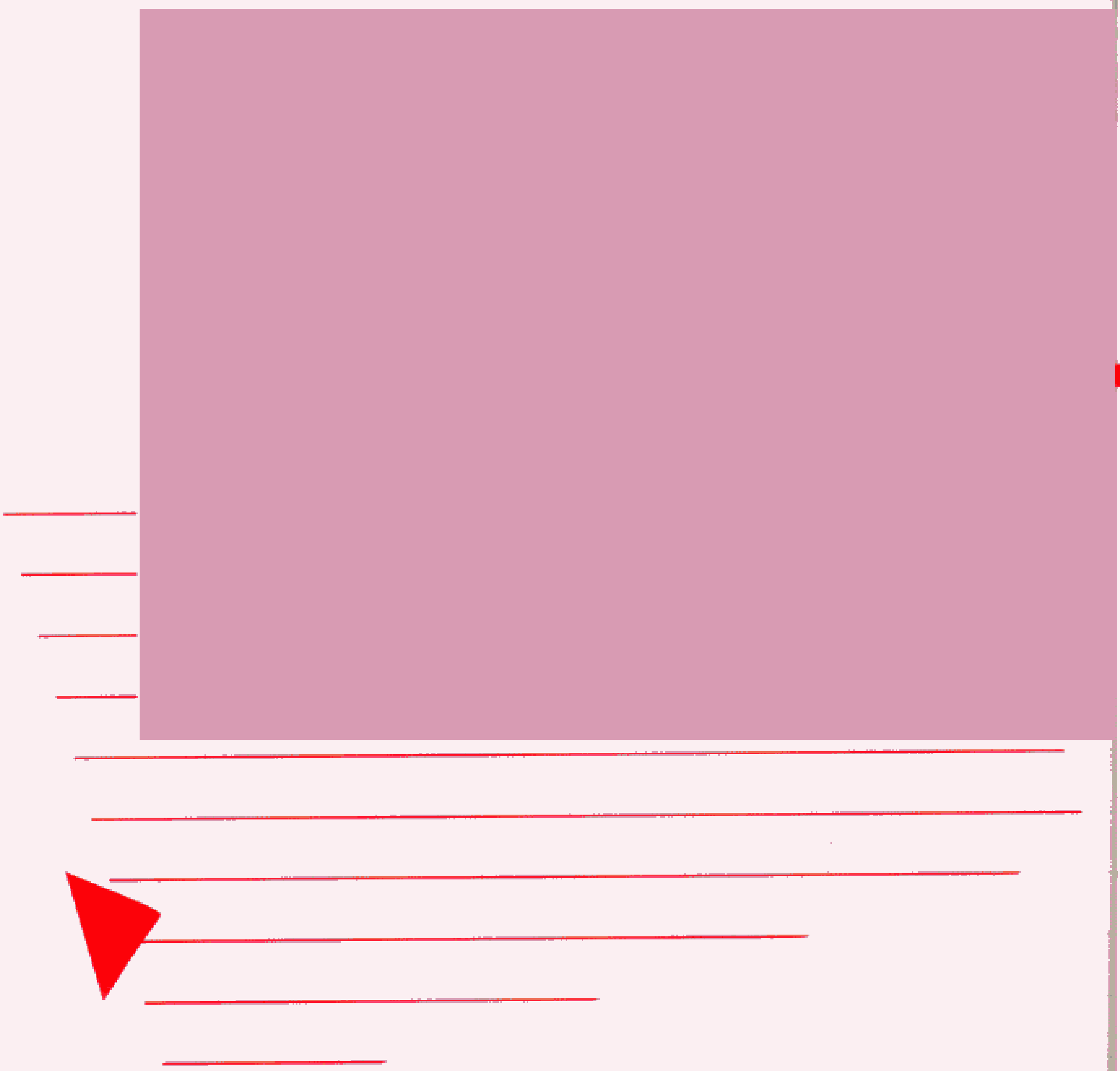
Put the transfer address into the two bytes at locations $3ED and $3EE, with the low-order byte first, as usual. The direction of the transfer is controlled by the carry bit: set the carry bit to transfer to a program in auxiliary memory; clear the carry bit to transfer to a program in main memory. Use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

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.

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.

Programming Examples



Chapter 4

Programming Examples

This chapter contains examples showing how to use the auxiliary memory from a program. These examples are not intended to be universal routines that everyone can use as is; rather, they are representative examples showing how specific operations have been implemented. You will probably want to study the examples to see how it is done, then copy or modify them to suit your application.

Identifying Different Configurations

By identifying the configuration of the machine they are running on, application programs for the Apple IIe can take advantage of the new features and still remain compatible with older Apple II's. This section gives a procedure for doing this from assembly language and shows how to use the identification routine in programs written in Applesoft BASIC and Pascal.

The identification routine returns a value to the calling program that depends on the type of machine it is running on. Table 4-1 shows the return codes.

Table 4-1. Identification Return Codes

$00	(0)	=	not an Apple IIe
$20	(32)	=	Apple IIe, but no Apple IIe 80-Column Text Card
$40	(64)	=	Apple IIe with 80-Column Text Card without auxiliary memory
$80	(128)	=	Apple IIe with Extended 80-Column Text Card

Note: An 80-column card installed in expansion slot 3 will work in an Apple IIe the same as in an Apple II or Apple II Plus, but it does not activate the built-in 80-column firmware. The identification program does not detect such a card, but returns a code of 32: no Apple IIe 80-Column Text Card.

Here is an outline of the procedure the identification routine uses to identify an Apple IIe and its variations:

1. Save four identification bytes from the ROM/RAM area ($D000 to $FFFF).
2. Disable interrupts.
3. Switch bank-switched memory to read ROM by reading $C089 twice.
4. Identify Apple IIe by finding the value 06 at $FBB3.
5. If Apple IIe, and high bit is on at location $C017, then the computer has a text card.
6. If Apple IIe with 80-Column Text Card, then check for auxiliary memory:
   a. If $C013's high bit is on, then reading auxiliary memory so must have auxiliary memory.
   b. If $C016's high bit is on, then reading auxiliary zero page so must have auxiliary memory.
   c. If sparse memory mapping (no upper four address bits so that $800 has the same RAM location as $C00), then no auxiliary memory.
      1. Exchange a section of zero page with the section of code that switches memory banks. This way the zero page data is saved and the program doesn't get switched out.
      2. Jump to the relocated code on page zero.
      3. Switch in auxiliary memory ($200 - $BFFF) for reading and writing by writing to $C005 and $C003.
         Note: Auxiliary memory locations $400-$800 and $2000-$4000 may not be available depending upon the setting of soft switches for 80-column display and high-resolution graphics—they have priority over auxiliary memory selection.

4. Store a value at $800, and see if same value at $C00. If not, then auxiliary memory.
5. Change value at $C00, and see if $800 changes to same value. If so, then no auxiliary memory.
6. Set soft switches for reading and writing to main memory by writing to $C002 and $C004.
7. Jump back into program on main RAM.
8. Put zero page back.

7. Store identification byte for later reference by calling routine.
8. If Pascal routine then turn card back on by reading $C08B twice.
9. The BASIC or assembly-language routines restore the RAM/ROM area as it originally was by checking four bytes saved at the start of the routine.
10. Enable interrupts.
11. Return to caller.

For some applications it may not be necessary to identify the exact configuration of the computer. For example, if your program cannot use the auxiliary memory, then you would not need to know whether it is available or not. In that case you may want to eliminate parts of the routine. For other applications the identification routine will use memory space required by your program, so you will need to move the routine to some other location.

Warning
If you change the identification routine, make sure that it still determines the configuration in the same way as the original. Later revisions of the Apple IIe may not support other identification procedures.

Apple IIe Identification in Assembly Language

The assembly-language subroutine given here is assembled to machine language in locations $2D0 through $3CF. To call the subroutine, your program does a jump to subroutine (JSR) to $2D4. When the subroutine returns, the identification code is stored in memory location $3CF.

Apple IIe Identification Program
1982

```
PARAM     ORG     $204
SAFE      EQU     $3CF
SAVE      EQU     $0001           ;START OF CODE RELOCATED ON PAGE ZERO
          EQU     $200            ;START OF FOUR BYTE LANGUAGE CARD ID
          PHP                     ;DISABLE INTERRUPTS
          SEI
          LDA     $E000           ;SAVE 4 BYTES FROM
          STA     SAVE            ;ROMRAM AREA FOR LATER
          LDA     $D000           ;RESTORING OF RAMROM
          STA     SAVE+1          ;TO ORIGINAL CONDITION
          LDA     $D400
          STA     SAVE+2
          LDA     $D800
          STA     SAVE+3
          LDA     $C081           ;ENSURE READING ROM BY TURNING OFF
          LDA     $C081           ;BANKABLE MEM.
          LDA     $FBB3           ;GET APPLE IIE SIGNATURE BYTE
          CMP     #$6
          BNE     OUT1            ;IF NOT #6 THEN NOT APPLEIIE
          LDA     $C017           ;WAS 80 COLUMNS FOUND DURING STARTUP
          BMI     OUT2            ;IF HI BIT ON THEN NO 80 COLUMN CARD
          LDA     $C013           ;SEE IF AUX MEMORY BEING READ
          BMI     OUT4            ;AUX MEM BEING USED SO AUX MEM AVAIL.
          LDA     $C016           ;SEE IF AUX ZP BEING USED
          BMI     OUT4            ;AUX ZP BEING USED SO AUX MEM AVAIL
          LDY     #DONE-START     ;NOT SURE YET SO KEEP CHECKING
MV        LDX     START-1,Y       ;SWAP SECTION OF ZP WITH
          LDA     SAFE-1,Y        ;CODE NEEDING SAFE LOCATION DURING
          STX     SAFE-1,Y        ;READ AUX MEM
          STA     START-1,Y
          DEY
          BNE     MV
          JMP     SAFE            ;JUMP TO SAFE GROUND
ON        PHP                     ;BACK FROM SAFE GROUND. SAVE STATUS
          LDY     #DONE-START     ;MOVE ZERO PAGE BACK
MV2       LDA     START-1,Y
          STA     SAFE-1,Y
          DEY
          BNE     MV2
          PLA                     ;GET BACK STATUS
          BCS     OUT3            ;CARRY SET SO NO AUX MEM
OUT4      LDA     #$80            ;MADE IT SO THERE IS AUX MEM SET
          STA     PARAM           ;PARAM=$80
          JMP     OUT
OUT3      LDA     #$40            ;80 COLUMNS BUT NO AUX SO SET
          STA     PARAM           ;PARAM=$40
          JMP     OUT
OUT2      LDA     #$20            ;APPLE IIE BUT NO CARD SO SET
          STA     PARAM           ;PARAM=$20
          JMP     OUT
OUT1      LDA     #0              ;NOT AN APPLE IIE SO SET PARAM=0
          STA     PARAM
```

```
OUT      LDA   $E000      ;IF ALL 4 BYTES THE SAME
         CMP   SAVE       ;THE LANGUAGE CARD NEVER
         BNE   OUTON      ;WAS ON SO DO NOTHING
         LDA   $D000
         CMP   SAVE+1
         BNE   OUTON
         LDA   $D400
         CMP   SAVE+2
         BNE   OUTON
         LDA   $D800
         CMP   SAVE+3
         BEQ   GOOUT
OUTON    LDA   $C088      ;NO MATCH, SO TURN FIRST
         LDA   $E000      ;BANK OF LC ON AND CHECK
         CMP   SAVE
         BEQ   OUTON0
         LDA   $C080
         JMP   GOOUT
OUTON0   LDA   $D000
         CMP   SAVE+1     ;IF ALL LOCATIONS CHECK
         BEQ   OUTON1     ;THEN DO NOTHING MORE
         LDA   $C080      ;OTHERWISE TURN ON BANK 2
         JMP   GOOUT
OUTON1   LDA   $D400      ;CHECK SECOND BYTE IN BANK 1
         CMP   SAVE+2
         BEQ   OUTON2
         LDA   $C080      ;SELECT BANK 2
         JMP   GOOUT
OUTON2   LDA   $D800      ;CHECK THIRD BYTE IN BANK 1
         CMP   SAVE+3
         BEQ   GOOUT
         LDA   $C080      ;SELECT BANK 2
GOOUT    PLP              ;RESET INTERRUPTS
         RTS
*** ROUTINE RUN IN SAFE AREA NOT AFFECTED BY MOVES ***
START    LDA   #$EE       ;TRY STORING IN AUX MEM
         STA   $C005      ;WRITE TO AUX WHILE ON MAIN ZP
         STA   $C003      ;SET TO READ AUX RAM
         STA   $800       ;CHECK FOR SPARSE MEM MAPPING
         LDA   $C00       ;SEE IF SPARSE MEMORY -SAME VALUE
         CMP   #$EE       ; 1K AWAY
         BNE   AUXMEM
         ASL   $C00       ;MAY BE SPARSE MEM SO CHANGE VALUE
         LDA   $800       ; & SEE WHAT HAPPENS
         CMP   $C00
         BNE   AUXMEM
         SEC              ;SPARSE MAPPING SO NO AUX MEM
         BCS   BACK
AUXMEM   CLC              ;THERE IS AUX MEM
BACK     STA   $C004      ;SWITCH BACK TO WRITE MAIN RAM
         STA   $C002      ;SWITCH BACK MAIN RAM READ
         JMP   ON         ;CONTINUE PROGRAM ON PG 3 MAIN RAM
DONE     NOP              ;END OF RELOCATED PROGRAM MARKER
```

Apple IIe Identification from BASIC

One way to identify the configuration of an Apple IIe from BASIC is to load (using `BLOAD`) the machine-code version of the assembly-language routine described in the previous section, then execute a `CALL` statement to location `724` (`$2D4`). When the subroutine returns to the BASIC program, executing a `PEEK` at location `975` (`$3CF`) gets the result.

Here is another approach to writing a BASIC program to identify the type of Apple II it is running on. In this program the assembled code for the assembly-language identification routine from the last section is included in the `DATA` statements.

Apple IIe Identification from Applesoft BASIC

```
10 DATA 8, 120, 173, 0, 224, 141, 208, 2, 173, 0, 208, 141, 209, 2, 173, 0,
        212, 141, 210, 2, 173, 0, 216, 141, 211, 2, 173, 129, 192, 173, 129,
        192, 173, 179, 251, 201, 6, 208, 73, 173
20 DATA 23, 192, 48, 60, 173, 19, 192, 48, 39, 173, 22, 192, 48, 34, 160, 42,
        190, 162, 3, 185, 0, 0, 150, 0, 153, 162, 3, 136, 208, 242, 76, 1, 0,
        8, 160, 42, 185, 162, 3, 153
30 DATA 0, 0, 136, 208, 247, 104, 176, 8, 169, 128, 141, 207, 3, 76, 73, 3,
        169, 64, 141, 207, 3, 76, 73, 3, 169, 32, 141, 207, 3, 76, 73, 3, 169,
        0, 141, 207, 3, 173, 0, 224
40 DATA 205, 208, 2, 208, 24, 173, 0, 208, 205, 209, 2, 208, 16, 173, 0, 212,
        205, 210, 2, 208, 8, 173, 0, 216, 205, 211, 2, 240, 56, 173, 136, 192,
        173, 0, 224, 205, 208, 2, 240, 6
50 DATA 173, 128, 192, 76, 161, 3, 173, 0, 208, 205, 209, 2, 240, 6, 173, 128,
        192, 76, 161, 3, 173, 0, 212, 205, 210, 2, 240, 6, 173, 128, 192, 76,
        161, 3, 173, 0, 216, 205, 211, 2
60 DATA 240, 3, 173, 128, 192, 40, 96, 169, 238, 141, 5, 192, 141, 3, 192,
        141, 0, 8, 173, 0, 12, 201, 238, 208, 14, 14, 0, 12, 173, 0, 8, 205, 0,
        12, 208, 3, 56, 176, 1, 24
70 DATA 141, 4, 192, 141, 2, 192, 76, 29, 3, 234
80 ALOOK = 975:START = 724
90 FOR I = 0 TO 249
100 READ BYTE
110 POKE START + I,BYTE
120 NEXT
130 CALL START
140 RESULTS = PEEK (ALOOK)
150 PRINT RESULTS: REM RESULTS OF 0 MEAN NOT A IIE; 32 MEANS AIIE BUT NO 80
        COLUMNS; 64 MEANS AIIE WITH 80 COLUMNS BUT NO AUX MEM; 128 MEANS AIIE
        WITH AUX MEM
160 END
```

Apple IIe Identification from Pascal

Here is the assembly-language identification program previously described in the form of a Pascal procedure.

Apple IIe Identification from Pascal

```
;
        .MACRO    POP         ;SAVE PASCAL RETURN ADDRESS
        PLA
        STA       %1
        PLA
        STA       %1+1
        .ENDM
;
;
;
        .MACRO    PULL_BIAS       ;ADJUST FOR FUNCTION
        PLA
        PLA
        PLA
        PLA
        .ENDM
;
;
;
        .FUNC     ID,0
RETURN  .EQU      0           ;TEMP STORAGE OF RETURN TO PASCAL ADDRESS
SAFE    .EQU      0002        ;START OF CODE RELOCATED ON PAGE ZERO
;
;
;
        POP       RETURN
        PULL_BIAS
        PHP                   ;LOCK OUT INTERRUPTS
        SEI
        LDA       0C089       ;ENSURE READING ROM BY TURNING OFF
        LDA       0C089       ;BANKABLE MEM
        LDA       0FBB3       ;GET APPLE IIe SIGNITURE BYTE
        CMP       #6
        BNE       OUT1        ;IF NOT #6 THEN NOT APPLE IIe
        LDA       0C017       ;WAS 80 COLUMNS FOUND DURING STARTUP
        BMI       OUT2        ;IF HIGH BIT ON THEN NO 80-COLUMN CARD
        LDA       0C013       ;SEE IF AUX MEMORY BEING READ
        BMI       OUT4        ;AUX MEM BEING USED SO AUX MEM AVAIL
        LDA       0C016       ;SEE IF AUX ZP BEING USED
        BMI       OUT4        ;AUX ZP BEING USED SO AUX MEM AVAIL
        LDY       #2A         ;NOT SURE YET SO KEEP CHECKING
MV      LDX       START-1,Y   ;SWAP SECTION OF ZP WITH
        LDA       SAFE-1,Y    ;CODE NEEDING SAFE LOCATION DURING
        STX       SAFE-1,Y    ;READ AUX MEM
        STA       START-1,Y
        DEY
```

```
        BNE     MV
        JMP     SAFE        ;JUMP TO SAFE GROUND
ON      PHP                 ;BACK FROM SAFE GROUND. SAVE STATUS
        LDY     #2A         ;MOVE ZERO PAGE BACK
MV2     LDA     START-1,Y
        STA     SAFE-1,Y
        DEY
        BNE     MV2
        PLA                 ;GET BACK STATUS
        BCS     OUT3        ;CARRY SET SO NO AUX MEM
OUT4    LDA     #80         ;MADE IT SO THERE IS AUX MEM-SET
        STA     PARAM       ;PARAM=$80
        JMP     OUT
OUT3    LDA     #40         ;80 COLUMNS BUT NO AUX SO SET
        STA     PARAM       ;PARAM=$40
        JMP     OUT
OUT2    LDA     #20         ;APPLE IIe BUT NO CARD SO SET
        STA     PARAM       ;PARAM=$20
        JMP     OUT
OUT1    LDA     #0          ;NOT AN APPLE IIe SO SET PARAM=0
        STA     PARAM
OUT     LDA     0C08B       ;GET PASCAL BACK
        LDA     0C08B
        PLP                 ;REACTIVATE INTERRUPTS
        LDA     #0          ;PUT 0 IN HIGH BYTE OF RESULTS
        PHA
        LDA     PARAM       ;PUT FOUND VALUE IN LOW BYTE & PUSH
        PHA
        LDA     RETURN+1    ;RESTORE PASCAL RETURN ADD
        PHA
        LDA     RETURN
        PHA
        RTS
PARAM   .BYTE
;
; ROUTINE RUN IN SAFE AREA NOT AFFECTED BY MOVES
;
START   LDA     #0EE        ;TRY STORING . IN AUX MEM
        STA     0C005       ;WRITE TO AUX WHILE ON MAIN ZP
        STA     0C003       ;SET TO READ AUX RAM
        STA     0800        ;CHECK FOR SPARSE MEM MAPPING
        LDA     0C00        ;SEE IF SPARSE MEMORY-SAME VALUE
        CMP     #0EE        ;1K AWAY
        BNE     AUXMEM
        ASL     0C00        ;MAY BE SPARSE MEM SO CHANGE VALUE
        LDA     0800        ;& SEE WHAT HAPPENS
        CMP     0C00
        BNE     AUXMEM
        SEC                 ;SPARSE MAPPING SO NO AUX MEM
        BCS     BACK
AUXMEM  CLC                 ;THERE IS AUX MEM
BACK    STA     0C004       ;SWITCH BACK TO WRITE MAIN RAM
        STA     0C002       ;SWITCH BACK MAIN RAM READ
        JMP     ON          ;CONTINUE PROGRAM ON PG 3 MAIN RAM
DONE    NOP                 ;END OF RELOCATED PROGRAM MARKER
        .END
```

Storing Graphics Pages from Applesoft

It is generally not practical to use the auxiliary memory from BASIC. A BASIC program can only move its variables in memory by getting very tricky with PEEK and POKE commands, an approach that is both inefficient and dangerous.

There is one form of data that uses lots of memory and is simple enough to handle from Applesoft: high-resolution graphics pages. The auxiliary memory is an ideal place to store as many as five complex graphics pages for rapid loading into the display buffer.

Like all of these examples, the following Applesoft example includes two short assembly-language subroutines. The first listing is the assembly-language form of the subroutines. The second listing is the Applesoft program with the machine-language subroutine included as a series of DATA statements. This method of adding a machine-language subroutine to a BASIC program is not very efficient, but it is convenient for short subroutines.

The program has two phases: in the first, the program generates five different high-resolution views and stores them in auxiliary memory; in the second, the program loads the stored graphics pages back into main memory one after another.

Hi-Res Page Mover for Auxiliary Memory Demo. Using AUXMOVE Subroutine. July 1982
PARM = Hi byte of BUF. ADDR. (Page # times 32).
Call PUTPAG to copy hi-res graphics page to AUX. MEM. location specified by PARM.
Call GETPAG to load hi-res graphics page from AUX. MEM. location specified by PARM.

```
            DSECT
            ORG         $3C
SRCBEG      DS          2
SRCEND      DS          2
            DS          2
DESTBEG     DS          2
            DEND
*
PG1BEG      EQU         $2000
PG1END      EQU         $3FFB
AUXMOVE     EQU         $C311
*
            ORG         $300
PARM        DS          1
*
* MOVE HI-RES PAGE TO AUX MEM:
*
```

```
PUTPAG     EQU     *
           LDA     #>PG1BEG       ;PAGE STARTING
           STA     SRCBEG         ;ADDRESS
           LDA     #<PG1BEG
           STA     SRCBEG+1
*
           LDA     #>PG1END       ;PAGE ENDING
           STA     SRCEND         ;ADDRESS
           LDA     #<PG1END
           STA     SRCEND+1
*
* PARM - DESTINATION ADDRESS
*
           LDA     #0             ;DESTINATION
           STA     DESTBEG        ;PAGE BEGINNING
           LDA     PARM           ;ADDRESS
           STA     DESTBEG+1
*
* USE AUXMOVE TO DO IT:
*
           SEC
           JSR     AUXMOVE
           RTS
*
*
* COPY PAGE TO MAIN MEMORY
*
GETPAG     EQU     *
           LDA     #>PG1BEG       ;DESTINATION
           STA     DESTBEG        ;PAGE BEGINNING
           LDA     #<PG1BEG       ;ADDRESS
           STA     DESTBEG+1
*
* PARM - SOURCE ADDRESSES
*
           LDA     #0             ;PARM FOR
           STA     SRCBEG         ;SOURCE BEGINNING
           LDA     PARM           ;ADDRESS
           STA     SRCBEG+1
*
           LDA     #$F8           ;COMPUTE SOURCE
           STA     SRCEND         ;ENDING ADDRESS
           CLC
           LDA     PARM
           ADC     #$1F
           STA     SRCEND+1
*
* USE AUXMOVE TO DO IT:
*
           CLC
           JSR     AUXMOVE
           RTS
*
```

Globe. Hi-res graphics demonstration for the Apple IIe Extended 80-Column Text Card.

```
99    REM    This program draws five views of a rotating globe and stores
100   REM    five copies of the Hi-Res page in auxiliary memory. It then
107   REM    moves the views from auxiliary memory back into the Hi-Res
108   REM    graphics page in main memory, one after another. The rapid
109   REM    succession of views creates the impression of a solid
117   REM    rotating globe.
118   REM
119   REM
127   REM
128   REM
129   REM
150   REM
160   TEST : HOME
170   PRINT  CHR$ (17): REM CTRL-Q for 40-column display
198   REM
199   REM    Pager subroutines in machine language:
200   DATA   169,0,133,60,169,32,133,61,169,248,133,62,169,63,133
210   DATA   63,169,0,133,66,173,0,3,133,67,56,32,17,195,96,0
220   DATA   169,0,133,66,169,32,133,67,169,0,133,60,173,0,3,133
230   DATA   61,169,248,133,62,24,173,0,3,105,31,133,63,24,32,17,195
             ,96
298   REM
299   REM    Read the Pager subroutines and store at $301:
300   PARM = 768:PUTPAGE = 769:BRINGPAGE = 800
310   FOR I = 0 TO 64
320   : READ BYTE
330   ': POKE PUTPAGE + I,BYTE
340   NEXT I
998   REM
999   REM    Set up constants for drawing meridians (ellipses):
1000  PI = 3.14159265:P2 = PI  2
1010  SP = P2  9: REM angle between meridians
1020  EP = SP  5: REM starting angle increment between views
1030  DT = PI  15: REM segment size (angle) for drawing meridians
1040  B = 1: REM Semi-major axis of ellipses.
1998  REM
1999  REM    Loop starting at 2000 draws five views and stores them:
2000  FOR VIEW = 1 TO 5
2029  : REM HGR to erase previous view:
2030  : HGR : HCOLOR= 3
2040  : REM Draw picture frame:
2050  : HPLOT 60,0 TO 60,159 TO 219,159 TO 219,0 TO 60,0
2100  : VTAB 23: HTAB 9
2120  : PRINT "...constructing view #";VIEW
2988  : REM
2990  :DP = EP * VIEW: REM different starting angle each view.
2999  : REM Loop starting at 3000 draws meridians (ellipses):
3000  : FOR IANGLE = DP TO PI STEP SP
3100  ::A = COS (IANGLE): REM Semi-minor axis of ellipse.
3200  ::FIRST = 1: REM for plotting
3990  :: REM
3999  :: REM Loop starting at 4000 draws a meridian (ellipse):
4000  :: FOR THETA = 0 TO PI STEP DT
4020  ::: LET X = A * SIN (THETA)
4040  ::: LET Y = B * COS (THETA)
4059  ::: REM Next two lines scale PX and PY for plotting.
4060  ::: LET PX = X * 55 + 140
4080  ::: LET PY = Y * 55 + 80
4100  ::: IF FIRST THEN HPLOT PX,PY:FIRST = 0
```

```
4110  ::: IF NOT FIRST THEN HPLOT TO PX,PY
4200  :: NEXT THETA
4300  : NEXT IANGLE
4400  : VTAB 23: HTAB 9
4410  : PRINT "...storing view #";VIEW
4499  : REM
4500  : REM Put view in auxiliary memory:
4510  : POKE PARM,VIEW * 32
4520  : CALL PUTPAGE
4600  NEXT VIEW
4689  REM
4690  REM     Five views stored-- now show them:
4700  HOME : VTAB 23
4720  HTAB 3: PRINT "Loading views from auxiliary memory."
4998  REM
4999  REM     Loop starting at 5000 brings views from auxiliary memory:
5000  FOR VIEW = 1 TO 5
5020  : POKE PARM,VIEW * 32
5040  : CALL BRINGPAGE
5060  NEXT VIEW
5997  REM
5998  REM     Repeat same five views forever,
5999  REM     or until the fuse blows:
6000  GOTO 5000
```

Storing Data Strings from Pascal

These Pascal routines use locations `$C00` to `$BFFF` in the auxiliary memory for storage and retrieval of strings.

The code that moves the strings to and from auxiliary memory is stored at `E000` in the Extended 80-Column Text Card. A separate initialization routine puts this code at `E000`, just once, to maintain system performance.

The retrieval routine is very fast, roughly equivalent to a `MoveLeft` from a `Packed Array of Char`. The storage routine is less efficient; if speed is important in your program, you may want to try to optimize it.

Like the other examples, these routines were written for a particular application and are not general-purpose subroutines. They are included here to show you the kind of thing you can do with the auxiliary memory.

Auxiliary Memory String Routines
by R. Lissner

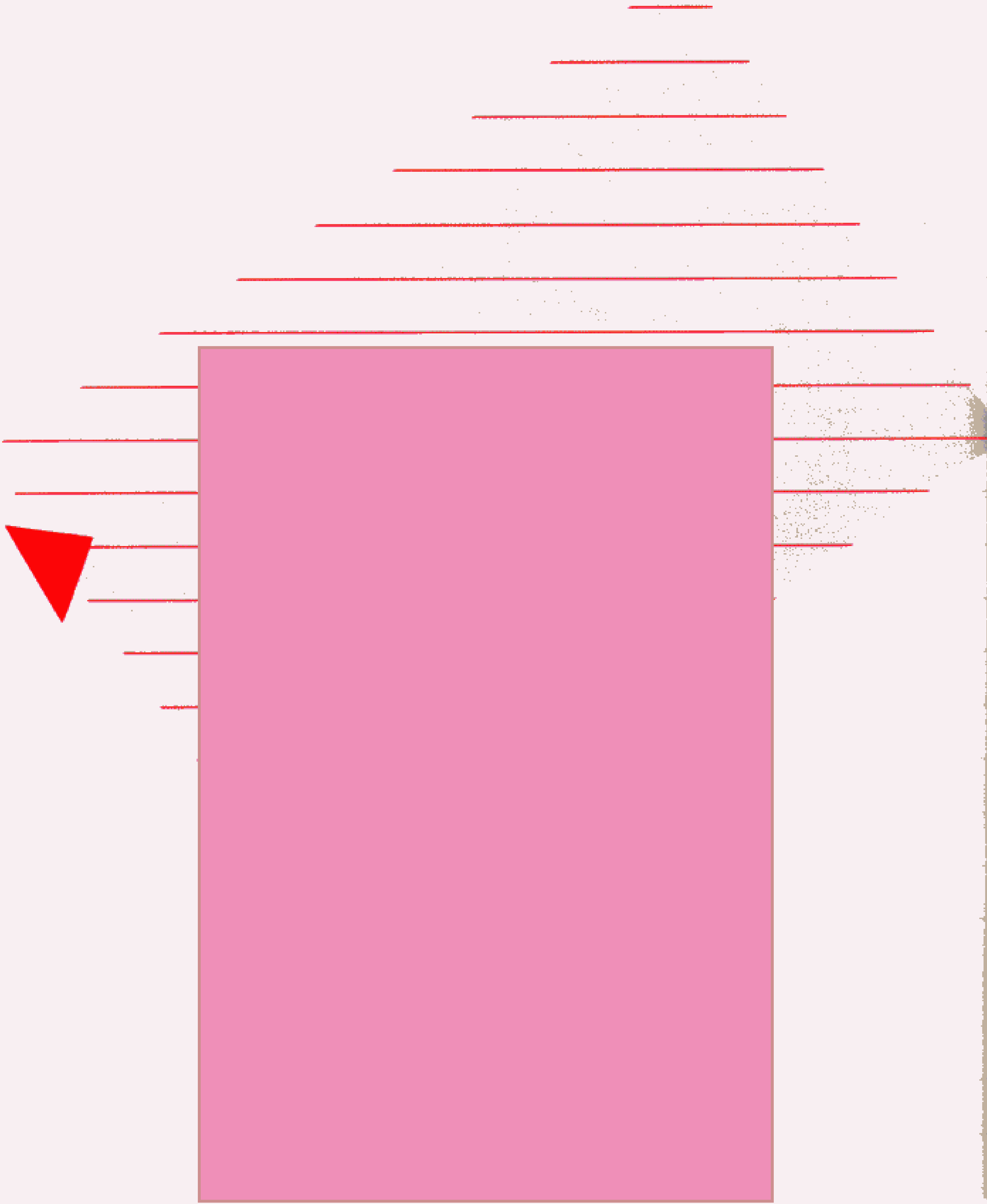
```
;
    The following routine is performed only once. The routines that move
    strings in and out of the Extended 80-Column Text Card are moved to Exxx
    in the auxiliary memory.
;
            .TITLE "ASSEMBLY ROUTINES IIe INITIALIZATION
            .PAGE
            .NOMACROLIST
            .NOPATCHLIST
;
RDMAIN48    .EQU            0C002           ; SOFT SWITCHES. SEE
RDAUX48     .EQU            0C003           ; IIe REFERENCE MANUAL
WRMAIN48    .EQU            0C004
WRAUX48     .EQU            0C005
RWMAIN16    .EQU            0C008
RWAUX16     .EQU            0C009
HIRESOFF    .EQU            0C056
;
;
RETURN0     .EQU            028
RETURN1     .EQU            02A
;
;       REGISTER MAP
;
ZREG00      .EQU            0
ZREG02      .EQU            4
ZREG04      .EQU            6
;
OUT4        STA             RWAUX16         ; WRITE AUX MEMORY
            LDY             #80.            ; LENGTH OF PATCH
OUT4ME0     LDA             E002STUF-1,Y    ;
            STA             0E001,Y
            DEY
            BNE             OUT4ME0
            LDY             #0FF            ; LENGTH OF PATCH
OUT4ME1     LDA             E102STUF-1,Y    ;CODE NEEDING SAFE LOCATION
            STA             0E101,Y
            DEY
            BNE             OUT4ME1
            STA             RWMAIN16        ; WRITE MAIN MEMORY
            STA             HIRESOFF        ; MAKE HIRES P. AVAILABLE
; END OF THIS ROUTINE
;
;
;       Purpose: Moves a string from auxiliary memory to Pascal.
;
;       If the program finds the Extended 80-Column Text Card, the
;       following code is moved to E002.
;
;       The program gets here from a JSR in MOVE_FR_AUX, and goes back so
;       that the auxiliary memory can be turned back off. Zero page on the
;       extended text card contains ZREG00 and ZREG02; they are the
;       arguments for the move. Stack usage: The return address in 48K main
;       memory is stored in the auxiliary stack. This is the only use of the
;       auxiliary memory stack.
```

```
;
;
E002STUF    CLD
            STA         RDAUX48         ; READ AUX 48K
            LDY         #0
            LDA         (ZREG00),Y      ; READING AUX 48K
;           USING AUX ZERO PAGE
            STA         (ZREG02),Y      ; WRITING MAIN 48K
            BEQ         E002EXIT        ; NOT LIKELY, BUT POSSIBLE
            TAY
E002LOOP    LDA         (ZREG00),Y
            STA         (ZREG02),Y
            DEY
            BNE         E002LOOP
E002EXIT    STA         RDMAIN48        ; READ MAIN 48K
            RTS                         ; GOING BACK TO 48K RAM
;
;
;       Purpose: Moves a string from Pascal to auxiliary memory.
;
;       If the program finds the Extended 80-Column Text Card, the
;       following code is moved to E102.
;
;       The program gets here from a JSR in MOVE_TO_AUX, and goes back so
;       that the auxiliary memory can be turned back off. Zero page on the
;       extended text card contains ZREG00 and ZREG02 exactly as they are
;       found on the main zero page.
;
;       Stack usage: The return address in 48K main memory is stored in the
;       auxiliary stack. This is the only use of the auxiliary memory
;       stack.
;
;       Note also that the auxiliary zero page is used for the to and from
;       addresses.
;
;       ZREG00: Address of string that wants to be stored
;       ZREG02: Address of integer that wants to know where it was stored,
;               or receive x'0000' if no room
;       ZREG04: Used to index on receiving address
;
;
NEXTAVAI    .EQU        0E102
E102STUF    .WORD       0C00
            CLD
; X'FF' MEANS RESET BACK TO BEGINNING. DONE FOR EACH NEW FILE
            LDY         #0
            LDA         (ZREG00),Y
            CMP         #0FF
            BNE         E102C0
            LDA         #0              ; RESET TO $C00
            STA         NEXTAVAI+1
            LDA         #0
            STA         NEXTAVAI
            BEQ         E102FAIL        ; UNCONDITIONAL
;           CONTINUE WITH NORMAL ROUTINES
E102C0      LDA         NEXTAVAI+1
            CMP         #0BF            ; CHECK FOR FULL
            BNE         E102C1
```

```
; SPACE IS FULL, SO RETURN ZERO
            LDA         #0
            TAY
            STA         (ZREG02),Y      ; RETURN A ZERO, FULL
            INY
            STA         (ZREG02),Y
            BNE         E102FAIL        ; UNCONDITIONAL
; THERE IS STILL ROOM, SO CONTINUE
E102C1      LDY         #1
            STA         (ZREG02),Y      ; STORE IN RETURN ADDR
            STA         ZREG04+1        ; SETUP THE MOVE
            DEY
            LDA         NEXTAVAI
            STA         (ZREG02),Y      ; LOW BYTE OF RETURN
            STA         ZREG04          ; MORE OF THE MOVE
; NOW INCREMENT THE NEXT AVAILABLE ADDRESS
            CLC
            ADC         #1              ; ADD 1 FOR STRING LENGTH
            BNE         *+5
            INC         NEXTAVAI+1      ; ROLLED INTO NEXT PAGE
            CLC
            LDY         #0
            ADC         (ZREG00),Y      ; ADD LENGTH OF STRING
            STA         NEXTAVAI        ; PUT IT BACK
            BCC         *+5
            INC         NEXTAVAI+1      ; INTO NEXT PAGE
            STA         WRAUX48         ; WRITING INTO AUX 48K
            LDA         (ZREG00),Y      ; READING AUX 48K
;           USING AUX ZERO PAGE
            STA         (ZREG04),Y      ; WRITING MAIN 48K
            BEQ         E102EXIT
            TAY
E102LOOP    LDA         (ZREG00),Y
            STA         (ZREG04),Y
            DEY
            BNE         E102LOOP
E102EXIT    STA         WRMAIN48        ; NOW WRITE MAIN MEM
E102FAIL    RTS                         ; GOING BACK TO 48K RAM
            .END
;
;  The following code is linked into the main Pascal program. This code
;  stores the arguments in the auxiliary zero page and then jumps to Exxx on
;  the Extended 80-Column Text Card.
;
            .TITLE "ASSEMBLY ROUTINES FOR IIe"
            .PAGE
            .NOMACROLIST
            .NOPATCHLIST
;
RDMAIN48    .EQU        0C002
RDAUX48     .EQU        0C003
WRMAIN48    .EQU        0C004
WRAUX48     .EQU        0C005
RWAUX16     .EQU        0C009
RWMAIN16    .EQU        0C008
;
;       RETURN ADDRESS ZERO PAGE LOCATIONS
;
RETURN0     .EQU        028
RETURN1     .EQU        02A
;
```

```
;        REGISTER MAP
;
ZREG00      .EQU          0
ZREG02      .EQU          4
;
;
;
            .TITLE        "MOVE STRINGS FROM IIe AUXILIARY MEMORY"
            .PROC         MOVEFRAU,2
;
;
;        PROCEDURE MOVE_FR_AUX (FROMA; VAR TOA) (* Move string *)
;
;
;        Purpose: Move a string from auxiliary memory to Pascal. Most of the
;        actual move is done at auxiliary memory location E002.
;
;        Stack usage: Input, output addresses.
;
;
; STORE RETURN ADDR IN AUX ZERO PAGE
            POP           RETURN0          ; RETURN TO PASCAL
; ADDRESSES ARE TWO BYTES. PULL BOTH BYTES OFF THE MAIN STACK, THEN SWITCH
; TO AUX ZERO PAGE AND STORE BOTH BYTES.
            PLA
            TAX
            PLA
            STA           RWAUX16          ; SWITCH TO AUX ZP
            STX           ZREG02           ; IN AUX ZERO PAGE
            STA           ZREG02+1         ; STILL IN AUX MEM
            STA           RWMAIN16         ; SWITCH TO MAIN ZP
; STORE FROM ADDRESS IN AUX ZERO PAGE
            PLA
            TAX
            PLA
            STA           RWAUX16          ; SWITCH TO AUX ZP
            STX           ZREG00           ; IN AUX ZERO PAGE
            STA           ZREG00+1         ; STILL IN AUX MEM
; NOW GET OVER TO AUX PAGE AND DO IT ALL
            JSR           0E002
; NOW PROCESS COMING BACK FROM E002 IN AUX MEMORY
            STA           RWMAIN16         ; MAIN ZP AND TOP
            PUSH          RETURN0
            RTS                            ; BACK TO PASCAL
;
;
            .TITLE        "MOVE STRINGS TO IIe AUXILIARY MEMORY"
            .PROC         MoveToAu,2
;
;        PROCEDURE MOVE_TO_AUX (VAR FROMA; VAR TOA) (* Move string *)
;
;
;        Purpose: Move a Pascal string to auxiliary memory. Most of the
;        actual move is done at auxiliary memory location E102.
;
;        Stack usage: Input, output addresses.
;
;
; STORE RETURN ADDR IN AUX ZERO PAGE
            POP           RETURN0          ; MAIN ZP STACK
```

```
; NOW STORE TO ADDRESS IN AUX ZERO PAGE
            PLA                             ; LSB OF ADDR TO RETURN
            TAX
            PLA                             ; MSB OF ADDR TO RETURN
            STA         RWAUX16             ; SWITCH TO AUX ZP
            STX         ZREG02              ; IN AUX ZERO PAGE
            STA         ZREG02+1            ; STILL IN AUX MEM
            STA         RWMAIN16            ; SWITCH TO MAIN ZP
; STORE FROM ADDRESS IN AUX ZERO PAGE
            PLA                             ; LSB OF INPUT STRING
            TAX
            PLA
            STA         RWAUX16             ; SWITCH TO AUX ZP
            STX         ZREG00              ; IN AUX ZERO PAGE
            STA         ZREG00+1            ; STILL IN AUX MEM
; NOW GET OVER TO AUX PAGE AND DO IT ALL
            JSR         0E104               ; JUMP OVER NEXTAVAI AT E102
; RETURN FROM E104 IN AUX MEMORY
            STA         RWMAIN16            ; MAIN TO MAIN ZP AND TOP
            PUSH        RETURN0
            RTS
;
            .END
```



Index

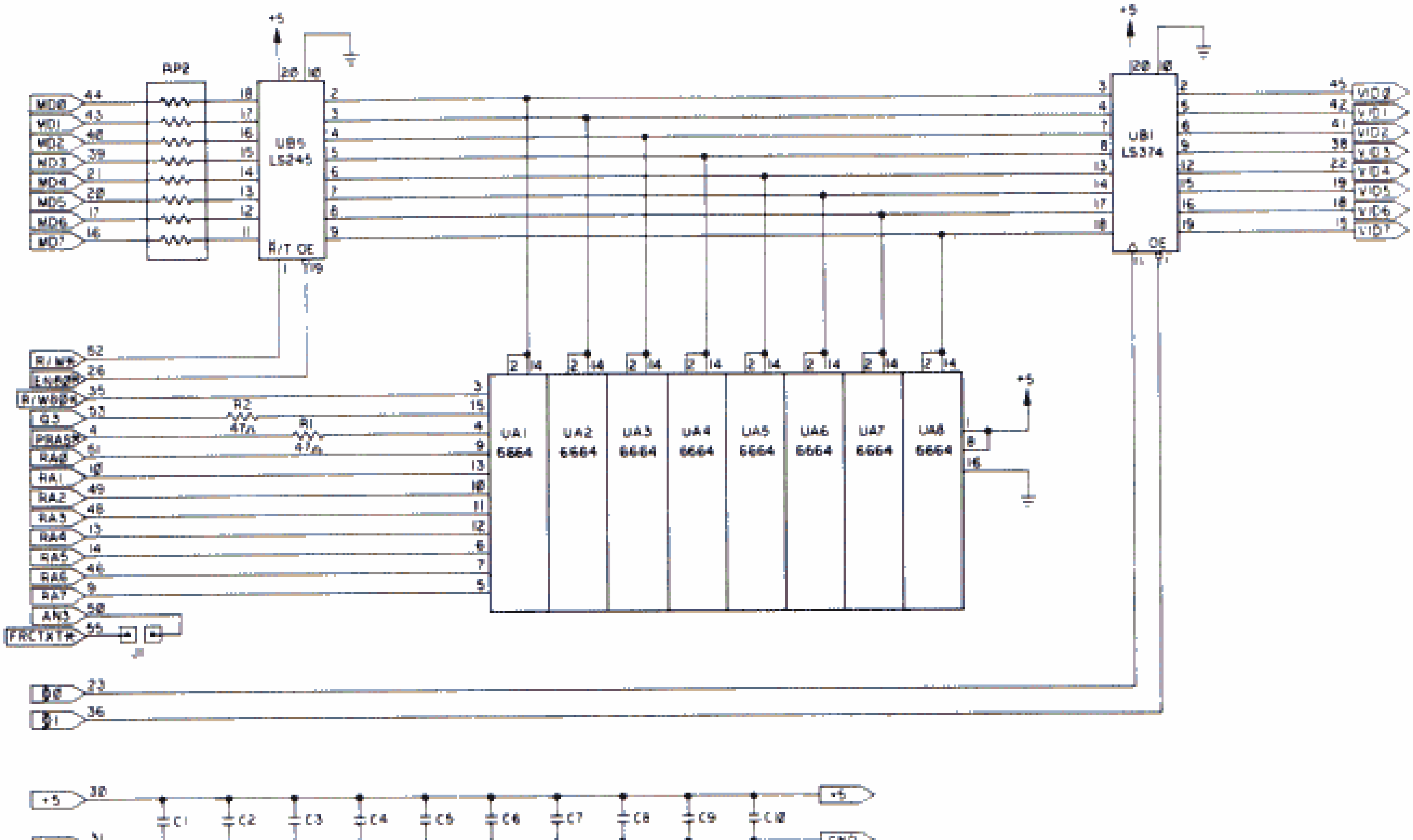
B

C

D

E

Schematic Diagram



apple computer

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

030-1202-A