SOFTECH MICROSYSTEMS

UCSD p-SYSTEM^{tm} VERSION IV

FOR THE APPLE II^{tm} COMPUTER

March 1983
SofTech Microsystems, Inc.
San Diego, California

Printed in the United States of America.

TABLE OF CONTENTS

INTRODUCTION 6

CHAPTER ONE
GETTING STARTED 9
I.1 Powering Up With the p-System 9
I.2 Backing Up Your Diskettes 10
I.2.1 Formatting a New Diskette 10
I.2.2 Making Backups 11
I.3 Important Keyboard Considerations 13
I.4 Running the STARTUP: Diskette 16

CHAPTER TWO
MAKING USE OF THE p-SYSTEM 18
II.1 Software Components 18
II.1.1 Operating System 18
II.1.2 Filer 19
II.1.3 Editor 19
II.1.4 Compilers 19
II.1.5 Libraries 21
II.1.6 Utilities 21
II.1.7 The Booter Utility 21
II.1.8 Disk Utilities 22
II.1.9 Using DISKCHANGE 22
II.1.10 Using DISKCHANGE 24
II.2 How to Create and Run a Simple Program 25
II.3 Ways to Configure Your Diskettes 29
II.4 Large Programs 33
II.5 Binding the Debugger into SYSTEM.PASCAL 35

CHAPTER THREE
CONFIGURING YOUR APPLE II SYSTEM
USING SUPPLIED PERIPHERAL DRIVERS 37
III.1 Reconfiguration Summary 37
III.2 Supplied Drivers 38
III.3 Linking Together an SBIOS 39
III.4 GOTOXY and SYSTEM.MISCINFO 41
III.5 The APPLEINFO Table 42
III.6 Detailed Description of the Supplied Drivers 42

CHAPTER FOUR
WRITING YOUR OWN PERIPHERAL DRIVERS 52
IV.1 The SBIOS Routines 52
IV.1.1 The Jump Vector 52
IV.1.2 Descriptions of the Routines 54
IV.1.3 User-Defined Devices 64
IV.1.4 Physical Organization of the SBIOS 68

IV.1.5 How SBIOS Routines are Called by the p-System 68
IV.1.6 Vector Lists and Register Assignments 69
IV.1.7 The SBIOS Global Variables 72
IV.1.8 Sample Disk Driver 74
IV.1.9 Other Sample Drivers 77
IV.1.10 Example Driver Programs 81
IV.1.11 Polling the Peripherals for I/O 87
IV.1.12 Configuring DISKVECT 87
IV.1.13 Memory Configuration Notes 88
IV.1.14 Reconfiguring the Interpreter 89
IV.1.15 Miscellaneous Notes 93
IV.1.16 Using Real Numbers 93
IV.2 The Utility SETUP 93
IV.2.1 Running SETUP 94
IV.2.2 Miscellaneous Notes for SETUP 95
IV.2.3 The Data Items in SYSTEM.MISCINFO 96
IV.2.4 Sample SYSTEM.MISCINFO Configurations 103
IV.3 GOTOXY 104
IV.3.1 Writing Your Own GOTOXY 104
IV.3.2 A Recipe for GOTOXY 106
IV.3.3 Binding GOTOXY Using the Library Utility 108
IV.3.4 Problems 110
IV.4 SCREENTEST 111
IV.4.1 Running SCREENTEST 111
IV.4.2 Results of SCREENTEST 112
IV.4.3 Problems that can be Fixed by Altering SYSTEM.MISCINFO . . 113
IV.4.4 Problems that can be Fixed by Changing GOTOXY 115
IV.4.5 Other Problems 115
IV.4.6 Miscellaneous Notes on SCREENTEST Problems 116
IV.5 Creating Your Own Bootstrap 116
IV.5.1 The Concept of Booting 117
IV.5.2 Primary, Secondary, and Tertiary Bootstraps 117
IV.5.3 The Standard APPLE Bootstrap 118
IV.5.4 Example Primary Bootstrap Outline 119
IV.5.5 Placing a Primary Bootstrap on the Disk 120
IV.5.6 Creating SYSTEM.BOOT 121

CHAPTER FIVE
SPECIAL SOFTWARE FACILITIES 123
V.1 The APPLESTUFF Unit 123
V.2 The TURTLEGRAPHICS Unit 127
V.2.1 Installing and Initializing TURTLEGRAPHICS 128
V.2.2 TURTLEGRAPHICS Character Fonts 128
V.2.3 The Turtle 131
V.2.4 The Display 134

V.2.5 Writing Characters with TURTLEGRAPHICS 134
V.2.6 Scaling 135
V.2.7 Figures and the Port 136
V.2.8 Pixels 139
V.2.9 Fotofiles 139
V.2.10 Routine Parameters 141
V.2.11 Sample Program 142
V.2.12 Memory Space Requirements 145
V.3 The CONFIG Utility 146

APPENDICES:

APPENDIX A: THE p-SYSTEM DEVICES AND VOLUMES 154

APPENDIX B: APPLE II SLOT ASSIGNMENTS 155

APPENDIX C: GENERIC APPLE DRIVERS 156

APPENDIX D: BOOTING DIFFICULTIES 164

INDEX 167

SOFTECH MICROSYSTEMS

UCSD p-SYSTEM, VERSION IV

FOR THE APPLE II

The new UCSD p-System, Version IV, is now available to APPLE II users. The UCSD p-System features:

UCSD Pascal

The most popular Pascal available today.

BASIC

This popular language can now be compiled on the p-System.

System Software

The well known UCSD p-System's software allows powerful screen oriented editing, easy file handling, separate compilation, assembly language programming, plus many new features including concurrent processing, special debugging facilities, dynamic memory allocation, program chaining, and I/O redirection.

Applications Software

A growing set of applications is available to run with the Version IV p-System including special editors, text formatting, data base managing, and more.

Portability

Applications may be developed on the APPLE II with the p-System and transported to other major microcomputers, including the IBM Personal Computer.

Extended Peripherals

Peripheral drivers for various terminals, boards, printers, disk drives, and so on, are available with the Version IV UCSD p-System on the APPLE II (or you may write your own driver for whatever peripherals you have, including a clock). Drivers are supplied for the following:

APPLE Disk II mini-floppies
Micro-Sci A-40/70 mini-floppies
Sorrento Valley Associates Floppy Controller (with 8 inch drives)
Lowercase Video Adapters
Lowercase Keyboard Adapters
APPLE Communications Card
APPLE Serial Card
APPLE Parallel Card
California Computer Systems Serial Card
Solid State Music AIO Serial Card
Sup'r'terminal 80 Column Video Board
Videx Videoterm (80 Column Video Board)
Smarterm^{tm} 80 Column Video Board
Doublevision^{tm} 80 Column Video Board
D.C. Hayes Micromodem II^{tm}

In order to execute applications programs with the UCSD p-System, the APPLE II computer must have at least 64K of RAM and a minimum of 280K bytes of disk storage. A video display device is also required (for example, a monitor or TV console with converter).

Chapter I of this introductory manual describes how to boot the UCSD p-System and backup your system disks. It also describes some keyboard considerations.

Chapter II briefly describes various parts of the p-System such as the Screen Oriented Editor, the filer, and the compiler. It shows how to create and run a simple program. It also gives some suggestions for arranging your files on diskettes so that you may most easily use the p-System. The general reference for the p-System is the Users' Manual which accompanies this manual and the software.

Chapter III describes how to configure the p-System so that it will interface with any of the peripherals mentioned above. This involves creating a file called the SBIOS (Simple Basic Input Output Subsystem) by linking together the drivers that you require.

Chapter IV is much more lengthy and complex. It describes how you may write your own peripheral drivers and configure the p-System to interface with the particular peripherals you may have. If you have no need to reconfigure the p-System, you should skip this chapter.

Chapter V describes some software facilities that are available with the p-System on the APPLE II. These include graphics display facilities, support of special APPLE I/O capabilities, and so on.

CHAPTER I

GETTING STARTED

Now that you have an APPLE Computer and a copy of the UCSD p-System, you are probably wondering how to use them. Once you have become familiar with the APPLE Computer hardware, and are ready to use the p-System, you should read this manual FIRST.

In this chapter we are going to discuss the following topics:

Powering up with the p-System.

Backing up diskettes.

Important keyboard considerations.

Running the STARTUP diskette.

It is important that you read through this chapter. If you don't understand some of it, you may be in danger of LOSING programs. If you have programmed before, you know that means losing a lot of time and possibly money if you happen to destroy the diskettes on which your p-System is shipped.

I.1 Powering Up with the p-System

First, take a look at the loose pages that accompany this manual. They contain directory listings of the diskettes that you should have received with the p-System. Make sure that all of it is included with the p-System you received.

There are two diskettes that will "bootstrap" on the APPLE Computer; SYSTEM1: and STARTUP:. For now, we need only SYSTEM1:. "Bootstrapping" means bringing the p-System up. Most of this work is done by the system, hence the term (as in "pulling yourself up by the bootstraps").

In a standard configuration, a disk controller card should be placed in slot 6, and two disk drives should be connected as drives 1 and 2. Drive 1, in this configuration, is referred as #4:, and drive 2 as #5:. Put SYSTEM1: in drive #4:. The label should be facing up, and the oblong slot that exposes the diskette should be pointed away from you. Turn the power on and then close the disk drive door.

After a few moments, the p-System Command menu should appear. The menu is the outer level of the "operating system," and it shows most of the major p-System commands. For now, you don't need to worry about what they mean. You will get to see how some of them are used in the next section.

If you don't see a menu displayed after about a minute, then open the disk drive doors and turn off the APPLE Computer. Read over the preceding few paragraphs and make sure you did everything correctly. Be sure you are using the SYSTEM1: diskette! Then try it again.

If you continue to have trouble, stop right here and call your dealer. Resume at this point once you have bootstrapped your p-System. Or, if you like, read ahead while you are waiting for help, and learn what to expect.

All right, now you have bootstrapped your p-System. The next thing to do is play it safe and make backup copies of ALL your diskettes.

I.2 Backing Up Your Diskettes

I.2.1 Formatting a New Diskette

A brand new diskette must be formatted before you can use it on the APPLE Computer. This is done by running a small "utility program" called FORMATTER.CODE.

A "utility program" is a program that comes with the p-System, and is run like any user's program.

The p-System is shipped on several diskettes, and you must make a backup copy of each of them. The first step is to format the new diskettes onto which you will copy each of the original diskettes you recieved.

We assume that you have bootstrapped your p-System, and are looking at the Command menu. To run FORMATTER.CODE, place the diskette called UTILITY: in drive #5:, and press 'X' for X(ecute. The p-System will respond:

Execute what file ?

Now enter '#5:FORMATTER' and then press <return> key.

FORMATTER should respond with the following prompt:

Format which drive (4,5,9..12) ?

REMOVE both the SYSTEM1: diskette and the UTILITY: diskette. Put a **BLANK** diskette into drive #5:, and enter '5' followed by <return>. (If you press a plain <return>, without the 5, the Format utility will finish without doing anything more.)

FORMATTER will respond:

Format 35 tracks ?

The actual number may be different from 35, depending upon the type of drive you are using. If, at this point, you press 'N', for no, the Format utility will ask how many tracks to format. If you press 'Y', for yes, the formatting process will proceed.

If you have a blank disk in the drive, and if nothing goes wrong, the message:

Format complete

will appear, and the first menu of the Format utility will appear again. If, at this point you want to exit the Format utility, press <return>. If you want to format another diskette, follow the above steps again.

If the disk to be formatted has a valid p-System directory, the Format utility will prompt:

Destroy volume VOLNAME:

(VOLNAME: is the disk name.) Pressing 'Y' will allow the formatting process to continue. NOTE: This will destroy the information contained on the diskette. Pressing 'N' will return you to the first prompt of the Format utility.

If the Format utility gives you an error message while trying to format a diskette, there may be something wrong with the diskette. Errors occur if no disk is on-line, if the drive door is opened in the middle of the formatting process, if the disk is in the drive upside down, if there are bad blocks on the disk, and so on.

An error message also occurs if you try to format a diskette in any type of drive other than APPLE II or Micro-Sci mini-floppy drives. The formatter won't work on 8" disks.

When you have formatted all your diskettes, you may proceed to backup the p-System diskettes.

I.2.2 Making Backups

You should put SYSTEM1: (the disk you bootstrapped with) back into drive #4. Now press 'F' for F(iler. This command puts you at another "level" of the p-System, where the letters you press are now commands to the File Handler.

Press 'T' for T(ransfer. The filer will ask you:

Transfer what file?

Now REMOVE the system disk, place one of the p-System diskettes in drive #4 and one of the formatted blank disks in drive #5.

Answer the prompt with '#4,#5', and press <return> key.

The filer then asks:

Transfer xxx blocks ? (Y/N)

Enter 'Y'. (The "xxx blocks" should indicate the size of the entire diskette.) The filer then asks:

Destroy BLANK: ?

Enter 'Y'.

The filer will transfer the contents of the diskette in drive #4 onto the diskette in drive #5. When the filer is finished, a message to that effect is displayed.

The transfer is then complete. Remove both diskettes, label the new backup copy appropriately, press 'T' again, and repeat these steps until you have made backup copies of all your p-System diskettes.

WARNING: If you put the diskettes in backwards (for example, the blank diskette in drive #4 and the master diskette in drive #5), the filer will display a message such as:

Destroy SYSTEM1: ?

You should press 'N' or 'n', and correct the mistake.

You are now ready to use your copies for everyday work. Keep the diskettes that were originally shipped in a safe place as backups. You may want to make an additional set of backups for safety.

I.3 Important Keyboard Considerations

When you run your p-System on the Apple Computer, there are some important keys that you should know about. This section describes them briefly. You probably won't want to absorb all of the information here on first reading, but do read through the section, and refer back to it when you need this information.

NOTE: The CTRL key is used in a similar manner to the shift key on a typewriter in that it is pressed in combination with other keys. But, rather than shifting those keys to uppercase, it causes them to take on some completely different meaning. For example, holding down CTRL and pressing O, acts as an up arrow.

Once the p-System has been bootstrapped, it isn't necessary to turnoff the APPLE Computer in order to bootstrap it again. You can insert the diskette you want to bootstrap from in drive #4 (or leave it there if it is already there), and press <reset> key. The APPLE Computer will rebootstrap your p-System.

The <return> key is located on the right side of the keyboard. Whenever you answer a prompt in the p-System, except for single-letter commands, you should follow it with <return>. When you are using the editor, the <return> key acts as a carriage return on the screen.

When you are entering input to the p-System, you can correct errors by backspacing over them. The <backspace> key is the left pointing arrow.

Entering CTRL-X acts to rub out an entire line when answering p-System prompts.

The vector keys consist of the left arrow, the right arrow, CTRL-O (up arrow), and CTRL-L (down arrow). The vector keys are usually used within the p-System editor to move the cursor around, along with <return>, <space> (the standard space bar), <tab> (CTRL-I), and some other commands.

Another character you must use while in the editor is called <etx>. This is the character that accepts insertions and deletions. To send an <etx>, enter CTRL-C.

If you do NOT want to accept an insertion or deletion, you may press the ESC key (for "escape").

If the display you are using is 40 columns wide, then you can use CTRL-A to toggle from one side of the (80 column) screen to the other.

The screen can be put in a mode where it will shift, by 1/3 of the screen, as the cursor moves off the screen. The key that does this is initially set to CTRL-Z.

When a program is sending output, it is possible to stop and start that output by entering CTRL-S. The first time you enter CTRL-S, output stops, the next time you enter it, output begins again, and so forth.

If you no longer want the program to send any output, press CTRL-F. F stands for "flush." The program will continue to run, but won't send any output, and should soon terminate.

If something goes wrong while a program is running, and you wish to halt it completely, entering CTRL-@ causes a BREAK. This amounts to pressing CTRL, SHIFT, and P all at the same time. The p-System displays an error message, requests that you press <space>, and then reinitializes itself.

Normally, the Apple keyboard can only generate uppercase alphabetic characters. The p-System converts all of these uppercase codes to lowercase codes. Special key combinations are then provided to shift and alpha-lock characters entered at the keyboard. If your Apple keyboard has the capability of producing uppercase and lowercase characters, it is possible to turnoff the automatic lowercase conversion and to enable the regular Shift key so that it can be used with alphabetic characters. These and other related items are discussed in the remaining paragraphs of this section.

For keyboards which only produce uppercase characters, a "soft" shift key is provided. This key should be pressed and then released; the next alphabetical character entered will be shifted to uppercase. In addition, the following special characters will be shifted:

Key	Unshifted	Shifted
Control-K	[	{ (Displayed as [)
Shift-M	]	} (Displayed as])
Shift-N	^	_
/	/	\
Shift-P	@	| (Displayed as \)

The shift key is initially set to CTRL-W. This key will shift only the next character entered. If more than one shifted key is to be entered using this method, CTRL-W must be re-entered each time.

Pressing CTRL-R will act as an alphalock toggle. Alphabetical characters will toggle between uppercase and lowercase whenever this is entered.

If your keyboard will generate lowercase and uppercase alphabetic characters, then the CONFIG utility may be used to set HAS_LC_KEYBOARD to true. This will disable the automatic lowercase conversion as well as the soft shift key.

The standard APPLE II setup only recognizes the hard shift key (the actual key labeled "shift") for certain special characters. To enable the p-System to recognize the hard shift key in conjunction with alphabetical characters, you can install a "shift wire mod." This is a wire connecting the keyboard shift key to the game port. When this is done, use the CONFIG utility to set the HAS_SHIFT_WIRE bit to true. The following table shows how to produce the following shifted and unshifted special characters when the shift wire mod has been installed:

Key	Unshifted	Shifted
Control-K	[	{
Control-J	]	}
Control-N	^	_
/	/	\
Control-P	@	|

If your Apple system has an internal video with the capability to accept lowercase screen codes and display them, such as is provided by the Dan Paymar Adapter, you may use CONFIG to set the HAS_LC_VIDEO bit to true.

If your Apple system is setup so that the video display won't show lowercase, and you need to be able to distinguish uppercase from lowercase output, you may set a flag that will display all uppercase letters as inverse uppercase and lowercase as normal (not inverse) uppercase. To do this, use the CONFIG utility to set the USE_INVERSE_LC bit to true.

NOTE: The characters: "~" (Tilde) and "`" (Grave accent) can't be entered with the standard APPLE II keyboard (although they may be displayed on the screen if generated by a program). This means that if you find it necessary to use the utility SCREENTEST, you will be unable to satisfy its requirement of entering those characters. This is only because of hardware limitations and should be of no real concern. These characters can usually be generated if you are using an 80 column video board; you should consult your hardware documentation.

NOTE: If you are using an 80 column board with the general driver CONSOLE.CODE (as described in Chapter III), some of the keys and special characters described in this section may be used differently, or unsupported altogether, depending upon your hardware. The shift-next-character key (CTRL-W), and the alpha-lock key (CTRL-R) will actually be whatever the hardware manual for the board indicates. Also, the special characters "[", "]", "^", "{", "}", "_", "/", "\", "@", and "|" will be produced in the manner that the hardware manual indicates.

The following table is a summary of this section:

Key	Function
CTRL-X	Erase Line
CTRL-O	Up Arrow
CTRL-L	Down Arrow
CTRL-I	Tab
CTRL-C	<etx>
ESC	<esc>
CTRL-A	40 Column Screen Toggle
CTRL-Z	Screen Mode Toggle
CTRL-S	<stop>
CTRL-F	<flush>
CTRL-@	<break>
CTRL-W	Soft Shift Key
CTRL-R	Alpha-lock

I.4 Running the STARTUP: Diskette

If you have never used the p-System before, then we recommend that you become familiar with it by using Ken Bowles's Beginners' Guide for the UCSD p-System. This guide uses a number of programs which are supplied on the diskette labeled STARTUP:.

STARTUP: can be bootstrapped. It contains a p-System with an editor, but not a compiler. To use a compiler, see Chapter II in this manual. STARTUP: also contains the following files:

NAMEFILE
SCDEMO.CODE
COPYSCUNIT.CODE
UPDATE.CODE
COMPDEMO.TEXT
EDITDEMO.TEXT
UPDATE.TEXT

The use of these files is explained in the Beginners' Guide.

When you bootstrap on the STARTUP: diskette, you will find yourself in the middle of a program. This is a demonstration program that is meant to give you some of the feel for using the p-System. The Beginners' Guide tells you how to use it and what to expect.

CHAPTER II

MAKING USE OF THE p-SYSTEM

This chapter should be more interesting than the previous chapter, since it describes the major components of the p-System, and some of the more effective ways to use them.

II.1 Software Components

What is the p-System, anyway? It's a collection of software components that come in the form of files saved on diskettes. This software is used for writing and running programs. The programs you write will use the Apple Computer's hardware in various ways.

When you use the p-System, you begin by bootstrapping it. When you have bootstrapped, you see a Command menu that looks like this:

Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, ?

Each capital letter shown on the Command menu is a command to the p-System. Entering one of these letters causes something to happen. Some of the commands, such as E(dit and F(iler, call other programs that in turn have their own menus and their own set of commands.

In other words, each component of the p-System serves a particular purpose. All of them are fairly self-explanatory, and all of them are meant to be used by a single user sitting in front of the Apple Computer display (in other words, "interactively").

This section gives some brief descriptions of the major components of the p-System.

II.1.1 Operating System

Very little of the operating system is actually seen by you. The operating system is essentially the program that controls the Apple Computer's resources and calls various other programs.

The part of the operating system that IS important to you is the Command menu; the one you see when the p-System is first bootstrapped. This set of commands allows you to call other portions of the p-System, and to run programs that you have written yourself.

The operating system is called SYSTEM.PASCAL. It can't run unless the files SYSTEM.MISCINFO, SYSTEM.INTERP, SYSTEM.BOOT, and SYSTEM.SBIOS are also on the disk that you bootstrap. These files all appear on the diskettes STARTUP: and SYSTEM1:, which are shipped with the p-System. Either of these diskettes can bootstrap the p-System, when placed in the #4: disk drive, as described in Chapter I.

The operating system and its commands are described in the Users' Manual.

II.1.2 Filer

You have already used the filer briefly to make backup copies of your p-System diskettes. The filer is used for maintaining the collection of files on a diskette, and transferring them from one location to another.

SYSTEM.FILER is the filer program. It is shipped on STARTUP: and SYSTEM1:.

The use of the filer is described in the Users' Manual.

II.1.3 Editor

The editor allows you to create new text files such as programs or documents. It also allows you to modify old text files. The editor makes use of the entire display screen, so it is easy to see the text you are working on, and modify it as necessary.

The editor is called SYSTEM.EDITOR, and it is also shipped on STARTUP: and SYSTEM1:.

The editor is described in the Users' Manual.

II.1.4 Compilers

With the UCSD p-System on the Apple Computer, you can write programs in UCSD Pascal or in BASIC. Programming in one of these languages entails creating a text file with the editor, and then "compiling" that text file by calling the appropriate compiler. The compiler translates the program text into a form that the p-System can execute on the Apple Computer.

When you compile a program, the name of the compiler must be SYSTEM.COMPILER. The p-System is initially setup with the Pascal compiler named SYSTEM.COMPILER. If you intend to use the BASIC compiler you must use the filer to change the name of SYSTEM.COMPILER to PASCAL.CODE (or whatever other distinct name you wish), and the name of BASIC.CODE to SYSTEM.COMPILER.

Also, when you compile a UCSD Pascal program, the file SYSTEM.SYNTAX should be on the system disk (the diskette you bootstrap with). This file contains all the error messages that the Pascal compiler needs if it encounters a syntax error while compiling your program. SYSTEM.SYNTAX is shipped on SYSTEM1:.

The UCSD Pascal language is described in the UCSD Pascal Handbook.

To compile a BASIC program, BASIC.CODE must be changed to SYSTEM.COMPILER (as described above), and BASIC.LIBRARY must be changed to SYSTEM.LIBRARY and placed on the boot disk (after changing the name of the standard SYSTEM.LIBRARY). Perhaps, a better alternative is to create an entirely separate bootable disk which is dedicated to BASIC and which contains the properly name compiler and library. We will say more about this in the section below on "Ways To Configure Your Diskettes."

BASIC doesn't have a file that corresponds to Pascal's SYSTEM.SYNTAX.

Information on the BASIC language, and using the LIB file that accompanies the BASIC compiler, may be found in the BASIC Manual.

It is also possible to write programs to be executed directly by the Apple Computer's 6502 microprocessor. These programs are said to be "assembled" rather than compiled, and the p-System program that does this is SYSTEM.ASSMBLER.

SYSTEM.ASSMBLER is shipped along with the files 6500.OPCODES and 6500.ERRORS. These two files must be present whenever the assembler is run.

We expect that only programmers already familiar with assembly language will attempt to use SYSTEM.ASSMBLER, and then only when they have a good reason to. It is far easier to program in a "high-level" language such as UCSD Pascal.

When you write a UCSD Pascal or BASIC program that calls assembly language routines, you will need to use the linker program, which is called SYSTEM.LINKER. The linker is described in the Users' Manual.

II.1.5 Libraries

Frequently, a number of programs will need to perform the same operations. It isn't necessary to "reinvent the wheel" and write the same code over for every program that needs it. You can write a portion of code called a "UNIT," and compile it separately from the programs that use it. A disk file that contains one or more UNITs for use by several programs is called a "library."

The most important library is called SYSTEM.LIBRARY, and such a file appears on the diskette SYSTEM1:. SYSTEM.LIBRARY contains the units Longops and Applestuff. If you have purchased the Turtlegraphics package, you may wish to install the appropriate Turtlegraphics routines into SYSTEM.LIBRARY (see Section V.2.1).

II.1.6 Utilities

Utilities are programs, and they are run in just the same way as your programs. They are shipped with the p-System and, in general, they provide important services, but aren't used as frequently as the programs (like the filer, editor, or compiler) that are called directly from the operating system.

Most of the p-System utilities are shipped on the UTILITY: diskette. They are described in the Users' Manual and in Chapter V of this manual.

II.1.7 The Booter Utility

The booter utility transfers a bootstrap from one disk to another. Normally, this utility shouldn't be used on the Apple II. To copy a bootstrap, simply copy an entire bootable disk onto a blank disk as described in Section I.2, "Backing Up Your Diskettes." If you just want to copy the bootstrap (without altering the directory or any of the files), T(ransfer 2 blocks, not the entire disk.

If you use a disk format where track 1 is the first p-System track, you should use the booter utility to copy bootstraps. The code for the booter utility is on the utilities disk under the name BOOTER.CODE. Execute BOOTER.CODE to copy a bootstrap.

The booter utility asks for two disks names and copies all of track 0 from the input disk to the output disk.

II.1.8 Disk Utilities

DISKCHANGE and DISKSIZE are two utilities that are provided with all adaptable systems. DISKCHANGE changes disk formats. DISKSIZE changes the capacity of a disk as it is recorded in the p-System volume directory.

Changing disk formats is chiefly done in two situations:

1. When a different format allows the system's disk hardware to function more efficiently; and
2. When you wish to use p-System disks that have been prepared on different hardware using a different disk format.

II.1.9 Using DISKCHANGE

A floppy disk's "interleaving" is the ordering of sectors within a track. For example, an interleaving ratio of one (that is, 1:1) means that logical sectors 1, 2, 3, and so on are stored in physical sectors 1, 2, 3, and so on, while an interleaving ratio of two (that is, 2:1) means that logical sectors 1, 2, 3, and so on are stored in physical sectors, 1, 3, 5, and so on, and 2, 4, and so on.

A floppy drive's "skew" is the offset when moving from one track to the next. For example, with one-to-one interleaving, a skew of zero means that sector 1 on one track is adjacent to sector 1 on the next track; skew of 6 means that sector 1 on one track is adjacent to sector 7 on the next, and so forth.

Interleaving and skew are characteristics of a disk format, not of a drive, but for each particular drive, there is a combination of interleaving and skew that is the most efficient, and results in faster disk performance. Some drives require a bit of "tuning" of disk formats, to determine what combinaton of interleaving and skew yields the fastest disk access. The utility FINDPARAMS that is supplied with adaptable systems is provided to help determine optimal values for interleaving and skew.

The utility DISKCHANGE allows a disk's interleaving and skew to be altered.

DISKCHANGE is supplied on the utilities disk that comes with your system. To run it, X(ecute 'DISKCHANGE'.

After a single run of DISKCHANGE, the screen will look something like this (underlined portions are user responses):

```
FLOPPY INTERLEAVING CHANGER    [B3]

      Type '!' to exit

      What is the source unit number? (4,5,9..12)  4
      What is the destination unit number? (4,5,9..12)  5
      What is the interleave ratio of the drives used for the transfer?  2

 SOURCE DISK TYPE:

      What is the interleaving ratio?  1
      What is the sector skew per track?  0
      What is the first interleaved track?  1

 DESTINATION DISK TYPE:

      What is the interleaving ration?  2
      What is the sector skew per track?  6
      What is the first interleaved track?  1

      Insert source disk in drive #4, dest disk in drive #5, and press return

      Insert system disk and press return
```

At any point, entering an exclamation point ('!') will abort the program.

The first two prompts asks which disk will be transferred to which disk.

NOTE: It's possible to transfer a disk to itself but backup the disk first as a precaution against losing it.

The next prompt asks for the interleaving of the drives (that is, the optimal interleaving for the drives you are using). This prompt is repeated if the program can't use the answer you give. This value only affects the speed at which DISKCHANGE operate.

For both source and destination disk, there are three prompts. Interleaving and skew you will have to determine yourself; the track virtually all p-System disks start on is track one (refer to the figures, and the following two sections).

The Apple II p-System disks come with two-to-one interleaving and zero skew. Bootstrap the system without changing these values; once you are able to run DISKCHANGE, it's safe to change them. However, this format has been choosen for optimal efficiency and compatibility with Apple Pascal.

When DISKCHANGE displays the final prompt of 'Insert system disk and press return', pressing 'R' instead of return causes a transfer to be done again with the same parameters. That is, when you press 'R' after the last prompt, DISKCHANGE again displays:

Insert source disk in drive #4, dest disk in drive #5, and press return

(or whatever drives were named).

You can't change the parameters, however, without finishing a DISKCHANGE run and starting over again.

II.1.10 Using DISKSIZE

When DISKCHANGE is used to unpack a disk by transferring 1 5-1/4" adaptable system disk image (1/2 of an 8" disk) onto an 8" disk, the new 8" disk directory will still indicate a small disk size (240 blocks). Eight-inch disks can hold about 494 blocks (the exact figure dpends on the hardware). To access the entire disk, use DISKSIZE to change the recorded size of the new disk.

When you X(ecute 'DISKSIZE', it prompts you for the number of a disk drive. It then asks for the number of blocks that the disk can hold. It then records this number of your disk.

Calculate the number of blocks available on the disks by using the bootstrap parameters for the system. Use the following formula.

```
    ( number of tracks per disk - first Pascal track )
  * ( number of sectors per track )
DIV ( 512 DIV number of bytes per sector)
```

II.2 How to Create and Run a Simple Program

In this section, we will "walk you through" creating a simple program, compiling it, and running it.

BASIC programmers note: this sample uses a brief Pascal program. What we are really trying to illustrate is the use of the editor to create a work file, the use of the R(un command to compile and execute it, and the use of the filer to change its name. These aspects of the p-System are the same regardless of which language you are using.

First, bootstrap the p-System if you haven't already done so. We assume that you are a new user who hasn't yet created a work file. If you do have a work file, you should save it before going through this demonstration.

Now press 'E' for E(dit. You should get a display that looks like this:

```
>Edit:
No workfile is present. File? ( <ret> for no file )
```

Since we are creating a new program, press <return> to continue.

Now press 'I' for I(nsert, and enter the following text as shown. To end a line, press <return> key (as you would use the carriage return on a typewriter). Notice that once you have indented a line, the lines that follow it are indented automatically:

```
PROGRAM EASY;
  BEGINN
    WRITELN('HELLO, THERE!');
    WRITELN;
    WRITELN('YOU HAVE JUST RUN YOUR FIRST PROGRAM.');
  END.
```

If you make a mistake while entering, <backspace> over it and re-enter the correct characters. When you have finished entering in the program, enter CTRL-C (you must hold down the CTRL key and then press 'C'). This should get you out of I(nsert, and back to the editor's main menu.

But the program we had you enter contains an error (if you didn't enter it just the way it's shown here, it may contain more than one!). The 'BEGIN' in the second line should contain one "N," not two.

To remove this extra letter, move the cursor (that's the movable block character) back to the FIRST "N" by using the "arrow keys." These consist of the left arrow, the right arrow, CTRL-O for up, and CTRL-L for down. Also <return>, <space>, and <tab> (CTRL-I) may be used. Try moving around your program until you get the feel for it.

Now, is the cursor at the first "N" in the second line? Good. To remove the "N," press 'D' for D(elete, then press a space. This should delete the first "N." Then enter CTRL-C, which should close up the line and return you to the main editor prompt. If this worked, we can go on.

Press 'Q' for Q(uit, and you will see a prompt that looks like this:

```
>Quit:
      U(pdate the workfile and leave
      E(xit without updating
      R(eturn to the editor without updating
      W(rite to a filename and return
```

Press 'U' for U(pdate. What this does is create a temporary file called SYSTEM.WRK.TEXT. This file is on your system disk, and it contains the text that you just entered. It is referred to as the "work file," and a little later we will show you how to save it under a different name.

Now that we have a program, we want to run it. But we have to compile it first. To do so, take your working copy of the disk SYSTEM2:, and put it in drive #5:. Press 'R' for R(un. The operating system will recognize that your work file hasn't been compiled, and so it will call the compiler SYSTEM.COMPILER that is on the disk in drive #5. The compiler will display the message:

Output file for compiled listing? (<cr> for none)

Either press the carriage return for no compiled listing or enter in a file name for the listing file. When the compiler is finished, the operating system will run your program. While the compiler is compiling, it displays some progress information:

```
Pascal compiler - release level IV.1 c65-2
<     0>..
EASY
<     2>...
     5 lines compiled

EASY        .
```

and when it's finished, it should run your program, creating some output that looks like this:

```
Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, ?
Running...
HELLO, THERE!

YOU HAVE JUST RUN YOUR FIRST PROGRAM.
```

The compiler creates another portion of the work file called SYSTEM.WRK.CODE. As long as SYSTEM.WRK.CODE exists, using the R(un command will run your program over again without recompiling it.

(There is little difference between the expressions "running a program" and "executing a program." But there is a large difference between the operating system command R(un and the operating system command X(ecute, as you will discover if you read Users' Manual.)

If there had been a "bug" or mistake in the program you were compiling (and there might be, if you entered it differently from the program in this booklet), the compiler would display a message something like this:

```
Pascal compiler - release level IV.0 c65-2
<     0>..
EASY
<     2>..
      WRITELN;
      WRITELN('YOU'VE <--
Illegal symbol (terminator expected)
Line 5
Type <sp> to continue, <esc> to terminate, or 'e' to edit
```

When the compiler tells you there is an error, there are three things you can enter:

ESC ends the compilation;

<space> or <return> continues the compilation and allows you to see what other bugs the compiler might report;

'E' or 'e' gets you back into the editor and allows you to fix the bug on the spot.

Now, we could show you more about the editor, but the Users' Manual does a good job of that. So for now, we will just save the work file that you have created.

Go back into the filer (by pressing 'F' at the Command menu). Now press 'S' for S(ave. The filer will ask you:

Save as what file ?

Enter the file name 'TESTING', and then press <return>. When the work file has been saved, we get this message:

```
Filer: G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng, T(rans
Text file saved & Code file saved
```

The files SYSTEM.WRK.TEXT and SYSTEM.WRK.CODE are renamed TESTING.TEXT and TESTING.CODE, respectively. These files are now your current work file.

Note that we didn't need to enter ".TEXT" or ".CODE". The filer supplies these suffixes. This is a convenient shortcut, but in the filer it only applies to the work file commands G(et and S(ave. For any other filer command, you must enter the entire file name. (The filer does have a "wild card" facility which can be used to specify files without entering the entire file name. This is described in the Users' Manual.)

Now that the TESTING files are considered the work file, we can go back into the editor (you should also be able to do this by now), and the editor will automatically read in TESTING.TEXT for you to edit.

When you want to get rid of the TESTING files, go into the filer and press 'R' for R(emove. The filer will ask:

```
Remove what file ?
```

Enter 'TESTING.TEXT'<return>. The display will now look like this:

```
Remove what file ? TESTING.TEXT
SYSTEM1:TESTING.TEXT        —> removed
Update directory ?
```

If you are certain that you want to get rid of TESTING.TEXT, press 'Y' or 'y'. Any other character will leave your disk unchanged.

You can remove TESTING.CODE in the same manner. There is also a shortcut, by which you can remove both TESTING files at once. You can discover what this is by looking at the section on "wild cards" in the Users' Manual.

II.3 Ways To Configure Your Diskettes

The entire p-System won't fit on a single diskette. It is shipped on the diskettes listed on the loose pages that accompany this manual. While you are using your p-System, you will want to create a set of diskettes that are convenient for you to work with. The files on these diskettes (both p-System files and the files you create) should be arranged so that you can do your work without too much "shuffling" of diskettes in and out of the drives.

For this reason, none of your diskettes should be "packed" with files. It is often convenient to save a file on the nearest available diskette, and leaving some space on all your diskettes allows you to do this.

We recommend that you arrange your p-System in the following way:

1. A system disk for bootstrapping,
2. A language disk for preparing programs,
3. A utility disk for frequently-used utility programs,
4. Any number of disks that contain your own work.

Here is an example:

```
SYSTEM:
   SYSTEM.PASCAL
   SYSTEM.MISCINFO
   SYSTEM.INTERP
   SYSTEM.SBIOS
   SYSTEM.BOOT
   SYSTEM.SYNTAX
   SYSTEM.LIBRARY
   SYSTEM.FILER
   SYSTEM.EDITOR

PASCAL:
   SYSTEM.COMPILER
   LIBRARY.CODE

BASIC:
   SYSTEM.COMPILER
   LIBRARY.CODE

ASSEM:
   SYSTEM.ASSMBLER
   6500.OPCODES
   6500.ERRORS
   SYSTEM.LINKER
   COMPRESS.CODE
```

UTILS:
  FORMATTER.CODE
  COPYDUPDIR.CODE
  MARKDUPDIR.CODE
  RECOVER.CODE
  PATCH.CODE
  DECODE.CODE
  XREF.CODE
  CONFIG.CODE

The first thing to create is a diskette that will bootstrap the p-System. We often refer to this simply as the "system disk." There are certain files that MUST be on the system disk. These are:

SYSTEM.PASCAL	{the operating system}
SYSTEM.MISCINFO	{configuration data}
SYSTEM.INTERP	{the interpreter}
SYSTEM.SBIOS	{the SBIOS routines}
SYSTEM.BOOT	{the Secondary and Tertiary Bootstraps}
SYSTEM.LIBRARY	{the main library}

A system disk must also contain a Primary Bootstrap. This invisible bootstrap code is located at the beginning of the diskette, before all other files. It is transferred to another diskette when you transfer an ENTIRE diskette onto another diskette by using the T(ransfer command in the filer.

Thus, if you want to create a system disk that has a different assortment of files than the system disks that we ship, T(ransfer a system disk onto a new diskette. Change the name of the new disk if you wish (using the C(hange command of the filer.) Then R(emove those files you don't want from the new diskette, and T(ransfer other files that you do want onto the new diskette (one at a time). Filer commands are described in the Users' Manual. The filer is also discussed in Ken Bowles's Beginners' Guide.

When you create a new system disk, remember two things:

1. You have to use FORMATTER on a a brand new disk before you can use it; and
2. The files listed above MUST be on the system disk.

If the file SYSTEM.SYNTAX is on your system disk when you compile a UCSD Pascal program, the compiler will produce full error messages (when necessary). You can save some space by leaving this file off your system disk, but then the Pascal compiler will only provide error numbers, which you must look up (they are listed in an appendix to the UCSD Pascal Handbook).

Once you have a system disk that suits your requirements, it is easy to go on to other things.

One "secret" to creating your set of diskettes was illustrated in the previous section. When we called the compiler by using the operating system's R(un command, the file SYSTEM.COMPILER was on the diskette in drive #5, and NOT on the system disk.

Operating system commands like F(iler, E(dit, and C(ompile call programs that have names like SYSTEM.FILER, SYSTEM.EDITOR, and SYSTEM.COMPILER. When you use a command that requires one of these SYSTEM.xxx files, the file itself does NOT need to be on the system disk. It can be on a diskette in any drive except, of course, for the files already mentioned that must be on the system disk. This gives you some flexibility when you decide which files to place on a certain diskette.

With this in mind, you can create a diskette for each of the languages you plan to be using. Notice that in the example, we have named the compiler SYSTEM.COMPILER on both the PASCAL: and the BASIC: diskettes. The files are different, but their names have been made the same for convenience. To compile a program in UCSD Pascal, place the diskette PASCAL: in drive #5, and use the C(ompile command or the R(un command. To compile a BASIC program, place BASIC: in drive #5, and use C(ompile or R(un in just the same way.

To run a BASIC program, the file SYSTEM.LIBRARY on SYSTEM: must contain certain portions of program code ("code segments"). Although the example doesn't show this, if you are going to use BASIC frequently, the SYSTEM.LIBRARY on the SYSTEM: disk should be a combination of the original SYSTEM.LIBRARY plus the code segments in BASIC.LIBRARY. The utility LIBRARY is used to combine libraries in this way. See the BASIC Manual and the Users' Manual for more details.

Since the language diskettes, as we have shown them, still contain a lot of room, this is a good area to keep work-in-progress. This could consist of programs that aren't yet completed. It could also consist of portions of larger programs, where the entire program is stored on some other diskette.

As you become more experienced with the p-System, you may find other arrangements that work better for your particular style of programming, or your particular use of the Apple Computer. The rule of thumb is, do what works best for you. In this section, we have made suggestions that should be generally useful. We have also pointed out some requirements of the p-System that apply regardless of what diskette configuration you eventually choose.

II.4 Large Programs

This section is a little more advanced than the previous sections, and you may want to skim it on first reading. But, it is probably a good idea just to read through it in order to get an idea of some of the p-System's facilities.

As you become more experienced with the p-System, it is likely that the programs you write will become larger and more sophisticated. The larger a program is, the more cumbersome it is to keep it in a single file. Also, several large programs may use identical portions of code. On the p-System, UCSD Pascal and BASIC allow you to compile a program in separate portions, called "units." These units can be located in files of their own, as we will explain below. They can also be used by separate programs.

A unit is a package of routines (for example, PROCEDUREs, FUNCTIONs, SUBROUTINEs, and so on), together with any appropriate variable and type declarations global to the routines. Any number of programs may "use" a unit. Using a unit means that the program can use the unit's declarations and call the unit's routines, just as if those declarations and routines had been written into the program itself.

Much of a unit is invisible to the program that uses it. The program doesn't know how the unit implements the routines that the program can use. In fact, the unit may contain other declarations and routines that the program doesn't know about at all.

The advantage of this is modularity. It is possible to recompile a unit—perhaps to fix a bug or improve an algorithm. If the unit's "interface" declarations (the ones that another program may use) aren't changed, then it's NOT necessary to recompile any programs that use the unit. It should be evident that this can save a lot of time.

It is also possible for a unit to use another unit.

A code file may contain a single program, a single unit, or a number of units. A code file that contains a number of units is called a "library." Libraries are especially useful for organizing the routines of a large program (especially one that involves several programmers), or organizing a number of general-purpose routines that many programs will use.

The library you have already encountered is SYSTEM.LIBRARY. This contains general-purpose routines for handling the Apple Computer hardware. If you are using BASIC, SYSTEM.LIBRARY should also contain routines that the p-System must use when BASIC programs are running.

SYSTEM.LIBRARY must reside on the system disk (unless the USERLIB facility is used to specify a SYSTEM.LIBRARY on another disk, see below). But it is possible to create any number of libraries of your own, and they may reside on any diskette.

When a text file that consists of a number of units is compiled, a library is created. It is also possible to create a library from several code files by using the utility LIBRARY. LIBRARY can also be used to maintain library files that have already been created.

If the unit your program uses isn't in SYSTEM.LIBRARY, then the program must declare which file contains the unit. This file name must also appear in a file called a "library text file." This is simply a text file that contains names of user's library files. The default library text file is called USERLIB.TEXT, and must reside on the system disk.

When a program is run, the p-System searches for each unit that the program uses. First it searches the files named in USERLIB.TEXT, and then the units in SYSTEM.LIBRARY. If it can't find the unit at all, it gives an error message.

It is actually possible to have more than one library text file. When you X(ecute a program, you can specify which library text file the p-System is to use (this is done with an "execution option string"; see the Users' Manual). Since the files named in a library text file are searched in the order they appear, it is possible to arrange different library text files that are more efficient for running certain programs.

All these facilities make it possible to break a large program into smaller pieces. Many of the pieces can be made general-purpose, which saves much effort when you are developing programs at a later time. By breaking down a large program in this way, you can make it easier to debug and faster to compile, and avoid running out of space on your diskettes.

II.5 Binding the Debugger into SYSTEM.PASCAL

A large unit of the operating system, the Pascal debugger (described in the Users' Manual), is initially supplied as a separate code file on the UNITS: diskette. If the debugger is to be used, it must be bound into SYSTEM.PASCAL (the operating system). The reason it is provided separately is that this makes SYSTEM.PASCAL smaller, and increases the amount of available disk space on the boot disk.

In order to bind the debugger into SYSTEM.PASCAL, the library facility is used. First, enter the filer and T(ransfer 'UNITS:DEBUGGER.CODE' onto the boot disk. Then return to the p-System Command menu and place the UTILITY: diskette into drive #5:. X(ecute '#5:LIBRARY'.

You need to have a disk around with enough contiguous memory space on it to hold the new SYSTEM.PASCAL. When library prompts for an output file, place that disk into drive #5:, and answer '#5:NEW.PASCAL'.

When library requests an input file, specify 'DEBUGGER' followed by <return>. The debugger code file actually consists of two code segments: Debugger and Extralex. These will be displayed.

At this point, press 'T', which will put the debugger into a mode where the unnecessary interface text won't be copied into the new operating system.

Then press '0' to select slot zero (the debugger segment) followed by <space>. Enter a number greater than 15 (21 is standard), to select the destination slot, followed by <space>. This will move the first segment over. Repeat this process as follows: press '1', <space>, enter another number greater than 15 (22 is standard), <space>.

Now press 'N' for a new input file. Specify '*SYSTEM.PASCAL', followed by <return>. Now you may simply press 'E', and every remaining operating system segment will be copied into the file you are creating.

Finally, press 'Q', for quit, followed by <return> when asked for a "Notice."

You may now place NEW.PASCAL onto a boot disk as SYSTEM.PASCAL using the T(ransfer command of the filer (or the C(hange command if NEW.PASCAL is already on the desired boot disk). Make sure there is enough room on the boot disk to hold the new SYSTEM.PASCAL. Also, make sure that the boot disk has all of the required files mentioned in Section II.3, including the invisible bootstrap code.

If everything has been performed properly, the new disk should boot, and the debugging facility should be installed.

NOTE: Be certain that you have at least one backup bootable disk that is completely separate from this entire process of binding the debugger into SYSTEM.PASCAL. This will insure that you are able to reboot if something goes wrong.

CHAPTER III

CONFIGURING YOUR APPLE II SYSTEM

USING SUPPLIED PERIPHERAL DRIVERS

The APPLE II Computer has at least a hundred peripheral devices that are available to users. The UCSD p-System provides a very good way to interface with the particular peripherals that you may have.

The collection of hardware driver routines within the p-System is called the SBIOS (Simplified Basic Input Ouput Subsystem). Initially, the SBIOS is setup to interface with the standard APPLE II hardware (keyboard, 40 column display, printer, serial line, and APPLE II disk drives). However, each of these driver routines are provided individually, along with several other individual driver routines. Any of these routines may be linked together to form a new SBIOS which interfaces with different peripherals. (It is also possible to write your own driver routines and link them into an SBIOS. See Chapter IV.)

This chapter describes how to link an SBIOS together.

III.I Reconfiguration Summary

The following steps are necessary to completely reconfigure a system properly and have all parts of the system work properly with any hardware change. Some steps may not be needed if the change you are making doesn't affect that step.

1. Check to see if the standard SBIOS drivers will support the hardware you have changed to. If you require a different driver in your SBIOS you will have to relink the SYSTEM.SBIOS as described in Section III.2 of this manual.

2. X(ecute the CONFIG utility and set the parameters as necessary. This is described in Section V.3 of this manual.

3. Relink the SYSTEM.INTERP to use the appropriate BIOS as described in Section IV.1.13 of this manual. (This is not always needed.)

4. Install the appropriate GOTOXY into the operating system (SYSTEM.PASCAL) as described in Section IV.3. This step is only needed if you are going to use an external terminal such as a SOROC that has a different gotoxy.

5. X(ecute the SETUP utility and set any parameters that have changed in SYSTEM.MISCINFO. This is usually only needed if you are adding an external terminal and is described in Section IV.2.

6. If each of the above steps are done correctly, then you should be able to boot the new system disk with the new hardware configuration.

III.2 Supplied Drivers

The following code files contain the peripheral drivers supplied with the UCSD p-System.

NOTE: DRIVERS.CODE linked with CONSOLE.CODE supports Sup-r-term, Smarterm, and Videx 80 column boards, as well as most other 80 column boards for the Apple II (except Doublevision).

SBIOS.CODE	SBIOS jump table and disk driver interface
DISKII.CODE	Driver for the APPLE/Micro-Sci mini-floppies
CONSOLE.CODE	Jump table for console routines
PRINTER.CODE	Jump table for printer routines
REMOTE.CODE	Jump table for remote routines
PATCHCARD.CODE	Patches CONSOLE.CODE, PRINTER.CODE, or REMOTE.CODE to use FIRMCARD.CODE, SERCARD.CODE, PRLCARD.CODE, COMMCARD.CODE, or INTCARD.CODE depending on the type of card in a certain slot
INTCARD.CODE	Internal console driver: the jump table in CONSOLE.CODE will be patched (using PATCHCARD.CODE) to this code if a card is not recognized in slot 3
COMMCARD.CODE	Communication card driver: the jump table in CONSOLE.CODE, PRINTER.CODE, or REMOTE.CODE will be patched with routines in this code if a communication card is recognized in a selected slot. ("Selected" signifies CONSOLE.CODE, PRINTER.CODE, or REMOTE.CODE was linked into your SYSTEM.BIOS
SERCARD.CODE	Serial card driver: used if a serial card is recognized in a selected slot
FIRMCARD.CODE	Firmware card driver: used if a firmware card is recognized in a selected slot
PRLCARD.CODE	Parallel card driver: used if a parallel card driver is recognized in the printer slot and PRINTER.CODE has been selected
EMPTY.INT.CODE	Stub for the internal console driver, should be used if space is needed for SYSTEM.SBIOS and the internal console driver is not needed
EMPTY.COMM.CODE	Stub for COMMCARD.CODE
EMPTY.FIRM.CODE	Stub for FIRMCARD.CODE

EMPTY.SER.CODE	Stub for SERCARD.CODE
EMPTY.PRL.CODE	Stub for PRLCARD.CODE
INTERRUPTS.CODE	Contains the SBIOS routines Quite, Enable, and Event
DRIVERS.CODE	Contains the LIBRARYed files PATCHCARD.CODE, INTCARD.CODE, COMMCARD.CODE, SERCARD.CODE, FIRMCARD.CODE, and PRLCARD.CODE

NOTE: EMPTY.CON.CODE and EMPTY.PRUC.CODE aren't listed. They are stubs for the console and printer/remote/user/clock, respectively. They should only be used when writing your own bootstrap (see Chapter IV).

III.3 Linking Together an SBIOS

In order to link these files together into an SBIOS, the p-System linker is used. The following example shows how the standard SBIOS is created:

First, put the disk volume CODE: in one of your drives and use the filer to P(refix to that volume. Call the linker by pressing 'L' from the Command menu. Then answer the following prompts with the underlined responses:

```
Host file: SBIOS
Lib File: DISKII
Lib File: DISKVECT
Lib File: CONSOLE
Lib File: KEYBOARD
Lib File: PRINTER
Lib File: REMOTE
Lib File: DRIVERS
Lib File: INTERRUPTS
Lib File: EMPTY.USR
Lib File: EMPTY.CLK
Lib File: ENDMARK
Lib File: <return>
Map File: <return>
Output File? SYSTEM.SBIOS
```

The DISKII routine must be the first Lib File linked. The rest of the routines may be linked in any order. ENDMARK should always be the last file.

The file should now be compressed by executing UTILITY:COMPRESS and making SYSTEM.SBIOS a relocatable file with the same name. The last address of the last PROC listed by COMPRESS must not exceed FFFH. If it does, it will extend past the space reserved for the SBIOS in the language card and be truncated at FFFH. All isn't lost. Using CONFIG with the desired cards in slots 1, 2, 3 you can determine which drivers you need to link into the SBIOS. The other drivers can be replaced with their appropriate stub. For example, after executing CONFIG and doing a G(et from M(emory, I(dent will display the following:

Slot	Device	Card Type
1	Printer:	PRL
2	Remote:	SER
3	Console:	FIRM

This means a parallel driver is needed for the printer, a serial driver for remote and a firmware driver for the console. The following SBIOS will work for this configuration.

```
Host File: SBIOS
Lib File: DISKII
Lib File: DISKVECT
Lib File: CONSOLE
Lib File: KEYBOARD
Lib File: PRINTER
Lib File: REMOTE
Lib File: PATCHCARD
Lib File: EMPTY.INT
Lib File: PRLCARD
Lib File: SERCARD
Lib File: FIRMCARD
Lib File: EMPTY.COMM
Lib File: INTERRUPTS
Lib File: EMPTY.USR
Lib File: EMPTY.CLK
Lib File: ENDMARK
Lib File: <return>
Map File: <return>
Output File: SYSTEM.SBIOS
```

The SBIOS has been reduced in size because stubs have replaced INTCARD.CODE and COMMCARD.CODE.

Then use the filer to T(ransfer SYSTEM.SBIOS to your boot disk. Be sure you have at least one bootable backup disk so that you may reboot if something goes wrong.

If, as an example, you want to use the Sup-r-term 80 column video board instead of the general console, replace CONSOLE with SUPRTERM. If, in this case, you also decide not to use PRINTER: or REMOTE: you eliminate the need for DRIVERS, or any combination that uses PATCHCARD.

All of the listed files, or appropriate substitutes, must be linked into the SBIOS. If, for example, you don't have a printer, you may link EMPTY.PRN in place of PRINTER into the SBIOS. But, don't omit a printer file all together.

NOTE: You shouldn't call the output file SYSTEM.SBIOS unless you have backed up the original SYSTEM.SBIOS. The p-System will remove any existing file which has the same name that you are giving to the linker output file. Somewhere you should have at least one backup bootable disk with SYSTEM.SBIOS on it so that you may reboot if something goes wrong during the creation of a new SBIOS.

III.4 GOTOXY and SYSTEM.MISCINFO

GOTOXY is a unit within the operating system which handles terminal-specific cursor movement.

SYSTEM.MISCINFO is a data file which contains various pieces of information pertaining to the hardware configuration (especially terminal-specific information).

In addition to the standard SYSTEM.MISCINFO and GOTOXY within the operating system, you are provided with two files called SOROC.GOTOXY and SOROC.MISCINFO. If you wish to use the SOROC terminal, instead of the standard APPLE internal console, you will <u>not</u> have to change the console driver within the SBIOS, as it works with both internal and external consoles. But, you will have to use the supplied SOROC GOTOXY and MISCINFO. To do this, you should bind SOROC.GOTOXY into the operating system (see Section IV.3.3, "Binding GOTOXY Using the LIBRARY Utility"). Also, you should use the filer to S(ave the old SYSTEM.MISCINFO as OLD.MISCINFO, and T(ransfer SOROC.MISCINFO to the boot disk as SYSTEM.MISCINFO.

If you wish to use some other terminal, you will have to write your own GOTOXY (see Chapter IV) and create your own SYSTEM.MISCINFO using the SETUP utility (see Chapter IV).

III.5 The APPLEINFO Table

The APPLEINFO Table contains some information pertaining to the configuration of peripherals specifically for the APPLE II computer. This table resides on the boot disk near the Primary Bootstrap code and is loaded into memory by the bootstrap at boot time. The table may be altered on disk to take effect at the next boot time or in memory to have immediate effect by using the CONFIG utility described in Chapter V. You should read about the CONFIG utility and decide which parameters you may want to change.

The areas in the table for disk parameters are used by the low-level routines in the operating system (RSP and BIOS) to describe how each of your disk drives are formatted and how blocks are mapped on to the physical disk sectors.

The areas pertaining to the console, printer, remote, user and clock are designed to be read only by the drivers in the SBIOS. They are there to allow the SBIOS drivers to be more flexible and have parameters that can be configured for a specific hardware setup. The unused areas are user-defined parameters that may be accessed by user-written SBIOS drivers.

III.6 Detailed Description of the Supplied Drivers

This section contains descriptions of the supplied drivers. A table that describes which cards belong to which slots within the APPLE II, is given in Appendix B.

SBIOS.CODE

Definitions:
SBIOS, SYSINIT, SYSHALT, POLLING, DSKINIT, DSKSTRT, DSKSTOP, DSKREAD, DSKWRIT, SETDISK, SETTRAK, SETSECT, SETBUFR
References:
CONINIT, CONSTAT, CONREAD, CONWRIT, DISKVECT, PRNINIT, PRNSTAT, PRNREAD, PRNWRIT, REMINIT, REMSTAT, REMREAD, REMWRIT, USRINIT, USRSTAT, USRREAD, USRWRIT, CLKREAD, SENABLE, SQUIET, SEVENT, ENDMARK
Zero Page: 20H-3FH used
Slot(s) used: n/a
Function:
Provides system init/halt routines, provides SBIOS jump table which is referenced by the BIOS interface routines, provides interface to appropriate disk driver by referencing the driver address table (DISKVECT). When a disk is selected (SETDISK) the jump table from the appropriate driver is copied onto the SBIOS jump table and that driver is used until the next call to SETDISK. This code is always a multiple of 256 bytes in length. It must be the first code in the SBIOS.

PRINTER.CODE

Definitions: PRINTER, PRNINIT, PRNSTAT, PRNREAD, PRNWRIT
References: NOTRDY, PATCH_ADDRS
Zero Page: 20H-3FH used
Slot(s) used: Slot 1 is checked for its card type.
Functions:
The first time PRNINIT is called, it calls the external routine PATCH_ADDRS which determines what kind of card is in slot 1 and patches the jump table in PRINTER.CODE accordingly. It outputs the character as appropriate using the routines that are patched into its jump table. PRNREAD is not implemented.

REMOTE.CODE

Definitions: REMOTE, REMINIT, REMSTAT, REMREAD, REMWRIT
References: NOTRDY, PATCH_ADDRS
Zero page: 78H-7FH used
Slot(s) used: Slot 2 is checked for its card type.
Functions:
The first time REMINIT is called, it calls PATCH_ADDRS which determines the card type in slot 2 and patches the REMOTE.CODE jump table accordingly. This driver then outputs the character as appropriate, using the routines that are patched into its jump table.

CONSOLE.CODE

Definitions: CONSOLE, CONINIT, CONREAD, CONWRIT
References: NOTRDY, PATCH_ADDRS
Zero page: 60H-6FH used
Slot(s) used:

Slot 3 is empty for internal console use, slot 3 contains a serial card for external console use, the following cards work for external use: CCS serial card, SSM serial card, Apple serial card, Apple Comm card, Sup-r-term 80 column video board, Smarterm 80 column video board, Videx 80 column video board, and most other 80 column video boards. (The Doublevision 80 column video board is not supported by CONSOLE.CODE; DBL.VISION.CODE supports this card.)

NOTE: It is expected that most other 80 column boards will work correctly with CONSOLE.CODE. In order for them to be supported, they must adhere to the serial card or firmware card protocol. A serial card is recognized if the hexadecimal location C305 contains the value 38 hexadecimal, the location C307 contains the value 18 hexadecimal, and the standard Apple entry points for console initialize (C800), console read (C84D), and console write (C9AA) are all present and interface to correct routines. A firmware card is recognized if the locations C305 and C307 contain the values just mentioned, and the location C30B contains the value 01 hexadecimal as well.

NOTE: Sup-r-term, Smarterm, and Videx are supported by their own board-specific drivers (as well as the general console driver). If you use the general console driver you won't need to relink the SBIOS in order to plug in and use an 80 column board (except for the Doublevision board). There are advantages to using the board-specific drivers, however. One advantage is that the shift key, alpha-lock key, and special characters may be produced in the exact manner described in Section I.3, "Important Keyboard Considerations." If you use the general console driver for an 80 column board, this may not be true; you should read Section I.3 in either case. Also, the board-specific drivers take up less memory room in bank two of the language card (on 64K systems). This may make a difference if you intend to write a large SBIOS (the SBIOS must fit into that space). Furthermore, the console-related parameters in the APPLEINFO table (described elsewhere in this manual) will be ignored if you use the general console routine with an 80 column video board. This means that you can't alter those parameters under this situation.

Function:

The internal console provides an 80 column terminal simulator with random cursor addressing, vertical and horizontal scrolling, uppercase only—lowercase/uppercase option, inverse uppercase option, selectable cursor option, keyboard entry routines which perform uppercase/lowercase conversion with soft shift key or shift wire modification, definable keys for special functions. The external console is automatically enabled if one of the supported serial cards are found in slot 3. If found, all console input/output goes to the card. Keyboard queuing works with both the internal and the external console.

COMMCARD.CODE

Definitions: COMINIT, COMSTAT, COMREAD, COMWRIT
References: KEYPRESSED, SIGNAL_EVENT, JUSTBOOT
Zero Page: 60H-7FH used
Slots(s) used: Slots 1-3 depending on the value of the X register.
Functions:

Calls the appropriate routine given there is a communication card in slot X where X is the X register.

DISKII.CODE

Definitions: DISKII
References: POLLING
Zero Page: 40H-5FH used
Slot(s) used: slot 6 contains physical drive 0,1
slot 5 contains physical drive 2,3
slot 4 contains physical drive 4,5

Function:

Reads/writes Apple format 16 sector tracks with 256 byte sectors, will position head properly for a track number up to 127. This driver works with Apple Disk II, Micro-Sci A-40/70 and other equivalent drives. It detects Apple Disk II or Micro-Sci and uses proper seek rates for each. It uses the APPLEINFO table to get the number of tracks on a particular drive which is used for recalibration of the head. This driver must be loaded on a page boundary.

DISKVECT.CODE

Definitions: DISKVECT
References: DISKII
Zero Page: none
Slots used: n/a
Function:

Provides an array of 6 pointers, each pointing to the jump table of one of the disk drivers. The 6 pointers correspond to physical drive numbers 0 through 5.

EMPTY.PRN.CODE

Definitions: EMPTYPRN, PRNINIT, PRNSTAT, PRNREAD, PRNWRIT
References: none
Zero Page: none
Slots used: none
Function:

Provides minimum code to return status code showing that the printer is off-line and that no character is waiting to be read from the printer.

EMPTY.REM.CODE

Definitions: EMPTYREM, REMINIT, REMSTAT, REMREAD, REMWRIT
References: none
Zero Page: none
Slots used: none
Function:

Provides minimum code to return status code showing that the remote device is off-line and that no character is waiting to be read from the remote port.

EMPTY.CON.CODE

Definitions: EMPTYCON, CONINIT, CONSTAT, CONREAD, CONWRIT
References: none
Zero Page: none
Slots used: none
Function:

Provides minimum code to return status code showing that the console is off-line and that no character is waiting to be read from the keyboard. Used only for creating a SYSTEM.BOOT file.

EMPTY.INT.CODE

Definitions: EMPTYINT, INTINIT, INTSTAT, INTREAD, INTWRIT
References: none
Zero Page: none
Slots used: none
Function:

Provides minimum code to return status code showing that the internal console is off-line and that no character is waiting to be read from the keyboard. Used if there is an 80 column card in slot 3 and CONSOLE.CODE is used.

EMPTY.USR.CODE

Routines: EMPTYUSR, USRINIT, USRSTAT, USRREAD, USRWRIT
References: none
Zero Page: none
Slots used: none
Function:

Provides minimum code to return status code showing that the user devices are off-line.

EMPTY.CLK.CODE

Definitions: EMPTYCLK, CLKREAD
References: none
Zero Page: none
Slots used: none
Function:

Provides minimum code to return status code showing that the clock is off-line. Also returns 0 in both time and words.

FIRMCARD.CODE

Definitions: FIRMINIT, FIRMSTAT, FIRMREAD, FIRMWRIT
References: KEYPRESSED, SIGNAL_EVENT, JUSTBOOT, INITCARD
Zero Page: 60H-7FH used
Slot(s) used: Slots 1-3 depending on the value of the X register.
Functions:

Calls the appropriate firmware routine given there is a firmware card in slot X where X is the X register.

INTCARD.CODE

Definitions: INTINIT, INTSTAT, INTREAD, INTWRIT
References: KEYINIT, KEYSTAT, KEYREAD
Zero Page: 60H-6FH used
Slot(s) used: This driver can only be used in slot 3.
Function:

The internal console provides an 80 column terminal simulator with random cursor addressing, vertical and horiztonal scrolling, uppercase only—lowercase/uppercase option, inverse uppercase option, selectable cursor option, keyboard entry routines which perform uppercase/lowercase conversion with soft shift key or shift wire modification, definable keys for special functions. The external console is automatically enabled if one of the supported serial cards are found in slot 3. If found, all console input/output goes to the card. Keyboard queuing works with both the internal and the external console.

INTERRUPTS.CODE

Definitions: SQUIET, SENABLE, SEVENT
References: none
Slot(s) used: none
Zero Page: none
Functions:

Enables or disables interrupts, returns event number given an interrupt has occurred.

PATCHCARD.CODE

Definitions: PATCH_ADDRS, INITCARD, NOTRDY, JUSTBOOT
References:

COMINIT, COMSTAT, COMREAD, COMWRIT,
SERINIT, SERSTAT, SERREAD, SERWRIT,
FIRMINIT, FIRMSTAT, FIRMREAD, FIRMWRIT,
PRLINIT, PRLSTAT, PRLREAD, PRLWWRIT,
INTINIT, INTSTAT, INTREAD, INTWRIT

Slot(s) used: Can enable slots 1-3
Zero Page: none
Functions:

Patches CONSOLE.CODE, PRINTER.CODE, or REMOTE.CODE with the appropriate routines. It also initializes a card to its startup state.

PRLCARD.CODE

Definitions: PRLINIT, PRLSTAT, PRLREAD, PRLWRIT
References: none
Zero Page: 70H-77H used
Slot(s) used: This driver can only be used for slot 1.
Functions:
Calls the appropriate parallel routine given there is a parallel in slot 1.

SERCARD.CODE

Definitions: SERINIT, SERSTAT, SERREAD, SERWRIT
References: KEYPRESSED, SIGNAL_EVENT, JUSTBOOT, INITCARD
Zero Page: 60H-7FH used
Slot(s) used: Slots 1-3 depending on the value of the X register.
Functions:
Calls the appropriate serial routine given there is a serial card in slot X where X is the X register.

SUPRTERM.CODE

Definitions: SUPRTERM, CONINIT, CONSTAT, CONREAD, CONWRIT
References: KEYINIT, KEYSTAT, KEYREAD
Zero Page: 20H-35H, 60H-6FH
Slot(s) used: Sup-r-term card must be in slot 3
Function:
Provides standard terminal driver with output cursor, up/down screen scrolling, 80 column uppercase/lowercase display with all standard terminal functions including random cursor addressing.

DBL.VISION.CODE

Definitions: DBLVISION, CONINIT, CONSTAT, CONREAD, CONWRIT
References: KEYINIT, KEYSTAT, KEYREAD
Zero Page: 20H-35H, 60H-6FH
Slot(s) used: Double Vision card must be in slot 3
Function:
Provides standard terminal driver with output cursor, up/down screen scrolling, 80 column uppercase/lowercase display with all standard terminal functions including random cursor addressing.

SMARTERM.CODE

Definitions: SMARTERM, CONINIT, CONSTAT, CONREAD, CONWRIT
References: KEYINIT, KEYSTAT, KEYREAD
Zero Page: 20H-35H, 60H-6FH
Slot(s) used: SMARTERM card must be in slot 3
Function:

Provides standard terminal driver with output cursor, up/down screen scrolling, 80 column uppercase/lowercase display with all standard terminal functions including random cursor addressing.

VIDEX.CODE

Definitions: VIDEX, CONINIT, CONSTAT, CONREAD, CONWRIT
References: KEYINIT, KEYSTAT, KEYREAD
Zero Page: 20H-35H, 60H-6FH
Slot(s) used: VIDEX card must be in slot 3
Function:

Provides standard terminal driver with output cursor, up/down screen scrolling, 80 column uppercase/lowercase display with all standard terminal functions including random cursor addressing.

KEYBOARD.CODE

Definitions: KEYINIT, KEYSTAT, KEYREAD
References: none
Zero Page: 20H-35H
Slot(s) used: none
Function:

Provides standard functions to be used with the APPLE II keyboard. Provides all the functions for uppercase/lowercase entry including soft shift key, shift wire mod support, console queuing and optional support of a full uppercase/lowercase keyboard.

SVA8INCH.CODE

Definitions: SVA8INCH
References: none
Zero Page: 20H-35H
Slot(s) used: SVA single density disk controller in slot 7
Function:

Provides interface to two eight inch floppy disk drives through a Sorrento Valley Associates floppy controller. Any two physical drives may be assigned to use this driver; one must be an odd physical drive number and one an even drive number. (Only two drives may use this driver.) The disk controller must be in slot 7 or the system will crash. This calls ROM routines for SBIOS functions and if the card isn't installed, it will fail. You must change the table of vectors and reassemble DISKVECT.TEXT (see Section IV.1.10). You must also set the parameters for the 8" drives using CONFIG (see Section V.3).

SVAVECT.CODE

Definitions: DISKVECT
References: DISKII, SVA8INCH
Function:

Assigns two eight inch drives to physical devices #4 and #5. Devices 0, 1, 2, 3 are APPLE mini-floppies.

CHAPTER IV

WRITING YOUR OWN PERIPHERAL DRIVERS

If you desire to write your own peripheral drivers you will need a detailed understanding of the SBIOS. This section describes the SBIOS routines and their requirements. Then it describes how SBIOS routines are called, and concludes with a section on how to test an SBIOS. (Chapter III describes how to link together an SBIOS out of individual drivers.)

The SETUP utility is described in this chapter. SETUP is used to create or alter SYSTEM.MISCINFO. SYSTEM.MISCINFO is a data file on the boot disk which contains various pieces of information, most of which pertain to terminal handling. You will probably have to alter SYSTEM.MISCINFO if you are using a terminal other than the standard internal console, or a SOROC external console. (Drivers and MISCINFO files for those consoles are provided, see Chapter III.)

The GOTOXY unit of the operating system is described in this chapter. GOTOXY moves the cusor to a given position on the screen. GOTOXY is a very small piece of code which you will have to write if you aren't using an internal console or a SOROC external console. (GOTOXY code is provided for these, see Chapter III.) If you need to write this code, you will also need to bind it into the operating system using the LIBRARY utiltiy. This process is also described in this chapter.

Also, bootstrapping is described in case you want to bootstrap from a disk other than an APPLE II mini-floppy or Micro-Sci A-40/70 mini-floppy (for which a Primary Bootstrap is provided).

IV.1 The SBIOS Routines

IV.1.1 The Jump Vector

These are the names of the SBIOS routines, along with a brief description of each routine. SBIOS routines are called through a jump vector table. The Vector Number column shows the order that these routines appear in the table and the Vector Offset column shows the actual offset of the jump instruction for a given routine within the table.

Routine Name	Vector Number	Vector Offset	Description
SYSINIT	0	0	initialize machine
SYSHALT	1	3	exit UCSD p-System
CONINIT	2	6	console initialize
CONSTAT	3	9	console status
CONREAD	4	C	console input
CONWRIT	5	F	console output
SETDISK	6	12	set disk number
SETTRAK	7	15	set track number
SETSECT	8	18	set sector number
SETBUFR	9	1B	set buffer address
DSKREAD	10	1E	read sector from disk
DSKWRIT	11	21	write sector to disk
DSKINIT	12	24	reset disk
DSKSTRT	13	27	activate disk
DSKSTOP	14	2A	deactivate disk
PRNINIT	15	2D	printer initialize
PRNSTAT	16	30	printer status
PRNREAD	17	33	printer read
PRNWRIT	18	36	printer write
REMINIT	19	39	remote initialize
REMSTAT	20	3C	remote status
REMREAD	21	3F	remote read
REMWRIT	22	42	remote write
USRINIT	23	45	user devices initialize
USRSTAT	24	48	user devices status
USRREAD	25	4B	user devices read
USRWRIT	26	4E	user devices write
CLKREAD	27	51	system clock read
SQUIET	28	54	disables interrupts
SENABLE	29	57	enables interrupts
SEVENT	30	5A	determines what kind of interrupt just occurred

IV.1.2 Descriptions of the Routines

SBIOS routines are called by the BIOS. They are called rarely, if ever, by your programs. Parameters are sometimes passed on the stack or in registers. The parameter passing conventions are discussed later in this chapter.

Many of the SBIOS routines return a status word: this status word is used as the system's IORESULT. It is important that status words be returned correctly. If they are incorrect, the system may crash or even fail to bootstrap. An IORESULT of 0 signifies a correct operation. An IORESULT of 9 should always be returned when an I/O device isn't on-line. (Remember that floppy disks are frequently removed and replaced, so the disk-handling routines should be careful to check that a disk is in the desired drive.)

The SBIOS must maintain four variables that describe the state of disk I/O. These are called CURDISK, CURTRAK, CURSECT, and CURBUFR. The first three describe the current disk drive (zero based) and the current track and sector on that disk. CURBUFR is a pointer to a read/write buffer in main memory.

Following is a description of each of the SBIOS routines, in order:

SYSINIT

SYSINIT is the first routine called when a system is bootstrapped. It should initialize the hardware in any way necessary. This may include setting up interrupt vectors, enabling RAM memories, and turning off any I/O devices that won't be used.

A pointer to the interpreter's jump table is passed to SYSINIT. This pointer isn't used by the bootstrap; it is provided for use by other SBIOS routines. It is saved on Zero Page in the word labeled BTABLE. See SBIOS.GLOB.TEXT source code.

SYSINIT is provided in SBIOS.CODE and may not be replaced.

SYSHALT

SYSHALT is called when the p-System terminates (through a H(alt). This enables the APPLE mother board ROMs and performs a power on reset.

SYSHALT is provided in SBIOS.CODE and may not be replaced.

CONINIT

CONINIT initializes the console port. It returns the status of the console connection.

Initializing the console means preparing the console hardware to send and receive characters. If the terminal's baud rate and parity bits can be set by software, CONINIT should configure it to operate as quickly as possible, ignoring parity bits.

If CONINIT encounters no problems in initializing the console, it should return a 0 (zero). If it detects that the terminal is off-line, it should return a 9.

CONSTAT

CONSTAT returns two parameters that describe the status of the console.

The first parameter is the state of the console connection. This is identical to the parameter returned by CONINIT: if the console is on-line, the parameter should return 0; if the console is off-line (disconnected), the parameter should return 9.

The second parameter describes the state of the console input channel. If a character has been entered on the keyboard, the parameter should return FF hexadecimal; otherwise, it should return 0. (Note: CONSTAT does <u>not</u> read the pending character, but merely reports its presence.)

CONREAD

CONREAD reads a single character from the keyboard. It returns that character, and the status of the console connection.

If the console is on-line and a character is pending, CONREAD reads that character. If the console is on-line but no character is pending, CONREAD <u>waits</u>, by polling the console, until a character appears, and then reads that character.

If the read was successful, the status parameter should return a 0. If the console was off-line, the parameter should return a 9. If a character was read but there appears to be a transmission problem, CONREAD should return the character, and the status parameter should be set to 1.

The character read should be returned <u>exactly</u> as read from the keyboard port with no modifications.

CONWRIT

CONWRIT writes a single character to the console. It reports the status of the console connection.

If the console is on-line, the character is sent, and CONWRIT returns 0. If the console is off-line, CONWRIT returns 9. If there is a transmission problem, CONWRIT returns 1: the system will assume that the character was lost.

CONWRIT shouldn't alter the output character in any way, unless it must do so in order for the console to display the character properly. (For example, don't strip parity bits, unless the terminal won't function properly when they are set).

SETDISK

SETDISK sets CURDISK.

CURDISK (as well as CURTRAK, CURSECT, and CURBUFR, which are mentioned below), is a global value in the BIOS. The SBIOS must keep a copy of these values, for use by the SBIOS disk-handling routines (DSKREAD, DSKWRIT, DSKINIT, DSKSTRT, and DSKSTOP).

Disk numbers may be in the range 0 through 5.

SETDISK merely changes a value; it doesn't alter the hardware state, nor does it return a status.

SETTRAK

SETTRAK sets CURTRAK.

CURTRAK is used by DSKREAD and DSKWRIT.

Track numbers range from 0 to one less than the number of tracks on the disk.

Like SETDISK, SETTRAK merely changes a value; it doesn't alter the hardware state, nor does it return a status.

SETSECT

SETSECT sets CURSECT.

CURSECT is used by DSKREAD and DSKWRIT.

Sector numbers range from 1 to the number of sectors on a track.

SETSECT doesn't alter the hardware state or return a status.

SETBUFR

SETBUFR sets CURBUFR.

CURBUFR is used by DSKREAD and DSKWRIT. It is the hardware address of a buffer area large enough to contain one sector.

SETBUFR doesn't alter the hardware state or return a status.

DSKREAD

DSKREAD reads a sector from a floppy disk and returns a status.

DSKREAD must ensure that the sector it reads is identified by the values CURDISK, CURTRAK, and CURSECT. It should read the sector into the buffer whose address is CURBUFR.

DSKREAD may assume that CURDISK, CURTRAK, CURSECT, and CURBUFR have all been correctly set by previous calls to SETDISK, SETTRAK, SETSECT, and SETBUFR. It shouldn't change these values.

If the read was successful, the status should return 0. If the disk was off-line or otherwise unavailable, the status should return 9. If there was an error in reading, the status should return 1. If there are any problems, DSKREAD should always return an error status; it shouldn't retry the read or hang on an error.

DSKREAD may also assume that DSKINIT has already been called at least once for the CURDISK, and that DSKSTRT has been called for the CURDISK more recently than DSKSTOP.

DSKWRIT

DSKWRIT writes a sector to a floppy disk and returns a status.

DSKWRIT must ensure that the sector it writes is identified by the values CURDISK, CURTRAK, and CURSECT. It should write the sector from the buffer whose address is CURBUFR.

DSKWRIT may assume that CURDISK, CURTRAK, CURSECT, and CURBUFR have all been correctly set by previous calls to SETDISK, SETTRAK, SETSECT, and SETBUFR. It shouldn't change these values.

If the write was successful, the status should return 0. If the disk was off-line or otherwise unavailable, the status should return 9. If there was an error in writing, the status should return (decimal) 16. If there are any problems, DSKWRIT should always return an error status; it shouldn't retry the write or hang on an error.

DSKWRIT may also assume that DSKINIT has already been called at least once for the CURDISK, and that DSKSTRT has been called for the CURDISK more recently than DSKSTOP.

To keep disk writes reasonably fast, DSKWRIT shouldn't do read-after-write checking.

DSKINIT

DSKINIT resets the disk CURDISK, and returns a status.

DSKINIT may assume that SETDISK and DSKSTRT have already been called to select CURDISK and set it in motion.

DSKINIT must move the recording head to track 0. If possible, the drive should be reset to its power-up state, and prepared for reading and writing.

If CURDISK is on-line (that is, the drive is connected, turned on, and contains a floppy disk) and the DSKINIT is successful, the status should return 0; otherwise, the status returns 9.

If there are any problems, DSKINIT should always return an error status rather than hang on an error.

DSKINIT shouldn't alter the values of CURDISK, CURTRAK, CURSECT, and CURBUFR.

DSKINIT is called when the system is booted or reinitialized (that is, after SYSINIT is called). It is also called when a disk read error occurs before a retry is attempted by the BIOS. It isn't called every time a disk read/write sequence is begun; that is the purpose of DSKSTRT.

DSKSTRT

DSKSTRT prepares the disk CURDISK for a series of read, write, or initialize operations (that is, for a sequence of calls to DSKREAD, DSKWRIT, and DSKINIT).

DSKSTRT may assume that SETDISK has already been called to set the value of CURDISK.

DSKSTRT should perform any motor starting and head loading operations that aren't done automatically (by the hardware) as consequences of read, write, and initialize operations.

DSKSTRT doesn't return a status.

This routine is intended for use with certain mini-floppy drives (5-1/4"). Most 8" floppies won't require that DSKSTRT perform any action.

DSKSTOP

DSKSTOP stops the disk CURDISK; it is meant to be called at the end of a series of disk read, write, or initialize operations.

DSKSTOP may assume that SETDISK has already been called to set the value of CURDISK.

DSKSTOP should perform any motor stopping and head unloading operations that aren't done automatically (by the hardware) after read, write, and initialize operations.

DSKSTOP doesn't return a status.

This routine is intended for use with mini-floppy drives (5-1/4"). Most 8" floppy hardware won't require that DSKSTOP perform any action.

PRNINIT

PRNINIT initializes the printer port. It reports the status of the printer connection.

Initializing the printer means preparing the printer hardware to receive (and possibly to send) characters. If baud rate and parity bits can be set by software, PRNINIT should configure the printer to operate as quickly as possible, with no parity translation. Any interrupt vectors associated with printer operation should be set in SYSINIT, not PRNINIT.

If PRNINIT encounters no problems, it should return a 0. If the printer is off-line, it should return a 9.

PRNINIT shouldn't send the printer a form feed.

PRNSTAT

PRNSTAT returns two parameters that describe the status of the printer.

The first parameter is the state of the printer connection. This is identical to the status returned by PRNINIT: if the printer is on-line, the status must be 0; if the printer is off-line, the status must be 9.

The second status is the state of the printer input or output channel depending on the I/O direction flag (X register). If the I/O flag is nonzero, it indicates input and PRNSTAT returns FF hexadecimal if a character is pending on the printer input channel; otherwise, it returns 0. (Note: PRNSTAT doesn't read the pending character, but merely reports its presence.) If the I/O direction flag is 0, it indicates output and PRNSTAT returns 0 if the printer can accept a character and FF hexadecimal if the printer is busy.

PRNREAD

PRNREAD reads a single character from the printer input channel. It returns the character, and the status of the printer connection.

If the printer is on-line and a character is pending on the input channel, PRNREAD reads that character. If the printer is on-line but no character is pending, PRNREAD waits, by polling the printer input channel, until a character appears, and then reads it.

If the read was successful, the status is 0. If the printer is off-line, the status is 9. If a character was read but there were problems in transmission, PRNREAD should return the character and set the status to 1.

The character should be returned exactly as read from the input channel, with no modifications.

If the system's printer has no input channel, PRNREAD should do nothing and return a status of 0.

PRNWRIT

PRNWRIT writes a single character to the printer output channel. It returns the status of the printer connection.

If the printer is on-line, the character is transmitted as soon as the printer is ready to receive it. The status returned is 0.

If there are transmission problems, the status returned is 1.

If the printer is off-line, the status returned is 9.

PRNWRIT shouldn't alter the output character except when this is necessary to display the character on the printer correctly (for example, don't strip parity bits, unless the printer won't function properly when they are set).

REMINIT

REMINIT initializes the remote port (which is intended for an extra serial line such as a phone link). It returns the status of the remote connection.

Initializing the remote port means preparing the remote hardware to send and receive characters. If baud rate and parity bits can be set by software, REMINIT should configure the port to operate as quickly as possible, with no parity translation. Any interrupt vectors associated with remote I/O should be set in SYSINIT, not in REMINIT.

If all is well, REMINIT returns a status of 0. If the remote port is off-line, or if there is no driver for the remote hardware, REMINIT returns 9.

REMSTAT

REMSTAT returns two parameters that describe the status of the remote port.

The first parameter is identical to the status returned by REMINIT: if all is well, the status is 0; if the port is off-line or there is no driver, the status is 9.

The second parameter returns FF hexadecimal if a character has been received on the remote channel, and 0 if no character has been received. (Note that REMSTAT doesn't read the pending character; it merely reports its presence.)

REMREAD

REMREAD reads a single character from the remote input channel. It returns the character, and the status of the remote connection.

If the remote port is on-line and a character is pending, REMREAD reads that character. If the port is on-line but no character is pending, REMREAD waits, by polling the remote port, until a character appears, and then reads it.

If the read was successful, the status is 0. If the remote port is off-line or has no driver, the status is 9. If the character was read but there was a transmission problem, REMREAD should return the character, and the status is 1.

The character read should be passed exactly as it is read from the remote input port, with no modifications.

REMWRIT

REMWRIT writes a single character to the remote output channel. It returns the status of the remote connection.

If the remote port is on-line, the character is sent and the status is 0. If the remote port is off-line or has no driver, the status is 9. If there is a transmission problem, the character is sent and the status is 1.

REMWRIT shouldn't alter the output character in any way, unless it must do so to ensure proper transmission. (For example, don't strip parity bits, unless the remote line or device won't function when they are present.)

CLKREAD

CLKREAD returns a time based on the current state of the system's hardware clock, and a status.

The time is returned as a 32-bit integer. Time is measured in 1/60ths of a second. If the system clock runs continually, time should be measured from midnight. Otherwise, time should be measured from the most recent call to SYSINIT.

Thus, SYSINIT must restart the system clock, unless the clock runs continually.

If the clock is on-line and enabled, CLKREAD returns the time, and a status of 0. If the clock is off-line, CLKREAD returns a status of 9, and sets the time equal to 0.

If the hardware clock doesn't count in 1/60ths of a second, CLKREAD should perform some reasonable approximation.

SQUIET

SQUIET disables interrupts.

SENABLE

SENABLE enables interrupts.

SEVENT

SEVENT checks the processor flags and sets register A to 33 (decimal) if the break flag is set. If it wasn't set and KEYPRESSED is nonzero event, 19 (decimal) is returned; otherwise, event 32 (decimal), IRQ interrupt, is returned.

IV.1.3 User-Defined Devices

The routines that handle user-defined devices (that is specialized hardware of one kind or another) have several features in common.

The system may support a number of user-defined devices. Yet the SBIOS has only one set of USRxxxx routines: USRINIT, USRSTAT, USRREAD, and USRWRIT.

When one of these routines is called, the user must specify which particular device is intended by passing the routine the device number. Numbers of user-defined devices may be in the range 128 through 255. Pascal programs may access user-defined devices by using the appropriate device numbers when calling the UNITREAD, UNITWRITE—family of intrinsics (see the UCSD Pascal Handbook).

NOTE: These numbers are truly user-definable; it is the SBIOS routines that are responsible for knowing which device is which, and what its number is. No other system routines have knowledge of user-defined devices.

The standard status parameters returned by most SBIOS routines include 0 for on-line (and all correct), and 9 for off-line. It may be that one or more user-defined devices in your system must return more detailed information about their state. If this is the case, the numbers 128 through 255 are available as user-definable status codes. The responsibility for handling these nonstandard status codes belongs entirely to the user's software.

USRINIT

USRINIT initializes a single user-defined device. It returns a status.

USRINIT is passed a device number.

If the specified device is on-line, USRINIT resets it to its power-up condition. Any interrupt vectors associated with the device should be initialized in SYSINIT, not USRINIT.

If the device is on-line, USRINIT returns a status of 0. If the device is off-line (or just plain nonexistent), USRINIT returns a status of 0. Other status codes may be defined by you.

USRSTAT

USRSTAT returns status information about a user-defined device.

USRSTAT is passed a device number, a pointer to a status record, and an input/output toggle.

A simple status is returned, as with most SBIOS routines. This is 0 for on-line, 9 for off-line. Other status codes may be defined by you.

The pointer points to a 30-word status record in memory. USRSTAT may write status information in this area. The format and meaning of the status record are entirely up to you.

The "input/output toggle" is a single word. If its low-order bit is 0, USRSTAT should report on the device's output channel. If its low-order bit is 1, USRSTAT should report on the device's input channel.

The three high-order bits of the input/output toggle may also be used to further specify the sort of status information required. This is entirely at your option.

USRSTAT is the SBIOS routine that corresponds to the Pascal intrinsic UNITSTATUS. You may wish to refer to the description of this intrinsic in the Users' Manual.

USRREAD

USRREAD reads information from a user-defined device into a buffer in main memory. It returns a status.

USRREAD is passed a device number, a pointer to a buffer, and three extra parameters.

Information is read from the specified device into the buffer in memory.

The three extra parameters may be defined according to the requirements of the specified device. This is entirely up to you.

USRREAD returns 0 for on-line, 9 for off-line, or a user-defined status number.

USRWRIT

USRWRIT writes information from a buffer in main memory to a user-defined device. It returns a status.

USRWRIT is passed a device number, a pointer to a buffer, and three extra parameters.

Information is written to the specified device from the memory buffer.

The three extra parameters may be defined according to the requirements of the specified device. This is entirely up to the user.

USRWRIT returns 0 for on-line, 9 for off-line, or a user-defined status number.

IV.1.4 Physical Organization of the SBIOS

The SBIOS should be organized with the jump vector (see below) at the beginning, followed by data space and code. A sample SBIOS might look like:

```
        .PROC SBIOS
        .INCLUDE SBIOS.GLOB.TEXT
        .DEF SYSINIT,SYSHALT,POLLING
        .REF CONINIT, CONSTAT, ...
SBIOS                           ; Beginning of the SBIOS
        JMP     SYSINIT         ; Jump to SYSINIT routine
        JMP     SYSHALT         ; Jump to SYSHALT routine
        JMP     CONINIT         ; Jump to CONINIT routine
        JMP     CONSTAT         ; Jump to CONSTAT routine
              .
              .
              .

POLLING  JMP   @BTABLE

SYSINIT
              .
              .
              .
        RTS                      ; Make sure to return to caller

SYSHALT
              .
              .
              .
        .ALIGN 256      ; Length must by a multiple of 256 to
                        ; maintain page boundaries
        .END
```

IV.1.5 How SBIOS Routines are Called by the p-System

This section is provided for your information. You shouldn't have to call any SBIOS routines.

Each SBIOS routine is called through a jump vector. The jump vector is an array of jump instructions. A program calling an SBIOS routine must access the jump vector rather than the routine's physical location; in this way, the system need not know the size of SBIOS routines, or how they are ordered in memory.

If the contents of the jump vector are correct, a call to the SBIOS routine will jump into the jump vector and then to the desired routine. The call to the SBIOS should is a subroutine call (JSR). Each individual SBIOS routine is responsible for returning to its caller properly.

IV.1.6 Vector Lists and Register Assignments

The system assumes that the SBIOS routines use all registers except the stack pointer.

SBIOS routines must return their status (IORESULT) in the X register.

Parameters are passed to SBIOS routines in the X and A registers. Where these registers appear together (XA), they represent a 16-bit quantity: X is the high-order byte and A is the low-order byte.

The disk read routines read into a buffer in main memory. The stack pointer shouldn't be modified (except as necessary to return from each routine in a standard manner; for example, to remove parameters, or place results on the stack).

The following table shows the parameters for each routine in the basic SBIOS, along with each routine's vector offset (that is, the position in the jump table of the instruction that jumps to that routine). (The vector offsets are shown in hexadecimal.)

Routine	Vector Offset	Parameters
SYSINIT	00	passed: XA = pointer to interpreter's jump table
SYSHALT	03	<none>
CONINIT	06	returns: X = IORESULT
CONSTAT	09	returns: X = IORESULT A = 0 if no char pending = FF if char pending
CONREAD	0C	returns: X = IORESULT A = input char
CONWRIT	0F	passed: A = output char returns: X = IORESULT
SETDISK	12	passed: A = disk no. (CURDISK)
SETTRAK	15	passed: A = track no. (CURTRAK)
SETSECT	18	passed: XA = sector no. (CURSECT)
SETBUFR	1B	passed: XA = buffer addr. (CURBUFR)
DSKREAD	1E	returns: X = IORESULT
DSKWRIT	21	returns: X = IORESULT
DSKINIT	24	returns: X = IORESULT
SKSTRT	27	<none>
DSKSTOP	2A	<none>
PRNINIT	2D	returns: X = IORESULT
PRNSTAT	30	returns: X = IORESULT A = 0 if no char pending = FF if char pending
PRNREAD	33	returns: X = IORESULT A = input char
PRNWRIT	36	passed: A = output char returns: X = IORESULT
REMINIT	39	returns: X = IORESULT
REMSTAT	3C	returns: X = IORESULT A = 0 if no char pending = FF if char pending
REMREAD	3F	returns: X = IORESULT A = input char
REMWRIT	42	passed: A = output char returns: X = IORESULT
USRINIT	45	passed: A = device number returns: X = IORESULT

Routine	Vector Offset	Parameters
USRSTAT	48	passed: SP = return address input/output toggle pointer to status rec device number
		returns: X = IORESULT
USRREAD	4B	passed: SP = return address extra parameter 2 extra parameter 1 pointer to buffer device number extra parameter 3
		returns: X = IORESULT
USRWRIT	4E	passed: SP = return address extra parameter 2 extra parameter 1 pointer to buffer device number extra parameter 3
		returns: X = IORESULT
CLKREAD	51	returns: X = IORESULT SP = least significant word most significant word
SQUIET	54	
SENABLE	57	
SEVENT	5A	returns: A = event number

Some of the above routines are passed parameters on top of the stack. These routine must remove these parameters from the stack, and not alter the stack in any other way. All stack parameters are 16-bit words. In the table below, parameters on the stack are shown in the order they appear on the stack, with the stack pointer (SP) at the top (the least significant byte of a word is popped first). The "extra parameters" 1, 2, and 3 for the USRREAD and USRWRIT routines correspond to (respectively) the byte count, block number, and control word parameters in the Pascal intrinsics UNITREAD and UNITWRITE.

IV.1.7 The SBIOS Global Variables

This section contains a listing of the source for the SBIOS global variables. On the CODE: diskette there is a file called SBIOS.GLOB.TEXT which contains a soft copy of these declarations. If you are going to write your own driver routines, you will want to .INCLUDE this file as shown in the sample drivers in the next two sections.

```
; Zero Page Variables and Globals
          .ASECT
          .ORG 0
; Reserved for Pascal External Procedures:
          .BLOCK 32
; Temporary space available to SBIOS routines, externals, etc.,
;  the state of these aren't guaranteed after a call to POLLING:
TEMP1     .WORD
TEMP2     .WORD
TEMP3     .WORD
TEMP4     .WORD
TEMP5     .WORD
TEMP6     .WORD
TEMP7     .WORD
TEMP8     .WORD
TEMP9     .WORD
TEMP10    .WORD
TEMP11    .WORD

; System Space:
BTABLE    .WORD      ; Pointer to BIOS jump table
                     ; Jump Vectors: 0=pollunits, 1=dskchng,
2=interrupt

APPLEINFO .WORD      ; Pointer to APPLEINFO area
          .WORD      ; Reserved
          .WORD      ; Reserved
          .WORD      ; Reserved

; SBIOS Drivers Space:
DSKZP     .BLOCK 32.      ; Allocate 32 bytes
CONZP     .BLOCK 16.      ; Allocate 16 bytes
PRNZP     .BLOCK 8. ; Allocate 8 bytes
REMZP     .BLOCK 8  ; Allocate 8 bytes

          .PSECT
; Return codes used by SBIOS routines:
ONLINE    .EQU 0     ; Online IORESULT (returned in Xreg)
CRCERROR  .EQU 1     ; CRC error IORESULT
BADNUM    .EQU 2     ; Bad device number IORESULT
OFFLINE   .EQU 9     ; Device off-line IORESULT
READONLY  .EQU 16    ; Device read only IORESULT
```

IV.1.8 Sample Disk Driver

This section contains an outline of the DISKII driver provided with the p-System. It is intended as an example outline of how a driver is written. The next section contains an outline for a console driver and some brief explanations for other drivers you may want to write.

NOTE: Only one disk driver may use the zero page area dedicated to disk drivers (DSKZP in the SBIOS globals, above). The supplied DISKII driver uses this area already. If you replace DISKII with your own driver you may use this space. If you are simply adding another driver (or in any situation where you are planning to use more than one disk driver), the additional driver(s) must use the TEMP words as Zero Page memory. These TEMPs aren't guaranteed between calls to the driver routines which use them.

The following is the sample disk driver:

```
            .PROC DISKII

            .INCLUDE SBIOS.GLOB.TEXT

            JMP SETDISK
            JMP SETTRAK
            JMP SETSECT
            JMP SETBUFR
            JMP DSKREAD
            JMP DSKWRIT
            JMP DSKINIT
            JMP DSKSTRT
            JMP DSKSTOP

; Variables:
CURDISK     .BYTE 0
CURTRAK     .WORD 0
CURSECT     .BYTE 0
CURBUFR     .WORD 0

SETDISK     ; Params: AREG = disk number in 0..5
            ; Note: It is not necessary to call DSKCHNG, that is
            ; handled by the SBIOS main procedure.
            STA CURDISK
            RTS

SETTRAK     ; Params: AREG = track lo, XREG=TRACK HI
            STX CURTRAK + 1
            ;  in range 0..(number of tracks -1)
            STA CURTRAK
            RTS

SETSECT     ; Params: AREG = sector number
            ;  in 1 .. number of sectors
            STA CURSECT
            RTS
```

```
SETBUFR   ; Params: AREG = buffer pointer lo,
          ;  XREG = buffer pointer hi
          ; Read or write sector to/from buffer
          ;  at this location
          STA CURBUFR
          STX CURBUFR+1
          RTS

DISKINIT  ; Start of code to move head to track 0.
          ... Init Code ...
          LDX #ONLINE  ; Return appropriate status code
          RTS

DSKREAD   ; Start of code to read a sector.
          ;  You might want to use TEMP1 as a
          ;  zero page pointer to the buffer
          LDA CURBUFR
          STA TEMP1
          LDA CURBUFR+1
          STA TEMP1+1
          ... Read code ...
          LDX #ONLINE  ; Return appropriate status code
          RTS

DSKWRIT   ; Start of code to write a sector.
          ;  You might want to use TEMP1 as a
          ;  zero page pointer to the buffer
          LDA CURBUFR
          STA TEMP1
          LDA CURBUFR+1
          STA TEMP1+1
          ... Write code ...
          LDX #ONLINE  ; Return appropriate status code
          RTS

DSKSTRT   ; Start of code to turn on drive motor
          ... Start motor code ...
          RTS

DSKSTOP   ; Sart of code to turn off drive motor
          ... Stop motor code ...
          RTS

          .END
```

IV.1.9 Other Sample Drivers

All drivers must first meet the performance requirements specified in Section IV.1.2 which defines what the routines actually do and their general parameters. They must also meet the requirements specified in Section IV.1.6 regarding register assignments for parameters and results. The following information is needed to write and link SBIOS drivers into an APPLE configurable SBIOS.

CONSOLE DRIVER

Your console routine must meet the following requirements:

1. The source text must begin with a .PROC (not .FUNC) and the PROC name must be unique among the drivers to be linked.
2. It must .DEF CONSTAT, CONINIT, CONREAD, CONWRIT
3. You may only use the zero page allocated(16 bytes at 60H) to the console.
4. You may use the zero page temporaries (20H thru 35H).

See recipe for a console driver below.

PRINTER DRIVER

The printer driver is very similar to the console driver. It must meet all the same requirements with the exception that it .DEF's PRNSTAT, PRNINIT, PRNREAD, PRNWRIT and uses the zero page allocated (8 bytes at 70H) to the printer driver.

REMOTE DRIVER

The remote driver is very similar to the console driver. It must meet all the same requirements with the exception that it .DEF's REMSTAT, REMINIT, REMREAD, REMWRIT and uses the zero page allocated (8 bytes at 78H) to the remote driver.

USER DRIVER

The user driver must is also similar except parameter passing also uses the stack. It must .DEF USRSTAT, USRINIT, USRREAD, USRWRIT. It may ONLY use the zero page temporaries. There is no zero page space reserved for the user routines.

CLOCK DRIVER

The clock driver returns the time words on the stack. It must .DEF CLKREAD. It may ONLY use the zero page temporaries. There is no zero page space reserved for the clock routine.

The following is an example console driver:

```
          .PROC YOURCONSOLE

          .DEF CONINIT,CONSTAT,CONREAD,CONWRIT

          .INCLUDE SBIOS.GLOB.TEXT

          .ASECT
          .ORG CONZP        ;beginning of zero page for
                            ; console (16 bytes)
;sample variables you might declare:
CH        .BYTE             ;cursor horizontal position
CV        .BYTE             ;cursor vertical position
          .PSECT

;All routines return status code in Xreg
;CONREAD returns keycode read in Areg
;CONWRIT get screen character in Areg
;CONSTAT returns status flag in Areg

CONINIT ... your init code ...
          LDX #ONLINE
          RTS

CONSTAT ... your status code ...
          LDA #0
          LDX #ONLINE
          RTS

CONREAD ... your read code ...
          LDX #ONLINE
          RTS
```

```
CONWRIT ... your write code ...

        ; if you want to handle upper/lowercase conversion:
        TAX             ;save character in Xreg
        LDY #4E         ;offset to HAS_LC_VIDEO flag for console
        LDA @APPLEINFO,Y
        LSR A           ;flag bit to carry
        TXA             ;restore character into Areg
        BCS $10         ;branch if true (no need to convert)
        CMP #61         ;check to see if its lowercase
        BCC $10         ;branch if its not a lowercase character
        SBC #32.        ;carry is set, convert to lowercase
                        ; to uppercase
$10
        ... write char to screen ...
        LDX #ONLINE
        RTS

        .END
```

IV.1.10 Example Driver Programs

```
;*******************************************************
;
;       8" floppy disk for Apple II sbios
;       Sorrento Valley Associates 8" controller
;       with Autoboot firmware
;
;*******************************************************

        .PROC SVA8INCH

; notes:
;   SVA controller firmware uses 18H,19H on
;   zero page although these locations
;   are saved and restored by this driver.
;   When using this driver do not set the
;   loword of memory below 0800H,
;   it uses locs: 47F,4FF,77F (and possibly others)

;
; zero page:
;
        .ASECT
        .ORG 18H
USED    .WORD
        .ORG 20H
TEMP1   .WORD
TEMP2   .WORD
TEMP3   .WORD
        .PSECT

;
; constants:
;
ONLINE  .EQU 0          ;status- on line (0)
READERR .EQU 1          ;disk read error (1)
OFFLINE .EQU 9          ;disk off line   (9)
READONLY.EQU 16.        ;disk write protected (16.)

DISABLE .EQU 0CFFF      ;disable roms using shared space at C800H
ENABLE  .EQU 0C700      ;enable SVA controller rom
SVASETTRAK .EQU 0C800   ;Areg=track#
SVASETSECT .EQU 0C803   ;Areg=sector#
SVASETBUFR .EQU 0C806   ;Areg=hi pointer byte,
```

```
                              ;Xreg=lo pointer byte
SVASETDISK .EQU 0C809         ;Areg=disk#
SVADSKREAD .EQU 0C80C         ;reads sector
SVADSKWRIT .EQU 0C80F         ;writes sector
SVADSKINIT .EQU 0C812         ;moves head to track 0

        ;jump table must be at beginning of disk driver code
        JMP SETDISK
        JMP SVASETTRAK
        JMP SVASETSECT
        JMP SETBUFR
        JMP DSKREAD
        JMP DSKWRIT
        JMP DSKINIT
        JMP DSKSTRT
        JMP DSKSTOP

SETDISK BIT DISABLE           ;disable shared rom space
        BIT ENABLE            ;enable for disk driver
        AND #1
        JMP SVASETDISK

SETBUFR TAY
        TXA
        JMP SVASETBUFR

DSKSTRT BIT DISABLE
        BIT ENABLE
        LDA USED              ;move USED word to TEMP2 word
                              ;(save zero page)
        STA TEMP2
        LDA USED+1
        STA TEMP2+1
        RTS

DSKSTOP BIT DISABLE
        LDA TEMP2             ;move TEMP2 word to USED word
                              ;(restores zero page)
        STA USED
        LDA TEMP2+1
        STA USED+1
        RTS

DSKINIT JSR SVADSKINIT
        BCC OK
```

```
          LDX #OFFLINE     ;disk off line
          RTS

DSKREAD   JSR SVADSKREAD
          BCC OK
          LDX #READERR     ;disk read error
          RTS

DSKWRIT   JSR SVADSKWRIT
          BCC OK
          LDX #READONLY    ;disk write protected
          RTS

OK        LDX #ONLINE      ;return on line
          RTS

          .END
```

```
;This is an example of an comm card driver
;that will assemble and
;execute when linked in a IV.0 Apple Sbios.

;*******************************************************
;
;       CONSOLE for Apple II(TM) sbios
;
;*******************************************************
        .PROC CONSOLE
        .DEF CONINIT,CONSTAT,CONREAD,CONWRIT
        .REF KEYINIT,KEYSTAT,KEYREAD


SLOT    .EQU 3
CARDADDR.EQU 0C000+<SLOT*100H>
PORTADDR.EQU 0C080+<SLOT*10H>

ONLINE  .EQU 0

;==================================
;       COMM CARD
; driver to be used if an Apple
; comm card is in console slot
;==================================
STATUS  .EQU PORTADDR+0EH       ;status port address for
                                ;6850 ACIA chip
        ;read status port:
        ; 0=RDRF,1=TDRE,2=DCD,3=CTS,4=FmErr,5=Ovrn,6=Parity,7=IRQ
        ;write status port:
        ;  XXXXXX01=initial baud rate divide
        ;  XXXXXX11=master reset
        ;  XXX000XX=7 bits, even parity, 2 stop bits
        ;  XXX001XX=7 bits, odd  parity, 2 stop bits
        ;  XXX010XX=7 bits, even parity, 1 stop bit
        ;  XXX011XX=7 bits, odd  parity, 1 stop bit
        ;  XXX100XX=8 bits and 2 stop bits
        ;  XXX101XX=8 bits and 1 stop bit
        ;  XXX110XX=8 bits, even parity, 1 stop bit
        ;  XXX111XX=8 bits, odd  parity, 1 stop bit
        ;  X00XXXXX=RTS=low, Transmit interrupt disabled
        ;  0XXXXXXX=recieve interrupts disabled
PORT    .EQU PORTADDR+0FH       ;input/output port address
```

```
BOOTFLAG.BYTE 0

CONINIT LDA BOOTFLAG            ;only initialize card once
        BNE $10
        LDA #00000011T          ;master reset signal
        STA STATUS
        LDA #00010001T          ;8 data bits and 2 stop bits
        STA STATUS
        DEC BOOTFLAG            ;note that card has been
initialized
$10     LDX #ONLINE             ;return on line
        RTS

CONSTAT LDX #ONLINE             ;assume console online
        LDA STATUS
        TAY
        LSR A
        BCC $10                 ;branch if no character in buffer
        TYA
        AND #00100000T          ;mask off overrun bit
        BNE $5                  ;branch if overrun has occurred
        LDA #0FF                ;char available
        RTS
$5      BIT PORT                ;clear overrun condition
$10     LDA #0                  ;no char available
        RTS

CONWRIT PHA                     ;save char to print
$5      LDA STATUS
        AND #00000010T          ;transmit buffer empty?
        BEQ $5
        PLA                     ;restore char to print
        STA PORT                ;transmit character
        LDX #ONLINE             ;return on line
        RTS

CONREAD LDA STATUS
        LSR A                   ;character recieved?
        BCC CONREAD             ;branch if no char recieved
        LDY PORT                ;get character from port
        ; check for overrun and if true clear overrun condition
        LDA STATUS
        AND #00100000T          ;mask off overrun bit
        BEQ $10                 ;branch if no overrun
        BIT PORT                ;clear overrun condition
```

```
$10     TYA
        LDX #ONLINE             ;return on line
        RTS

        .END
```

IV.1.11 Polling the Peripherals for I/O

The POLLING routine places any I/O characters in the proper queue. It should be called from any time consuming SBIOS routine (such as a disk driver). You may call POLLING by declaring:

```
.REF   POLLING
```

within your routine and then simply performing a:

```
JSR   POLLING
```

at reasonable intervals. POLLING is called by the p-System before every sector read/write and during character I/O to the console, printer, and remote line.

NOTE: Calls to POLLING may destroy values in the SBIOS Zero Page Temporaries. No registers are destroyed by POLLING, however.

IV.1.12 Configuring DISKVECT

DISKVECT is a tiny 6502 assembly language procedure that you must alter if you are writing your own disk driver. The source (DISKVECT.TEXT) is provided, along with the code for the standard version. It looks like this:

```
.MACRO VECTORS
.REF %1,%2,%3,%4,%5,%6
.WORD %1,%2,%3,%4,%5,%6
.ENDM

.PROC DISKVECT
VECTORS DISKII,DISKII,DISKII,DISKII,DISKII,DISKII
 ;Drives:  0      1      2      3      4      5
.END
```

What this does is assign physical drives 0 through 5 to the listed disk driver's PROC names. In the standard case, all of the drivers are DISKII (APPLE II or Micro-Sci mini-floppies). To change DISKVECT, edit the source file and put the name(s) for your disk driver routine(s) in the desired spot(s), in place of DISKII. Then reassemble the source. The new DISKVECT.CODE must be linked into SYSTEM.SBIOS (see Chapter III) and SYSTEM.BOOT (see Section IV.5).

NOTE: Initially physical drives 0 through 5 are assigned to p-System drives #4:, #5:, #9:, #10:, #11:, #12: consecutively. This may be changed using the CONFIG utility (see Chapter V).

IV.1.13 Memory Configuration Notes

All memory addresses are word addresses: the low byte must be even (for example, the highest word in memory is FFFE hexadecimal; the highest byte is FFFF).

The SBIOS may use any interrupt vectors it needs without fear of conflicting with the p-System.

The stack pointer must be initialized to 0FF hexadecimal before bootstrap parameters are pushed onto the stack.

The following is the APPLE II memory configuration for 64K systems:

0000-001F	Reserved for user assembly programs
0020-0035	Reserved for user and SBIOS temporary variables
0036-003F	SBIOS system info
0040-005F	SBIOS disk controller variables
0060-006F	SBIOS console driver variables
0070-0077	SBIOS printer driver variables
0078-007F	SBIOS remote driver variables
0080-00FF	Interpreter/BIOS variables
0100-01FF	Processor hardware stack
0200-03FF	APPLEINFO table and reserved space
0400-07FF	Internal screen, displayed half
0800-	Bottom of p-System Heap after booting
0800-08FF	Primary bootstrap from track 0, sector 0 (entry at 0801),
0900-0BFF	Second part of primary bootstrap from track 0, sectors 2,4,6
1000-17FF	Volume #4: directory loaded by primary bootstrap
1800-3???	Tertiary bootstrap from SYSTEM.BOOT (entry at 1800)
-BBFE	Bottom of p-System Stack if using internal console
BC00-BFFE	Internal screen, undisplayed half
-BFFF	Bottom of p-System Stack if using 80 column board
C000-C0FF	Control locations for built in devices
C100-C7FF	ROM space for slots 1 through 7
C800-CFFF	Expansion ROM space
D000-DFFF	(Bank 1) Interpreter
D000-DFFF	(Bank 2) SBIOS
E000-FFF9	Interpreter (continued)/BIOS/Interface
FFFA-FFFF	RAM containing interrupt vectors

IV.1.14 Reconfiguring the Interpreter

The interpreter disk contains code files which may be linked together to form an interpreter configured differently than the SYSTEM.INTERP that is shipped already linked.

These are the relevant files:

Name	Description
INTERP.CODE	Interpreter with no real numbers
BIOS.CODE	A simple BIOS with no input queuing for console, printer, or input (this is the smallest BIOS)
BIOS.C.CODE	BIOS with queuing for console
BIOS.CR.CODE	... queuing for console and remote
BIOS.CRP.CODE	... queuing for console, remote, and printer
APPLEINTER.CODE	SBIOS interface

The SYSTEM.INTERP that is shipped is INTERP.CODE linked with BIOS.C.CODE and APPLEINTER.CODE. With the 64K APPLE you have the same amount of available memory without regard to the size of SYSTEM.INTERP.

To create a new interpreter, you must link the desired code files together. Follow these steps (throughout these examples, user input is underlined):

1. Link the code file.

You must make the following choices:

Whether to use BIOS, BIOS.C, BIOS.CR, and BIOS.CRP. These progressively larger BIOS files. Queuing allows more efficient I/O. Use the BIOS that most closely matches your hardware configuration. One important restriction to note when selecting a BIOS is that queuing won't work on the Apple Serial card and the BIOS you select must not have queuing on the device for which you have connected an APPLE Serial card. For example, if you are using an Apple Serial card for console you MUST use BIOS.CODE, none of the other BIOS's can be used; if you are using an Apple Serial card for remote, you may NOT use BIOS.CR.CODE or BIOS.CRP.CODE.

Once you know what the pieces of your new interpreter will be, you can link them together with the system's linker. The interpreter code file you choose will always be the 'Host file?', and the remaining code files will be entered as 'Lib file?'s, always in the following order:

the INTERP file you have chosen
the BIOS you have chosen
APPLEINTER

and let the output file be the work file. (For more information on the linker, see the Users' Manual, Section VIII.4.)

Example:

At the system Command menu, press 'L' for L(ink. The following prompts appear (<return> means the carriage return key, and comments are in {}):

Host file? INTERP<return>
Lib file? BIOS.CRP<return> {or other BIOS}
Opening BIOS.CRP.CODE
Lib file? APPLEINTER<return>
Opening APPLEINTER.CODE
Lib file? <return>

... {more Linker output}

Output file? <return> {makes *SYSTEM.WRK.CODE}

2. Compress the code file.

   Place the UTILITY disk in drive #5:. At the system Command menu, press 'X' for X(ecute, then '#5:COMPRESS<return>'. The utility COMPRESS shows a series of prompts; answer them as follows:

   Assembly Code File Compress

   Press '!' to escape

   Do you wish to produce a relocatable object file (Y/N)Y

   File to compress : SYSTEM.WRK

   Output file (<ret> for same) : NEW.INTERP

   and COMPRESS will either complete its work, or issue an error message, in which case you must try again. The last address of the last PROC displayed while COMPRESS is running must not exceed 2FFFH. If it does, the interpreter won't fit into the language card on a 64K system.

   COMPRESS is described in the Users' Manual, Section X.1.

3. Change file names

   First, make sure you have at least one bootable backup disk so that you may reboot if something goes wrong. Then, at the system Command menu, press 'F' for f(iler. C(hange SYSTEM.INTERP to OLD.INTERP. Then C(hange NEW.INTERP to SYSTEM.INTERP.

You should now be ready to try booting your system again, with the new interpreter and new SBIOS.

IV.1.15 Miscellaneous Notes

6502 Systems use an expression stack that is limited to 128 words of data. When this stack overflows, the system is halted and reinitialized. Correct UCSD Pascal programs that run on other systems may not run on 6502 Systems. This problem can be avoided by following two rules:

1. Sets must contain no more than 512 elements. (The maximum element can't exceed 511.)

2. Nested set expressions must be written so that there are never more than 3 unevaluated operands as the expression is evaluated from left-to-right. For example:

   A+(B+(C+D))

   requires that all four variables be on the stack before evaluation begins. A safer and equivalent expression would be:

   ((C+D)+B)+A

IV.1.16 Using Real Numbers

To use two word reals, transfer the file SYSTEM.REAL2 from the CODE: disk to your boot disk renaming if SYSTEM.REAL. Then L)ibrary REAL2.CODE, from the SYSTEM2: disk, into your SYSTEM.PASCAL making a new SYSTEM.PASCAL on your boot disk. (Make sure you backup the current SYSTEM.PASCAL.) The same procedure is used for four word reals using SYSTEM.REAL4 instead of SYSTEM.REAL2 and REAL4.CODE instead of REAL2.CODE.

IV.2 The Utility SETUP

SETUP is provided as a system utility called SETUP.CODE. SETUP changes a file that contains details about your terminal, and a few miscellaneous details about the system in general. SETUP can be run, and the data changed, as many times as you desire. After running it, it's important to reboot (or I(nitialize) so that the system will start using the new information. It is also important to backup old data.

The file that SETUP uses to store all of this information is called SYSTEM.MISCINFO. Each system initialization loads it into main memory. New versions of SYSTEM.MISCINFO are created by SETUP, and are called NEW.MISCINFO. Backups are created by renaming or copying SYSTEM.MISCINFO with the filer, and then changing NEW.MISCINFO to SYSTEM.MISCINFO.

SYSTEM.MISCINFO contains three types of information:

1. Miscellaneous data about the system;
2. General information about the terminal; and
3. Specific information about the terminal's various control keys.

IV.2.1 Running SETUP

SETUP is a utility program, and is run like any other compiled program: press 'X' for X(ecute, and then answer the prompt with 'SETUP'<enter>. It will display the word 'INITIALIZING' followed by a string of dots, and then the menu:

SETUP: C(HANGE T(EACH H(ELP Q(UIT [version]

To call any command, just press its initial letter.

H(ELP gives you a description of the commands that are visible on any menu where it appears.

T(EACH gives a detailed description of the use of SETUP. Most of it is concerned with input formats. They are mainly self-explanatory, but if this is your first time running SETUP, you should look through all of T(EACH.

C(HANGE gives you the option of going through a menu of all the items, or changing one data item at a time. In either case, the current values are displayed, and you have the option of changing them. If this is your first time running SETUP, the values given are the system defaults.

Q(UIT has the following options:

H(ELP),

M(EMORY) UPDATE, which places the new values in main memory;

D(ISK) UPDATE, which creates NEW.MISCINFO on your disk for future use;

R(ETURN), which lets you go back into SETUP and make more changes; and

E(XIT), which ends the program and returns you to the Command menu.

Please note that if you have a NEW.MISCINFO already on your disk, D(ISK) UPDATE will write over it.

The section below entitled, "Miscellaneous Notes For Setup," contains a detailed description of the data items in SYSTEM.MISCINFO.

If you use SETUP to change your character set, don't under estimate the importance of using keys you can easily remember, and making dangerous keys like BREAK and ESCAPE hard to hit.

Once you have run SETUP, you should always backup SYSTEM.MISCINFO under some other name (for example, OLD.MISCINFO), then change the name of NEW.MISCINFO to SYSTEM.MISCINFO and reboot or I(nitialize. It is indeed possible to update memory only (rather than create a NEW.MISCINFO), and go on using the system without rebooting, but the results will be lost when the system is rebooted or I(nitialized. In general, M(EMORY) UPDATE is a Q(UIT option that you will use only when experimenting. If you desire, the current in-memory SYSTEM.MISCINFO can be saved by rerunning SETUP and doing a D(ISK) UPDATE before you reboot.

When you reboot or I(nitialize, the new SYSTEM.MISCINFO will be read into main memory and its data used by the system, provided it has been stored under that name on the system disk (the disk from which you boot).

IV.2.2 Miscellaneous Notes for SETUP

The STUDENT bit, one of SYSTEM.MISCINFO's data items, should always be set to FALSE.

The HAS 8510A bit is always FALSE.

HAS WORD ORIENTED MACHINE is always FALSE.

HAS BYTE FLIPPED MACHINE is FALSE.

SETUP and the manual refer to PREFIXED [DELETE CHARACTER]. This refers to the backspace function. Think of it as PREFIXED [BACKSPACE]. It will be FALSE.

If you are using a terminal instead of the internal APPLE console and keyboard, your terminal should be set to run in full duplex, with no auto-echo.

Don't use terminal functions that do a "Delete and close up" on lines or characters—not all terminals have these functions, and so they are supplied through the Screen Oriented Editor's software.

In general, if SETUP prompts for a feature that your terminal doesn't have, set the item to NUL (CTRL-@).

IV.2.3 The Data Items in SYSTEM.MISCINFO

This section enumerates the data items within SYSTEM.MISCINFO. The information in this section is very specific, and you may skip it on first reading. If you have a question about a certain data item, look in this section. The default values for the standard APPLE internal console and keyboard are given. The items are ordered according to SETUP's menu.

NOTE: SETUP frequently makes a distinction between a character which is a key on the keyboard, and a character which is sent to the screen from the UCSD p-System (they aren't necessarily the same).

There are a few characters which you can't change with SETUP. These are CARRIAGE RETURN, LINE FEED (<lf>), ASCII DLE (CTRL-P), and TAB (CTRL-I). ASCII DLE (data link escape) is used as a blank compression character. When sent to an output text file, it is always followed by a byte containing the number of blanks which the output device must insert. If you try to use CTRL-P for any other function, you will run into trouble.

BACKSPACE

When sent to the screen, this character should move the cursor one space to the left. Default: ASCII BS (CTRL-H).

CODE POOL BASE[FIRST WORD]
CODE POOL BASE[SECOND WORD]

The Apple II doesn't use these fields. Default: 0.

EDITOR ACCEPT KEY

This key is used by the Screen Oriented Editor. When pressed, it ends the action of a command, and accepts whatever actions were taken. Default: ASCII ETX (CTRL-C).

EDITOR ESCAPE KEY

This key is used by the Screen Oriented Editor. It is the opposite of the EDITOR ACCEPT KEY--when pressed, it ends the action of a command, and ignores whatever actions were taken. Default and Suggested: ASCII ESC (CTRL-[).

EDITOR EXCHANGE-DELETE KEY

This key is also used by the Screen Oriented Editor. It operates only while doing an X(change, and deletes a single character. Default: CTRL-D.

EDITOR EXCHANGE-INSERT KEY

Like the EDITOR EXCHANGE-DELETE KEY, this only operates while doing an X(change in the Screen Oriented Editor: it inserts a single space. Default: CTRL-K.

ERASE LINE

When sent to the screen, this character erases all the characters on the line that the cursor is on. Default: CTRL-W.

ERASE SCREEN

When sent to the screen, this character erases the entire screen. Default: CTRL-L.

ERASE TO END OF LINE

When sent to the screen, this character erases all characters from (and including) the current cursor position to the end of the same line. Default: CTRL-].

ERASE TO END OF SCREEN

When sent to the screen, this character erases all characters from (and including) the current cursor position to the end of the screen. Default: CTRL-K.

FIRST SUBSIDIARY VOL NUMBER

This entry is the first unit number to be used as a subsidiary volume. For example, if you set it to 14, the first subsidiary volume is device #14:.

NOTE: In previous versions of the UCSD p-System, only 6 block devices were allowed: 4, 5, 9 through 12. Now the number of block devices is configurable. The devices from 9 through "First subsidiary vol number" -1 are now standard block devices. Subsidiary volumes start with the device number indicated by "First subsidiary vol number." The number of subsidiary volumes is determined by "Max number of subsidiary vols." The highest device number allowed for subsidiary volumes or standard block devices.

WARNING: "First subsidiary vol number" must be greater than 8 to allow apace for all of the standard system units (13 is recommended).

HAS 8510A

Is always FALSE.

HAS BYTE FLIPPED MACHINE

Is always FALSE.

HAS CLOCK

Default is FALSE.

HAS EXTENDED MEMORY

This should always be FALSE on the Apple II.

HAS LOWER CASE

Default is FALSE.

HAS RANDOM CURSOR ADDRESSING

Is always TRUE unless your terminal is not a CRT.

HAS SLOW TERMINAL

May be TRUE or FALSE. When this bit is TRUE, the system's menus and messages are abbreviated. It is suggested that you leave this set at FALSE. Default: FALSE.

HAS SPOOLING

This should always be set to FALSE. (The Print Spooler isn't available on the Apple II.

HAS WORD ORIENTED MACHINE

Is always FALSE.

KEYBOARD INPUT MASK

Characters that are received from the keyboard will be logically ANDed with ths value. For the Apple II, set this value to 7F hexadecimal (which throws away the eighth bit).

KEY FOR BREAK

When this key is pressed while a program is running, the program will terminate with a run-time error. Default: ASCII NUL (CTRL-@).

KEY FOR FLUSH

This key may be pressed while the system is sending output (writing to the file OUTPUT). The first time it is pressed, output is no longer displayed, and will be ignored ("flushed") until FLUSH is pressed again. This can be done any number of times; FLUSH functions as a toggle. Note that processing continues while the output is ignored, so using FLUSH causes output to be lost. Default and suggested: ASCII ACK (CTRL-F).

KEY FOR STOP

This key may be pressed while the system is writing to OUTPUT. Like FLUSH, it is a toggle. Pressing it once causes output and processing to stop, pressing it again causes output and processing to resume, and so on. No output is lost; STOP is useful for slowing down a program so the output can be read while it is being sent to the terminal. Default and suggested: ASCII DC3 (CTRL-S).

KEY TO ALPHA LOCK

This character, when sent to the screen, locks the keyboard in uppercase (alpha mode). Default: ASCII DC2 (CTRL-R).

KEY TO DELETE CHARACTER

Deletes the character where the cursor is, and moves cursor one character to the left. Default and suggested: ASCII BS (CTRL-H).

KEY TO DELETE LINE

Deletes the line that the cursor is currently on. Default and suggested: ASCII DEL (CTRL-X).

KEY TO END FILE

Sets the intrinsic Boolean function EOF to TRUE when pressed while reading from the system input files (either KEYBOARD or INPUT, which come from device CONSOLE:). Default and suggested: ASCII ETX (CTRL-C).

KEY TO MOVE CURSOR DOWN
KEY TO MOVE CURSOR LEFT
KEY TO MOVE CURSOR RIGHT
KEY TO MOVE CURSOR UP

These keys are recognized by the Screen Oriented Editor, and are used when editing a document to move the cursor about the screen. Default (in order): CTRL-L, CTRL-H, CTRL-U, CTRL-O.

LEAD IN FROM KEYBOARD

Pressing certain keys generates a two-character sequence. The first character in these cases must always be a prefix, and must be the same for all such sequences. This data item specifies that prefix. Note that this character is only accepted as a lead in for characters where you have set PREFIXED[<itemname>] to TRUE. Default: ASCII NUL (CTRL-@).

LEAD IN TO SCREEN

Some terminals require a two-character sequence to activate certain functions. If the first character in all these sequences is the same, this data item can specify this prefix. This item is similar to the one above. The prefix is only generated as a lead in for characters where you have set PREFIXED[<itemname>] to TRUE. Default: ASCII NUL (CTRL-@).

MAX NUMBER OF SUBSIDIARY VOLS

This field indicates the maximum number of subsidiary volumes that may be on-line at once. Because the p-System Unit Table expands a few bytes with each additional subsidiary volume entry, set this number to the smallest convenient value. (Also see FIRST SUBSIDIARY VOL NUMBER.)

The highest subsidiary volume will be "First subsidiary volume number + "Max number of subsidiary vols" -1. This expression must be less than or equal to 127, which is the highest device number allowed for system units.

MAX NUMBER OF USER SERIAL VOLS

User-defined serial devices aren't available on the Apple II. This should be set to 0.

MOVE CURSOR HOME

When sent to the terminal, moves the cursor to the upper left-hand corner of the screen (position (0,0)). Default: CTRL-Y.

MOVE CURSOR RIGHT

When sent to the terminal, moves the cursor nondestructively one space to the right. Default: CTRL-\.

MOVE CURSOR UP

When sent to the terminal, moves the cursor vertically up one line. Default: CTRL-_ (underline).

NONPRINTING CHARACTER

The character that will be displayed on the screen when a nonprinting character is entered or sent to the terminal while using the Screen Oriented Editor. Default and suggested: '?'.

PREFIXED [<itemname>]

If any two-character sequence must be generated by a key or sent to the screen, the system will recognize that if you set PREFIXED[<itemname>] to TRUE. See the explanations for LEAD IN FROM KEYBOARD and LEAD IN TO SCREEN. All PREFIXED[<itemname>] defaults are FALSE.

SCREEN HEIGHT

The number of lines in your display screen, starting from 1. Default: 24 (base ten).

SCREEN WIDTH

The number of characters in one line on your display, starting from 1. Default: 79 (base ten). This default is intended for the 40 column standard APPLE console. If you have an 80 column display, you may set this to 80. (The standard APPLE 40 column configuration emulates an 80 column screen using the special keys described in Section I.3. The reason for making the default 79, instead of 80, is that the p-System shortens some of the filer prompts if the screen is less than 80 columns wide. This results in easier use of the filer with the 40 column display.)

SEGMENT ALIGNMENT

This should be set to 0 on the Apple II.

STUDENT

Should always be FALSE.

VERTICAL MOVE DELAY

May be a decimal integer from 0 to 11. Many terminals require a delay after vertical cursor movements. This delay allows the movement to be completed before another character is sent. This data item specifies the number of nulls that the system sends to the terminal after every CARRIAGE RETURN, ERASE TO END OF LINE, ERASE TO END OF SCREEN, CLEAR SCREEN, and MOVE CURSOR UP. Default: 0.

IV.2.4 Sample SYSTEM.MISCINFO Configurations

Here is a list of SYSTEM.MISCINFO data items followed by some sample values for the internal console and two popular terminals. Some items in the SETUP menu haven't been included; these are data items that refer to the processor configuration, not your terminal.

These examples represent what we consider reasonable layouts for a few different keyboards, but we don't guarantee that they work for your particular hardware, or match your individual taste.

Terminals:	INTERNAL CONSOLE	HAZELTINE 1500/1510	SOROC IQ120
Data Items:			
BACKSPACE	ctrl-H	backspace	ctrl-H
EDITOR ACCEPT KEY	ctrl-C	ctrl-C	home
EDITOR ESCAPE KEY	esc	esc	esc
ERASE LINE	NUL	NUL	NUL
ERASE SCREEN	ctrl-L	ctrl-\	'*'
ERASE TO END OF LINE	ctrl-]	ctrl-O	T
ERASE TO END OF SCRN	ctrl-K	ctrl-X	Y
HAS LOWER CASE	FALSE	TRUE	TRUE
HAS RAND CURS ADDR	TRUE	TRUE	TRUE
HAS SLOW TERMINAL	FALSE	FALSE	FALSE
KEY FOR BREAK	ctrl-@	break *	break
KEY FOR FLUSH	ctrl-F	ctrl-F	ctrl-F
KEY FOR STOP	ctrl-S	ctrl-S	ctrl-S
KEY TO ALPHA LOCK	ctrl-R	NUL	ctrl-R
KEY TO DELETE CHAR	ctrl-H	backspace	l-arrow
KEY TO DELETE LINE	ctrl-X	shift-DEL	rubout
KEY TO END FILE	ctrl-C	ctrl-C	ctrl-C
KEY TO MV CURS DOWN	ctrl-L	ctrl-K	d-arrow
KEY TO MV CURS LEFT	ctrl-H	backspace	l-arrow
KEY TO MV CURS RGHT	ctrl-U	ctrl-P	r-arrow
KEY TO MV CURS UP	ctrl-O	ctrl-L	u-arrow
LEAD IN FROM KEYBD	ctrl-@	NUL	NUL
LEAD IN TO SCREEN	ctrl-@	~	esc
MOVE CURSOR HOME	ctrl-Y	ctrl-R	ctrl-^
MOVE CURSOR RIGHT	ctrl-\	ctrl-P	r-arrow
MOVE CURSOR UP	ctrl-_	ctrl-L	u-arrow
NON PRINTING CHAR	'?'	'?'	'?'
PREF [DELETE CHAR]	FALSE	FALSE	FALSE
PREF [ED ACCEPT KEY]	FALSE	FALSE	FALSE
PREF [ED ESCAPE KEY]	FALSE	FALSE	FALSE

Terminals:	INTERNAL CONSOLE	HAZELTINE 1500/1510	SOROC IQ120
PREF [ERASE LINE]	FALSE	FALSE	FALSE
PREF [ERASE SCREEN]	FALSE	TRUE	TRUE
PREF [ERASE TO EOLN]	FALSE	TRUE	TRUE
PREF [ERSE TO EOSCN]	FALSE	TRUE	TRUE
PREF [KEY DEL CHAR]	FALSE	FALSE	FALSE
PREF [KEY DEL LINE]	FALSE	FALSE	FALSE
PREF [KEY MV CRS DN]	FALSE	FALSE	FALSE
PREF [KEY MV CRS LT]	FALSE	FALSE	FALSE
PREF [KEY MV CRS RT]	FALSE	FALSE	FALSE
PREF [KEY MV CRS UP]	FALSE	FALSE	FALSE
PREF [MOVE CRS HOME]	FALSE	TRUE	FALSE
PREF [MOVE CURS RT]	FALSE	FALSE	FALSE
PREF [MOVE CURS UP]	FALSE	FALSE	FALSE
PREF [NONPRINT CHAR]	FALSE	FALSE	FALSE
SCREEN HEIGHT	24	24	24
SCREEN WIDTH	79	80	80
STUDENT	FALSE	FALSE	FALSE
VERTICAL MOVE DELAY	0	5	10

* Break is also control-@ on Hazeltines.

IV.3 GOTOXY

GOTOXY is a Pascal UNIT embedded in the operating system. It provides random addressing for your terminal's cursor. There are two GOTOXY units provided with the p-System on the APPLE II. These are the GOTOXY for the standard APPLE internal console, and for the SOROC terminal. If you need to write your own GOTOXY for some other terminal, you will have to code the Pascal unit, compile it, then bind it into the operating system using the utility LIBRARY.

Before you write your own GOTOXY, you should understand the I/O intrinsic UNITWRITE, which is described in Section VI.2 of the Users' Manual.

IV.3.1 Writing Your Own GOTOXY

You may write GOTOXY using either YALOE or the Screen Oriented Editor, whichever you find more convenient.

The purpose and the calling protocol of GOTOXY are quite simple. The procedure is given two parameters, X and Y. They must be in that order, and they must be of type INTEGER. The procedure should position the terminal's cursor at coordinates (X,Y), where (0,0) is home (the upper left-hand corner of the screen). That is all it should do.

To get your GOTOXY to run at all, there are a few things that are required.

First, the name of your unit must be GOTOXY. The name of the procedure itself must be something different.

Second, you must include the pseudo-comment {$U-}. This compiler option allows you to use the predeclared name GOTOXY as the name of your unit—it will become part of the operating system. This comment must be the first line of your source code. If it doesn't look like one of the following lines:

```
(*$U-*)
{$U-}
```

your GOTOXY will not compile.

Finally, the code for GOTOXY should be compiled as a UNIT, as shown in the next section.

Your procedure should check that the values of X and Y are within bounds. If they are off the screen, change them to a value that is on the screen (such as the nearest location along the border—this is what all the sample procedures do).

You will need to move the cursor by a WRITE to the terminal, a repeated set of WRITEs within a loop, or a UNITWRITE of a vector. Using UNITWRITE is recommended—it can speed up your terminal handling by about 10%. (Although if you use UNITWRITE, you can't redirect console output.)

To summarize, your GOTOXY should contain, in order:

1. The pseudo-comment '{$U-}';
2. In the program body, a check to make sure that X and Y are on the screen;
3. A section that fills an array with all the characters you must send to the terminal; and
4. The actual write to the terminal, preferably with UNITWRITE.

NOTE: Some terminals take a bias on X and Y. For example, sending (X+32,Y+32) actually positions the cursor at (X,Y). If your terminal is capable of this, you should include these offsets in your procedure. This will eliminate any problems you might run into with the ASCII DLE (CTRL-P) character, which is always interpreted as a blank-compression character. You don't want to send this value as a cursor control character.

The following section contains a more detailed description of GOTOXY.

IV.3.2 A Recipe for GOTOXY

The following is an example of a GOTOXY unit:

```
{$U-}            { ALWAYS include this compiler directive. }
UNIT GOTOXY;

INTERFACE

PROCEDURE AGOTOXY(X,Y: INTEGER);


IMPLEMENTATION

PROCEDURE AGOTOXY;

CONST  TELL_LENGTH_MINUS_1 = 3,
        OFFSET = 32;
{ You may have to change these, depending on your terminal. }

VAR    TELL: PACKED ARRAY [0..TELL_LENGTH_MINUS_1]
             OF 0..255;

BEGIN
  IF X>79 THEN X:=79
    ELSE IF X<0 THEN X:=0;
  IF Y>23 THEN Y:=23
    ELSE IF Y<0 THEN Y:=0;
  {  This range-checking is necessary.  The actual
     screenwidth and height may be different for you. }

  { These first elements of TELL must contain
    the characters which tell your terminal to
    position the cursor at (X,Y):
    fill in the blanks...          }
  TELL[0] := _____;
  TELL[1] := _____;
  ...
  { The actual X and Y values are usually the
    last things in the array;
    the order may be different on your terminal. }
  TELL[TELL_LENGTH_MINUS_1 - 1]   := Y+OFFSET;
  TELL[TELL_LENGTH_MINUS_1]     := X+OFFSET;

  UNITWRITE(1,TELL,TELL_LENGTH_MINUS_1 + 1)
END {AGOTOXY};

END {UNIT GOTOXY}.
```

Once your GOTOXY is written, it should be compiled to a code file. You may name the code file anything you wish.

A common error is incorrectly entering the comment '{$U-}'. If this isn't the first line in your source file, if the comment contains spaces that aren't shown in this manual, or if there are any other variances, your GOTOXY will not compile. You will get the error message 'GOTOXY predeclared' when you try to compile.

IV.3.3 Binding GOTOXY Using the LIBRARY Utility

The p-System is shipped with the standard APPLE internal console GOTOXY bound into the operating system. If you want to use another GOTOXY, either one you wrote or one that is provided with the p-System, you must bind that GOTOXY into the operating system (SYSTEM.PASCAL).

You should also make sure that the STUDENT bit in SYSTEM.MISCINFO is set to FALSE—otherwise, GOTOXY binding won't work, and you will get the message 'No proc in seg table' when you try to reboot the system.

First, backup your system disk. If the binding works, all will be well, and you will have a functioning system with a new (and hopefully functioning) GOTOXY. If the binding does not work, your system may be destroyed. Make sure you have a backup boot disk.

The LIBRARY is a utility program which is shipped under the name LIBRARY.CODE. To run it, X(ecute LIBRARY.

The first prompt LIBRARY gives you is:

Output file? NEW.PASCAL

the underlined portion is a sample response. Choose any unambiguous name that suits you—this new output file will become the new operating system if all goes well. Be sure you have enough room on your disk for the new system: most systems are from 80 to 120 blocks long. If there isn't enough room on your disk, either use the filer's K(runch command to create more room, or use another disk with more room.

LIBRARY then asks:

Input file? MYGOTO.CODE

the underlined portion is a sample response. This should be the file that contains your compiled GOTOXY procedure. It will be displayed in slot 0 of the input file. You must move it to a slot in the output file (this new slot must be greater than 15).

Press 'T'. The INTERFACE part of your unit won't be copied.

Press '0'. LIBRARY prompts:

Copy from slot 0?

press <space>. LIBRARY prompts:

Copy to which slot? 16

respond with a number greater than 15 (as shown).

Now press 'N' for N(ew. This causes a repeat of the prompt:

Input file? SYSTEM.PASCAL

enter in the name of your operating system, as shown. This is the new input file.

Finally, press 'E' for E(very. This will cause all of the slots in SYSTEM.PASCAL to be transferred to the output file, except for GOTOXY, which won't be destroyed because it is already there.

Before using E(very, your screen should look like this:

```
Library: N(ew, 0-9(slot-to-slot, E(very, S(elect, C(omp-unit, F(ill,?

Input file: SYSTEM.PASCAL
  0 u KERNEL     1490      9 u SCREENOP    854     18 u SMALLCOM    163
  1 s PRINTERR    650     10 s SEGSCINI    402     19 u COMMANDI    983
  2 s INITIALI   1139     11 u CONCURRE    146     20 u REALOPS    1270
  3 s GETCMD     3599     12 u PERMHEAP    190
  4 u PASCALIO    765     13 u OSUTIL      235
  5 u EXTRAIO     248     14 u FILEOPS    2032
  6 u HEAPOPS     246     15 s USERPROG   1605
  7 u EXTRAHEA    699     16 u GOTOXY       55
  8 u STRINGOP    236     17 u SOFTOPS     604

Output file: NEWSYS.CODE
  16 u GOTOXY       29
```

NOTE: There is a GOTOXY in the SYSTEM.PASCAL that is shipped. This will be abandoned by the E(very command, since you have already put a GOTOXY in the output file.

Pressing 'Q' for Q(uit causes the changes you have made to be saved in your output file.

Once you are out of LIBRARY, use the filer to change the name of SYSTEM.PASCAL to something like OLD.PASCAL, and NEW.PASCAL (or whatever you have called your new output file) to SYSTEM.PASCAL. Then bootstrap your system again; the new GOTOXY will be in effect.

If at any point while using LIBRARY, you think you have made a mistake, A(bort will exit without recording any changes. When modifying the operating system, it is far better to be safe than sorry.

NOTE: While using LIBRARY on the operating system, <u>never</u> move KERNEL from slot 0 or USERPROG from slot 15.

IV.3.4 Problems

If your newly created system won't bootstrap at all, it may be because you moved the USERPROG segment when you used LIBRARY. USERPROG <u>must</u> be at slot fifteen in SYSTEM.PASCAL. Boot your system's backup, and try again.

If the system starts to boot, but halts with the message 'No unit in seg table', it may also mean that the STUDENT bit is on in your SYSTEM.MISCINFO file. The STUDENT bit must be FALSE when you compile your GOTOXY. Boot your system's backup, change the STUDENT bit to FALSE (using SETUP), recompile your GOTOXY, and use LIBRARY again.

For more information on LIBRARY, see Section VIII.5 in the Users' Manual.

Once LIBRARY has been successfully run, and the system successfully rebooted, you should run SCREENTEST to make sure the Screen Oriented Editor interface will work.

IV.4 SCREENTEST

After you have changed your SYSTEM.MISCINFO, or your GOTOXY, or both, you will want to test the results. SCREENTEST is a utility which accomplishes that. Like SETUP, it is largely self-explanatory. SCREENTEST checks that the interpreter and operating system are sending and receiving characters correctly, that the control keys are set up correctly, and that the Screen Oriented Editor will interface to the terminal as it is supposed to.

When you run SCREENTEST, it will display patterns on the screen and ask you if they are correct. You will need to be seated at your terminal while SCREENTEST is running; it takes roughly five minutes.

SCREENTEST will also output a report of errors to any file you specify. If you do encounter problems, you will need this report to help track them down, especially if you require assistance from your supplier's support group.

IV.4.1 Running SCREENTEST

Press 'X' for X(ecute, and enter 'SCREENTEST<return>'. It will respond by displaying a heading, telling you that all questions must be answered with either 'Y' or 'N' (either uppercase or lowercase; all other characters are ignored), and will then prompt you for the name of an error log file.

If you press <return> instead of specifying a log file name, no error report will be generated. You may want to do this if you are running SCREENTEST for the first time and don't anticipate any problems. If you do have trouble, you can run it again, this time with a log. Sending the log to "PRINTER:" may suit your needs if you have a hard copy device; otherwise, you can save it on a disk file named "LOG.TEXT" or something similar. (The .TEXT suffix is necessary if you want to look at it with the editor.)

If your terminal is set up correctly, you should be able to answer 'Y' to all of the yes/no questions that SCREENTEST asks. If there is any problem with the questions about individual characters, SCREENTEST will tell you immediately. The log file will also contain a record of all problems.

IV.4.2 Results of SCREENTEST

SCREENTEST consists of twelve individual tests. Their names follow:

test_basic
test_clr_screen
test_gotoxy
test_clr_line
test_erase_eol
test_etoeos
test_home
test_single_vectors
test_scroll
test_DLE_expansion
test_keyboard
test_normal_keys

Each of these tests may generate error messages. While the text of each error message is fairly clear, some further explanation follows. The error messages are grouped by the nature of the problems—what you must check in order to solve them. They are further grouped under the name of the test that generates them. This information is included in the error log.

IV.4.3 Problems that can be Fixed by Altering SYSTEM.MISCINFO

If you get any of these error messages, check your SETUP values. To the right of each error message listed below is a suggestion as to which key or character value might be in error. These suggestions won't always pinpoint your problem, but they will tell you what you should check first. It may be the case that changing SETUP doesn't fix your problem. Some special cases are described at the end of this section. If these don't cover your particular problem, you should probably ask for help.

test_clr_screen:

screen not cleared -> is ERASE SCREEN OK?
cursor not left at (0,0) afterwards
-> is MOVE CURSOR HOME OK?

test_clr_line:

didn't clear enough - (x,y)
(where x and y are the cursor co-ordinates)
-> is ERASE LINE OK?
Clearing one line affected another
-> is ERASE LINE OK?

test_erase_eol:

sc_erase_to_eol didn't work
-> is ERASE TO END OF LINE OK?

test_etoeos:

sc_eras_eos didn't work
-> is ERASE TO END OF SCREEN OK?

test_home:

cursor didn't go home
-> is MOVE CURSOR HOME OK?

test_single_vectors:

sc_right didn't work	-> is MOVE CURSOR RIGHT OK?
sc_left didn't work	-> is BACKSPACE OK?
sc_up didn't work	-> is MOVE CURSOR UP OK?
sc_down didn't work	-> this shouldn't happen; call p-System Support!

test_keyboard:

<key> not correct	-> is <key> OK? <key> means one of the following: KEY TO MOVE CURSOR DOWN KEY TO MOVE CURSOR LEFT KEY TO MOVE CURSOR RIGHT KEY TO MOVE CURSOR UP BACKSPACE EDITOR ACCEPT KEY EDITOR ESCAPE KEY KEY TO DELETE LINE KEY TO END FILE

test_normal_keys:

Can't type these - <list>	-> <list> means a list of any standard printing characters; this usually means that a standard character is being interpreted as a special key, which usually happens when HASPREFIX is incorrect—it should be FALSE for a key which needs no prefix, or TRUE for a key which does need one; check your own terminal manual;

IV.4.4 Problems that can be Fixed by Changing GOTOXY

test_gotoxy:

gotoxy(0,0) did not go home
gotoxy(screenwidth-1,screenwidth) not ok
box not correctly drawn
exhaustive_gotoxy_check: first pass not ok
exhaustive_gotoxy_check: top line not ok
-> all these problems relate to your GOTOXY procedure; if you find any discrepancies, you will have to change it; refer to the previous section in this document for a description of using GOTOXY, and to the first paragraph in the miscellaneous notes below;

IV.4.5 Other Problems

test_basic:

not all characters written out
-> there is a problem with the p-System intrinsic UNITWRITE, or with the SBIOS. Disregard the rest of SCREENTEST's results until this particular problem is cleared up;

test_scroll:

sc_down at bottom didn't scroll properly
-> there is a note below about scrolling;

test_DLE_expansion:

expansion not happening properly
-> there is a problem in your interpreter's terminal handling; this may be hardware-related; it is still possible to run with improper DLE expansion—you may encounter off-by-one errors and the like in your output and your editing; DLE is an ASCII character used as a blank-compression code to save space in output strings;

IV.4.6 Miscellaneous Notes on SCREENTEST Problems

The system interprets an ASCII DLE or chr(16) (base ten) within a text file as a blank-compression code (this is its standard use). It can lead to problems if GOTOXY ever writes out a chr(16) as an X or Y value. If you run into this problem, check whether your terminal can handle an offset on X and Y values, that is, whether sending it X+32 and Y+32 will position the cursor at (X,Y) (the value 32 is just an example). If so, this will fix your problem. If not, you will have to modify GOTOXY so it catches this situation; see above.

Some terminals won't scroll at all, or scroll two lines at a time. The p-System's Screen Oriented Editor unfortunately can't handle these terminals—you must use YALOE for SYSTEM.EDITOR.

Use your judgment when interpreting the results of SCREENTEST: if something is reported as an error, but the Screen Oriented Editor performs to your satisfaction, don't worry about the SCREENTEST evaluation.

IV.5 Creating Your Own Bootstrap

You may wish to bootstrap from a disk other then the APPLE II mini-floppies or the Micro-Sci A-40/70 mini-floppies. For example, you may have 8" floppies or a hard disk that you want to use as the system disk. In this case, you will need to write your own Bootstrap. This section details what a bootstrap is and how to write one.

IV.5.1 The Concept of Booting

"Booting" or "bootstrapping" is the process of starting a software system on hardware which is running either no software at all, or a totally different system. The term comes from the phrase "pulling yourself up by the bootstraps"; a bootstrap is essentially a program which (starting from scratch) loads another program and then transfers control to that program.

The UCSD p-System runs on a virtual "p-machine", which on most microprocessors is emulated by the system's interpreter. The task of the bootstrap is to load the interpreter, associated low-level I/O routines, and portions of the operating system, and then start the interpreter's execution.

IV.5.2 Primary, Secondary, and Tertiary Bootstraps

The bootstrap is divided into three separate parts. This section summarizes the actions of each. If you are booting from a disk other than a standard APPLE II mini-floppy or a Micro-Sci A-40/70 mini-floppy, you will only need to rewrite the Primary Bootstrap. The descriptions of the Secondary and Tertiary Bootstraps, which reside in SYSTEM.BOOT, are for your information only.

The Primary Bootstrap:

1. Puts a copy of the APPLEINFO Table at 0200H
2. Loads the directory from blocks 2,3,4,5 into 1000H.
3. Searches directory to find SYSTEM.BOOT.
4. Loads SYSTEM.BOOT starting at 1800H.
5. Pushes hardware configuration parameters onto the stack:
   1: Low memory word (Top of stack, last pushed)
   2: High memory word
   3: Maximum sectors per track
   4: Maximum bytes per sector
   5: Pointer to directory
6. Jumps to 1800H (the secondary bootstrap).

The Secondary Bootstrap:

1. Initializes the BIOS (which is part of this bootstrap).
2. Searches the directory for the interpreter (SYSTEM.INTERP).
3. Loads the interpreter and SYSTEM.SBIOS.
4. Transfers control to the tertiary bootstrap (which is included within SYSTEM.BOOT).

The Tertiary Bootstrap (whose code is linked into SYSTEM.BOOT along with the Secondary Bootstrap):

1. Saves the BIOS initialization words (which are on the stack).
2. Initializes some hardware devices and peripherals.
3. Locates SYSTEM.PASCAL (the operating system).
4. Reads block 0 of the operating system in order to initialize the system's environment.
5. Reads the kernel and initialization segments of the operating system.
6. Initializes the p-machine.
7. Starts execution of the operating system.

IV.5.3 The Standard APPLE Bootstrap

This section details the standard APPLE bootstrap routines.

When the machine is turned on, or the reset key is pressed, the 6502 processor jumps to the Boot ROM. The Boot ROM is located at C600-C6FF hexadecimal (the disk controller card). It loads sector 0 of track 0 from the boot disk into 0800-08FF hexadecimal and jumps to the beginning of that code. This is the first sector of the of the Primary Bootstrap. If you are writting your own Primary Bootstrap, you will need to place it on the boot disk where it will be loaded by the boot ROM.

The Primary Bootstrap loads the last three sectors of itself into 0900-0BFF hexadecimal. Included within the Primary Bootstrap is the APPLEINFO table. A copy of this table is placed at 0200H. (You are supplied with the source for the APPLEINFO table. You must include this in your Primary Bootstrap and move it to 0200H. For more information about this table, see the CONFIG Utility in Chapter V.) The Primary Bootstrap then reads the disk directory into RAM starting at 1000 hexadecimal. Then it reads SYSTEM.BOOT into RAM starting at 1800 hexadecimal. Finally, it jumps to 1800 hexadecimal. Note that this code makes calls to the BOOTROM code to read sectors from the boot disk.

The Secondary and Tertiary Bootstraps are now located from 1800 to 3??? hexadecimal (they may not exceed 3FFF hexadecimal).

The Secondary Bootstrap reads SYSTEM.SBIOS (which is relocatable) into RAM and relocates it to just below BFFF hexadecimal if you have a 48K machine; or to start at D000 hex in bank 2 of the language card if you have a 64K machine. It then sets the Zero Page pointer to the SBIOS. It reads SYSTEM.INTERP (which is relocatable) into RAM and relocates it to start just below SBIOS if you have a 48K machine, or to start at D000 (in bank 1 of the language card) is this is a 64K machine.

The Tertiary boot loads KERNEL and USERPROG onto the stack. It creats SIBs and TIBs (p-machine data structures). It initializes the interpreter, BIOS and SBIOS. It then starts execution of the interpreter by jumping to the instruction fetch loop.

The Secondary and Tertiary bootstraps are self-contained. Within them is a copy of some of the SBIOS drivers.

IV.5.4 Example Primary Bootstrap Outline

The following is an outline for the standard Primary Bootstrap routine provided with the UCSD p-System on the APPLE II:

```
          .PROC PRIBOOT

DIRBLOCK .EQU 2           ;directory start block
DIRSIZE  .EQU 8           ;directory size in 256
                          ; byte sectors
DIRECTORY.EQU 1000H       ;directory load address
SECBOOT  .EQU 1800H       ;secondary boot load
                          ; address

ENTRY     JMP PRIMARY     ;will change as loading
                          ; proceeds
         .BYTE 0          ;filler (PARAMS must start
                          ; at offset 4 for CONFIG)

PARAMS    .WORD 0800      ;address of lowest word in
                          ; contiguous memory
          .WORD 0BFFE     ;address of highest word of
                          ; contiguous memory
          .WORD 26.       ;maximum number of sectors
                          ; per track for all disks
          .WORD 256.      ;maximum number of bytes
                          ; per sector for all disks
          .WORD DIRECTORY
PARMSIZE .EQU *-PARAMS

PRIMARY ...code to perform primary bootstrap...
         ...
         ...
         JMP SECBOOT
```

```
        .ALIGN 768.
;Must be at 768. for CONFIG to work
;APPLEINFO tables, these must be at offset 768.
;(2nd page of block1) in bootstrap for CONFIG.
;At run-time APPLEINFO will be in main memory for
;access by SBIOS in SYSTEM.BOOT, SBIOS in
;SYSTEM.SBIOS, and formatter.
INFOTABLE     ;disk format table
        ;     ntrks,nscts,bytes,intrlv,first,skew
        .WORD   35.,  16., 256.,     2,    0,   0
        .WORD   35.,  16., 256.,     2,    0,   0
        .WORD   35.,  16., 256.,     2,    0,   0
        .WORD   35.,  16., 256.,     2,    0,   0
        .WORD   35.,  16., 256.,     2,    0,   0
        .WORD   35.,  16., 256.,     2,    0,   0
PHYSDISK.BYTE 0,1,2,3,4,5 ;physical disk
                          ; translation table
;16 bytes per section:
CONSOLE .BYTE 0.        ;HAS_LC_VIDEO (0=false,255=true)
        .BYTE 0.        ;HAS_LC_KEYBOARD
        .BYTE 17H       ;SHIFT_KEY (ascii code)
        .BYTE 01H       ;FLIP_KEY
        .BYTE 1AH       ;FOLLOW_KEY
        .BYTE 0         ;USE_INVERSE_LC
        .BYTE 0         ;HAS_SHIFT_WIRE
        .BLOCK 9.       ;unassigned
PRINTER .BYTE 255.      ;HAS_LC_PRINTER
        .BYTE 255.      ;NEEDS_LINEFEED
        .BLOCK 14.      ;unassigned
REMOTE  .BLOCK 16.      ;unassinged
USER    .BLOCK 16.      ;unassinged
CLOCK   .BLOCK 16.      ;unassigned

        .ALIGN 256.
        .END
```

IV.5.5 Placing a Primary Bootstrap on the Disk

When you have written your Primary Bootstrap, you need to place it within the first two blocks of the disk. To do this, simply transfer the bootstrap code file to the disk volume, using the T(ransfer command of the filer. As long as the file is 2 blocks or less in size (and it must be), the filer will allow you to make the transfer. This will install the bootstrap on the disk without destroying the directory.

Another way to move a bootstrap into place is to do a volume-to-volume T(ransfer in the filer specifying a source disk which already contains a bootstrap. The bootstrap will be copied along with the rest of the disk.

Alternatively, the Primary Bootstrap may use Track 0 and the p-System volume may start on Track 1. In this case, the utility ABSWRITE may be used to write the bootstrap onto Track 0.

IV.5.6 Creating SYSTEM.BOOT

If you have written your own disk driver routine, you will need to create an new SYSTEM.BOOT with that driver linked into it. This is a similar process to creating a SYSTEM.SBIOS (where your disk driver must also be present). Call the linker by pressing 'L' at the Command menu. Answer the linker prompts with the following underlined responses:

Host File: APPLESEC
Lib File: SBIOS
Lib File: YOURDRIVER
Lib File: DISKVECT
Lib File: EMPTY.CON
Lib File: EMPTY.PRUC
Lib File: APPLETERT
Lib File: BIOS
Lib File: INTERRUPTS
Lib File: KEYBOARD
Lib File: ENDMARK
Lib File: <return>
Map File: <return>
Output File? NEW.BOOT

At this point, you must X(ecute Compress (on the utilities disk) and compress NEW.BOOT into a nonrelocatable file whose starting address is 1800H. Use "NEW.BOOT" as the input and output file names.

APPLESEC and SBIOS are each a multiple of 256 bytes in length, and they both must start on a page boundary. If YOURDRIVER must start on a page boundary, it should immediately follow SBIOS. If you have more than one driver that must start on a page boundary, all of the routines linked before those drivers must be multiples of 256 bytes. The rest of the files may be in any order. ENDMARK must be last.

When you have created NEW.BOOT, use the filer to change SYSTEM.BOOT to OLD.BOOT, and NEW.BOOT to SYSTEM.BOOT. Make sure you have at least one bootable backup disk, so that you may reboot in the standard configuration if something goes wrong.

CHAPTER V

SPECIAL SOFTWARE FACILITIES

This chapter details several software facilities that are available with the UCSD p-System on the Apple Computer.

V.1 The APPLESTUFF Unit

The APPLESTUFF Unit contains routines which interact with the Apple Computer's hardware. The following is the Interface section of the APPLESTUFF Unit:

```
UNIT APPLESTUFF;

INTERFACE

TYPE
  AS_PADDLE_NUM = 0..3;
  AS_BUTTON_NUM = 0..2;
  AS_TTLOUT_NUM = 0..3;
  AS_BYTE = 0..255;

FUNCTION PADDLE (SELECT:AS_PADDLE_NUM): AS_BYTE;
FUNCTION BUTTON (SELECT:AS_BUTTON_NUM): BOOLEAN;
PROCEDURE TTLOUT (SELECT:AS_TTLOUT_NUM; DATA:BOOLEAN);
FUNCTION KEYPRESS: BOOLEAN;
FUNCTION RANDOM: INTEGER;
PROCEDURE RANDOMIZE;
PROCEDURE NOTE (PITCH:AS_BYTE; DURATION:AS_BYTE);
PROCEDURE NOISE (PITCH:AS_BYTE; DURATION:AS_BYTE);
PROCEDURE VIDEOMODE (MODE:INTEGER);
PROCEDURE PALETTE (COLOR:INTEGER);
```

The remaining paragraphs of this section describe the APPLESTUFF routines.

FUNCTION PADDLE (SELECT:AS_PADDLE_NUM): AS_BYTE;

This function selects one of the four Analog Input locations described in the APPLE II Reference Manual. The value of the parameter SELECT determines which input location is read. The function returns a value (0 through 255) which indicates the setting of the game paddle associated with that input location.

FUNCTION BUTTON (SELECT:AS_BUTTON_NUM): BOOLEAN;

This function selects one of the three One-Bit Inputs (described in the APPLE II Reference Manual) depending upon the value of the parameter SELECT. It returns a boolean function value which indicates whether or not the push-button associated with that input is depressed.

PROCEDURE TTLOUT (SELECT:AS_TTLOUT_NUM; DATA:BOOLEAN);

This procedure selects one of the four Annunciator Outputs, again depending upon the value of the parameter SELECT. The value of the parameter DATA will determine if the referenced annunciator will be turned "on" or "off."

FUNCTION KEYPRESS: BOOLEAN;

This function will return true if a character is waiting in the type ahead queue, and false otherwise.

FUNCTION RANDOM: INTEGER;

This function returns a ramdom integer.

PROCEDURE RANDOMIZE;

This function gives a new seed to the RANDOM function (based on I/O transactions).

PROCEDURE NOTE (PITCH:AS_BYTE; DURATION:AS_BYTE);
PROCEDURE NOISE (PITCH:AS_BYTE; DURATION:AS_BYTE);

Procedure NOTE produces a note on the APPLE speaker. Procedure NOISE produces a pitched white noise sound.

For parameter DURATION, a value of 255 is longest, and a value of 0 is shortest. For parameter PITCH, a value of 0 indicates a rest, a value of 1 indicates a single click of the speaker, and the values 2 through 50 indicate 49 notes, each a half step apart (four octaves all together).

PROCEDURE VIDEOMODE (MODE:INTEGER);
PROCEDURE PALETTE (COLOR:INTEGER);

These two procedures work together to determine what colors may be selected by procedures within the TURTLEGRAPHICS Unit described in Section V.2.). The video mode may be set to any of the values 0 through 4, by calling VIDEOMODE with parameter MODE set to the appropriate value. The palette color may be set to 0 or 1, and only has an effect upon video mode 2. The following video modes cause the indicated colors to be available from TURTLEGRAPHICS:

MODE 0:

This is the normal console I/O mode. If the APPLE display is used, it will be configured for APPLE "TEXTMODE."

MODE 1:

This is the APPLE "LORES" graphics mode. It has 40 pixels (light dots) horizontally, and 48 pixels vertically. Sixteen colors are available. Procedure PENCOLOR within TURTLEGRAPHICS may choose among:

```
 0 = black
 1 = magenta
 2 = dark blue
 3 = light purple
 4 = dark green
 5 = grey
 6 = medium blue
 7 = light blue
 8 = brown
 9 = orange
10 = grey
11 = pink
12 = green
13 = yellow
14 = blue-green
15 = white
```

MODE 2:

This is a version of APPLE "HIRES" graphics supporting 4 colors from one of two palettes. The screen has 140 pixels horizontally and 192 pixels vertically. Depending upon the value set by procedure palette, the following are the available colors in this mode:

```
PALETTE (0)
  0 = black
  1 = violet
  2 = green
  3 = white

PALETTE (1)
  0 = black
  1 = blue
  2 = orange
  3 = white
```

MODE 3:

This version of APPLE "HIRES" graphics mode allows 6 colors. When certain colors occur in close proximity, however, some unexpected results may occur. (This mode is for real APPLE afficionaddos!) The screen resolution is again 140 x 192. The color mapping is as follows:

```
0 = black
1 = violet
2 = green
3 = blue
4 = orange
5 = white
```

MODE 4:

This video mode is for use on black and white displays only. It can be used on color monitors only by turning the color saturation control to its lowest intensity. It provides the highest resolution available on an APPLE, 280 x 192.

```
0 = black
1 = white
```

V.2 The TURTLEGRAPHICS Unit

Turtlegraphics is a package of routines for creating and manipulating images on a graphics display. These routines can be used to control the background of the screen, draw figures, alter old figures, and display figures using viewports and scaling. It also contains routines that allow the user to save figures in disk files and retrieve them.

The simplest Turtlegraphics routines are intentionally very easy to learn and use. Once the user is familiar with these, more complicated features (such as scaling and pixel addressing) should present no problem.

A "pixel," by the way, is a single "picture element" or dot on the display.

Turtlegraphics allows the user to create a number of "figures," or drawing areas. One such figure is the display screen itself, and other figures may be saved in memory. Each figure has a turtle of its own. The size of a figure may be set by the user (it does not need to be the same size as the actual display). See Section V.2.7.

The actual display is addressed in terms of a display scale, which may be set by you. This allows your own coordinates to be mapped into pixels on the display. All other figures are scaled by the global display scale. See Section V.2.6

You may define a "viewport," or window on the display. This limits all graphic activity to within that port. See Section V.2.7.

Each subsection below is divided into two parts. The first part is an overview of the topic at hand, and the second part consists of descriptions of the relevant Turtlegraphics routines.

For quick reference, Section V.2.10 contains a listing of the Interface part of the Turtlegraphics unit.

Section V.2.11 contains a sample program which illustrates a number of the Turtlegraphics routines.

V.2.1 Installing and Initializing TURTLEGRAPHICS

The TURTLEGRAPHICS package is sold separately from the rest of the UCSD p-System. If you have purchased TURTLEGRAPHICS you should have received a disk called TURTLE:. It contains the files:

TURTLE2.CODE
TURTLE4.CODE
USRGRAFS.CODE
SYSTEM.FONT
CHARSIZE.CODE
CHAREDIT.CODE

TURTLE2.CODE or TURTLE4.CODE is the main collection of graphics routines that make up the package. TURTLE2.CODE corresponds to the 2-word real version and TURTLE4.CODE corresponds to the 4-word real version. The remaining text will refer to ths code file as TURTLEx.CODE. (You are free to choose the real size.) It must either be installed within *SYSTEM.LIBRARY (the "*" indicates that the file resides on the boot disk) or a USERLIB must be created so that the p-System can find the unit within TURTLEx.CODE. Both of these processes are described below. SYSTEM.LIBRARY must contain Applestuff if you are going to use Turtlegraphics.

USRGRAFS.CODE contains special initialization and termination code used by TURTLEGRAPHICS. You must install it, along with TURTLEx.CODE into *SYSTEM.LIBRARY, or create an appropriate USERLIB.

SYSTEM.FONT defines the character set used by procedures WCHAR and WSTRING (described in Section V.2.5). SYSTEM.FONT must be T(ransfered onto the boot disk using the filer.

V.2.2 TURTLEGRAPHICS Character Fonts

Turtlegraphics allows programs to label figures by calling two special routines, WChar and WString. These routines draw characters in figures by using a table stored in a file called *SYSTEM.FONT.

The standard system is shipped with a character font that contains 128 ASCII codes, similar in style to those on the some personal computers. Each character occupies an area 8 pixels high by 8 pixels wide.

Two code files, CHARSIZE.CODE and CAREDIT.CODE, are provided on the TURTLE: disk. Executing the code file CHARSIZE allows you to change the character size in TURTLEGRAPHICS. CHAREDIT enables you to create or edit a character font. To use the character font, save it on the boot disk as SYSTEM.FONT. Note that SYSTEM.FONT must have the same character size as specified by CHARSIZE.

Most of the rest of this section describes the two processes of installing TURTLEx.CODE and USRGRAFS.CODE into *SYSTEM.LIBRARY, and creating a USERLIB file. When you have properly performed one of these two tasks, and when you have T(ransfered SYSTEM.FONT onto the boot disk, TURTLEGRAPHICS will be installed.

During compilation time and during run-time, the p-System needs to locate the two appropriate units which make up TURTLEGRAPHICS. The p-System looks for a file called SYSTEM.LIBRARY on the boot disk to find units. If a file called USERLIB.TEXT exists, the p-System also will examine its contents. USERLIB.TEXT should be a regular text file, created with the editor. Within this text file names of code files which serve as additional libraries of units are expected. So, if you wish, you may enter the editor and create a file with the contents:

```
#5:TURTLEx.CODE
#5:USRGRAFS.CODE
```

Now this text file should be saved on disk as USERLIB ("USERLIB.TEXT" is how it will actually appear). Now, you must be sure that you have a disk in drive #5: containing the two files whenever you run TURTLEGRAPHICS.

Another way to use the USERLIB facility, would be to make the contents of USERLIB.TEXT:

```
TURTLEx.CODE
USRGRAFS.CODE
```

In other words, to leave off the "#5:". In this case, you should use the filer to T(ransfer the code files onto the boot disk. It won't be necessary to have a disk in drive #5: to run TURTLEGRAPHICS if you do it in this manner.

The other way to install TURTLEGRAPHICS is to place the two appropriate units into *SYSTEM.LIBRARY. To do this, X(ecute the utility LIBRARY (on the UTILITY: diskette).

When LIBRARY asks you for an output file, specify 'NEW.LIBRARY'. When it asks you for an input file, specify 'SYSTEM.LIBRARY'. The units within the current SYSTEM.LIBRARY on the boot disk will be displayed. Press 'E', for Every, and all of these units will be moved into the file you are creating.

Then press 'N', for a new input file. When LIBRARY prompts you for this input file, specify '#5:TURTLEx.CODE' (the disk TURTLE: should be placed in drive #5:). Again press 'E', for Every.

Repeat the steps in the last paragraph and indicate '#5:USRGRAFS.CODE' as the new input file. Again, press 'E', for Every.

Now that you have moved all of the standard SYSTEM.LIBRARY, as well as the TURTLEGRAPHICS units, into the new library, press 'Q', for Quit. Then press <return> when asked for a Notice. The output file will be called NEW.LIBRARY.

If there isn't enough room on the boot disk for this output file, you will receive an error message somewhere along the way. In this case you should somehow make more room. You may do this by R(emoving unnecessary files, K(runching the disk, and so on. Alternatively, you may specify another disk as the one to contain the output file.

Use the filer to C(hange SYSTEM.LIBRARY to OLD.LIBRARY, and NEW.LIBRARY to SYSTEM.LIBRARY. (If the new SYSTEM.LIBRARY isn't on the boot disk, T(ransfer it there.) Once your are confident that you have done all of this correctly, you may R(emove OLD.LIBRARY if you wish.

NOTE: If you use this second method of moving the units into *SYSTEM.LIBRARY, be certain that you have a backup of the original SYSTEM.LIBRARY somewhere which is completely separate from the entire process. This will insure you against damaging that file by any mistakes you might make during this process.

It is worth mentioning the $U Pascal compiler option here. Both of the above methods allow access to units during compilation time as well as run-time. There is, however, still another way to access units during (Pascal) compilation time. Rather than simply using the Pascal statement:

```
USES TURTLEGRAPHICS;
```

(as shown in the sample program, Section V.2.11) you may use a statement like:

```
USES {$U turtle:TURTLEx.CODE} TURTLEGRAPHICS;
```

The compiler will search for the disk TURTLE:, and will then find the file TURTLEx.CODE on that disk. The compiler will then be able to USE the interface section of the needed unit. This is a way to compile a program which uses a unit which is not within *SYSTEM.LIBRARY. But, in order to run that program, you will still need to employ one of the two methods described in this section.

NOTE: Unless you have the standard APPLE 40 column display you will probably have to perform some special operations to ready the hardware to display graphic information. You will probably have to switch the display hardware from normal text mode into graphics mode and back. Because of the wide variety of configurations, this responsibility has been left to you. By using procedure TTLOUT within the APPLESTUFF unit, you may reference the Annunciator Outputs, if necessary, to perform any necessary mode switching.

V.2.3 The Turtle

The "turtle" is an imaginary creature that resides on the display screen. It carries with it a "pen," and can be made to draw lines by moving it about the display. The possible movements of a turtle are:

move in a straight line (Move);

move to a particular point on the display (Moveto);

turn, relative to the current direction (Turn);

turn to a particular direction (Turnto).

Thus, the turtle draws straight lines in some given direction. The color of the lines it draws can be specified (Pen_color), and so can the nature of the line drawn (Pen_mode).

Wherever the turtle is located, its position and direction can be ascertained by three functions: Turtle_x, Turtle_y, and Turtle_angle.

NOTE: The turtle may be moved anywhere; it isn't limited by the size of the figure or the size of the display. But only movements within the figure will be visible.

To use the turtle in a figure other than the actual display, you may call Activate_Turtle. We will discuss new figures in Section V.2.7.

The remainder of this section describes the routines that handle the turtle. Since this section is meant to double as a reference, some of the routine descriptions mention features we haven't yet discussed. Just skim over anything you don't yet understand.

Procedure Move (distance: real);

Moves the active turtle the specified distance along its current direction. The turtle leaves a tracing of its path (unless the drawing mode is 'nop'). The distance is specified in the units of the current display scale (see Section V.2.6). The movement will be visible unless the current turtle is in a figure that isn't currently on the display.

Procedure Moveto (x,y: real);

Moves the active turtle in a straight line from its current position to the specified location. The turtle leaves a tracing of its path (unless the drawing mode is 'nop'). The x,y coordinates are specified in the units of the current display scale.

Procedure Turn (rotation: real);

Turns the active turtle by the amount specified (in degrees). A positive angle turns the turtle counterclockwise, and a negative angle turns it clockwise.

Procedure Turnto (heading: real);

Sets the direction (the "heading") of the active turtle to a specified angle. The angle is given in degrees; zero (0) degrees faces the right-hand side of the screen, and ninety (90) degrees faces the top of the screen.

Procedure Pen_color (shade: integer);

Parameter Shade determines what the pen color is. The color that Shade indicates depends upon how procedures VIDEOMODE and PALLETE of the APPLESTUFF unit have been used (see Section V.1).

Procedure Pen_mode (mode: integer);

Sets the active turtle's drawing mode. This mode doesn't change until Pen_mode is called again.

These are the possible modes:

0 = Nop - doesn't alter the figure.

1 = Substitute - writes the current pen color.

2 = Overwrite - writes the current pen color.

3 = Underwrite - writes the current pen color. When the pen crosses a pixel that isn't of the background color, that figure is not overwritten.

4 = Complement - the pen complements the color of each pixel that it crosses. (The complement of a color is its opposite; the complement of the complement of a color is the original color.)

Values greater than 4 are treated as Nop.

(These descriptions apply to movements of the turtle. They have a more complex meaning when a figure is copied onto a figure that is already displayed—see Figures and the Port.)

Function Turtle_x : real;

Returns a real value that is the x-coordinate of the active turtle, in units of the current Display_scale.

Function Turtle_y : real;

Returns a real value that is the y-coordinate of the active turtle, in units of the current Display_scale.

Function Turtle_angle : real;

Returns a real value that is the direction (in degrees) of the active turtle.

Procedure Activate_Turtle (screen: integer);

Specifies to which figure subsequent Turtlegraphics commands are directed. Each invocation of this procedure puts the previously active turtle to sleep and awakens the turtle in the designated figure. When Turtlegraphics is initialized, the turtle in the actual display is awake.

V.2.4 The Display

The color of the display (or any other figure) depends in part on settings determined by procedures VIDEOMODE and PALETTE within the APPLESTUFF unit (Section V.1).

A figure can be filled with a single color (not necessarily the background color) by calling Fillscreen.

NOTE: When Turtlegraphics is initialized, the video mode is set to 2. You may call VIDEOMODE in APPLESTUFF to change this, but the display is cleared, and if the new mode is other than 2, Display_scale must be called immediately to reinitialize the display. At the end of a program using TURTLEGRAPHICS, the VIDEOMODE is restored to 0.

Procedure Fillscreen (screen: integer; shade: integer);

Fills the specified figure ("screen") with the specified color ("shade"). If screen = 0, which indicates the actual display screen, then only the current viewport is shaded. For user-created figures, the entire figure is shaded.

Procedure Background (screen: integer; shade: integer);

Specifies the backgound color for a figure. The initial background color of all figures is black (color 0).

V.2.5 Writing Characters with TURTLEGRAPHICS

It is possible to draw legends, labels, and so forth on the display while using the Turtlegraphics unit. This is done with the following two procedures:

Procedure Wchar (c:char; copymode,shade:integer);
Procedure Wstring (var s:string; copymode,shade:integer);

These procedures will display the character or string arguments at the current active turtle location. They can be used to label either user-created figures, or the actual display screen.

V.2.6 Scaling

When you wish to display data without altering the input data, it is possible to set scaling factors that translate data into locations on the display. This is done with Display_scale. The display scale applies globally to all figures.

Because of the shape of the actual display, data for particular shapes (especially curved figures) might become distorted when using a different display scale. In this case, the function Aspect_ratio can be used to preserve the "squareness" of the figure.

Procedure Display_scale (min_x,min_y,max_x,max_y: real);

Defines the range of input coordinate positions that are to be visible on the display. Turtlegraphics maps your coordinates into pixel locations according to the scale specified in Display_scale.

This procedure sets the viewport (see Section V.2.7) to encompass the whole display. The display bounds apply to input data. For the actual display, these bounds can be any values the user requires, but user-created figures always have (0,0) as their lower left-hand corner.

The default display scale in Videomode 2 is:

```
min_x = 0, max_x = 140
min_y = 0, max_y = 191
```

which is simply the array of pixels on the full display.

As an example, if you wish to graph a financial chart from the years 1970 to 1980 along the x axis, and from $500,000 to $500,000,000 along the y axis, the following call could be used:

```
Display_scale(1970, 5.0E5, 1980, 5.0E8)
```

After this, calls to turtle operations could be done using meaningful numbers rather than quantities of pixels.

Function Aspect_ratio : real;

Returns a real number that is the width/height ratio of the CRT. This can be used to compute parameters for Display_Scale that provide square aspect ratios.

If an application is designed to show information where the aspect ratio of the display is critical (that is, circles, squares, pie-charts, and so on) it must insure that the ratio:

```
(max_x - min_x) / (max_y - min_y)
```

is the same as the aspect ratio of the physical screen upon which the image is being displayed. When the Turtlegraphics unit is initalized, min_x and min_y are set to 0. Max_x is initialized to the number of pixels in the x direction, and max_y is initialized to the number of pixels in the y direction. In order to change to different units that still have the same aspect ratio, a call similar to the following can be used:

```
Display_scale(0, 0, 100*ASPECT_RATIO, 100);
```

This utilizes Function Aspect_ratio described above, and makes the y axis 100 units long.

Turtlegraphics always treats the turtle as being in a fixed pixel location. Changing the scaling of the system with a call to this routine in the middle of a program doesn't alter the pixel position of any of the turtles in the figures. However, the values returned from X_pos and Y_pos may change.

V.2.7 Figures and the Port

You can create and delete new figures, each with its own turtle. When a new figure is created, it is assigned an integer, and this integer refers to that figure in subsequent calls to Turtlegraphics procedures. New figures can be saved (Putfigure) or displayed on the screen (Getfigure).

The actual display is always referred to as figure 0.

The active portion of the display can be restricted by calling Viewport, which creates a "window" on the screen in which all subsequent graphics activity takes place. You might create a figure, specify the port, then display that figure (or a portion of it) within the port. Specifying a viewport doesn't restrict turtle activity, it merely restricts what is displayed on the screen.

User-created figures can be saved in p-System disk files. See Section V.2.9.

Function Create_figure (x_size,y_size: real): integer

Creates a new figure which is rectangular, and has the dimensions (x_size, y_size), where (0,0) designates the lower left-hand corner. The dimensions are in units of the current display scale. The figure is identified by the integer returned by Create_figure.

When a figure is created it contains its own turtle, which is at 0,0 and has a direction of 0 (it faces the right-hand side of the figure). The turtle in a user-created figure can be used by calling Activate_Turtle (described in Section V.2.3).

Procedure Delete_figure (screen: integer);

Discards a previously created display figure area.

Though figures may be created and destroyed, indiscriminate use of these contructs may rapidly exhaust the memory available in the p-System due to Heap fragmentation. For example, a figure may be created using Create_figure (or it may be read in from disk using Function Load_Figure, described below). If possible, after that figure is used (for example, with a Getfigure, Putfigure, Load_figure or Store_figure operation) it should be deleted before other figures are created. If many figures are created, and randomly deleted, the Heap fragmentation problem may occur.

Procedure Getfigure (source_screen: integer;
corner_x,corner_y: real; mode: integer);

Transfers a user-created figure (the "source") to the display screen (the "destination") using the drawing mode specified. The figure is placed on the display such that its lower left-hand corner is at (corner_x, corner_y). The x and y positions are specified in the units of the current display scale. If the display scale has been modified since the figure was created, the results of this procedure are unpredictable.

These are the effects of the drawing mode:

0 = Nop - doesn't alter the destination.

1 = Substitute - each pixel in the source replaces the corresponding pixel in the destination.

2 = Overwrite - each pixel in the source that is not of the source's background color replaces the corresponding pixel in the destination.

3 = Underwrite - each pixel in the source that isn't of the source's background color is copied to the corresponding pixel in the destination only if the corresponding pixel is of the destination's background color.

4 = Complement - for each pixel in the source that isn't of the source's background color, the corresponding pixel in the destination is complemented.

Values greater than 4 are treated as Nop.

If a portion of the source figure falls outside the display or the window, it is set to the source's background color.

Procedure Putfigure (destination_screen: integer;
corner_x,corner_y: real; mode: integer);

Transfers a portion of the display screen to a user-created figure using the drawing mode specified (see above). The portion transferred to the figure is the area of the display that the figure covers when it is placed on the display with its lower left-hand corner is at (corner_x, corner_y). If the display scale has been modified since the figure was created, the results of this procedure are unpredictable.

NOTE: When a figure is moved to the display by Getfigure, further modifications to the display do not affect the copy of the figure that is saved in memory. If you wish to save the results of graphics work on the display, it is necessary to call Putfigure.

Procedure Viewport (min_x,min_y, max_x,max_y: integer);

Defines the boundaries of a "window" which confines subsequent graphics activities. The Viewport procedure applies only to the actual display. When a window has been defined, graphics activities outside of it are neither displayed nor retained in any way. Therefore, lines, or portions thereof, that are drawn outside the window are essentially lost and won't be displayed (this is true even if the window is subsequently expanded to encompass a previously drawn line). The viewport boundaries are specified in the units of the current display scale. If the specified size of the viewport is larger than the current range of the display, the Viewport is truncated to the display limits.

V.2.8 Pixels

It is possible to ascertain (Read_pixel) or write (Set_pixel) the color of an individual pixel within a given figure. These routines are more specific than the turtle-moving routines. They are less straightforward to use, but give you greater control.

Function Read_pixel (screen: integer; x,y: real): integer;

Returns the value of the color of the pixel at the x,y location in the specified figure. The x,y location is specified in the units of the current Display_scale.

Procedure Set_pixel (screen: integer; x,y: real; shade: integer);

Sets the pixel at the x,y location of the specified figure to the specified color. The x,y location is specified in the units of the current Display_scale.

V.2.9 Fotofiles

You may create disk files that contain Turtlegraphics figures. New figures may be written to a file, and old figures restored for viewing or modification.

When figures are written to a file, they are written sequentially, and assigned an "index" that is their location in the file. They may be retrieved "randomly" by using this index value.

The p-System name for files of figures always contains the suffix ".FOTO". It isn't necessary to use this suffix when calling Read_figure_file or Write_figure_file (if absent, it will be supplied automatically).

Function Read_figure_file (title: string): integer;

Specifies the title of a file from which all subsequent figures will be loaded. If a figure file is already open for reading when this function is called, it is closed before the new file is opened. Only one figure file may be open for reading at a single time. This function returns an integer value which is the IORESULT of opening the file.

Function Write_figure_file (title: string): integer;

Creates an output file into which user-created figures may be stored. If another figure file is open for writing when this function is called, it is closed, with lock, before the new file is created. Only one figure file may be open for writing at a single time. This function returns an integer result which is the IORESULT of the file creation.

Function Load_figure (index: integer): integer;

Loads the indexed figure from the current input figure file and assigns it a new, unique, figure number. An automatic Create_figure is performed. If the operation fails for any reason, a Figure_number of zero (0) is returned.

Function Store_figure (figure: integer): integer;

Sequentially writes the designated figure to the output figure file. The function returns an integer that is the figure's positional index in the current output figure file. Positional indexes start at one (1). If the index returned equals zero (0), Turtlegraphics didn't successfully store the figure.

V.2.10 Routine Parameters

The following is the interface section for the Turtlegraphics unit, showing the parameters to all Turtlegraphics routines:

```
Unit Turtlegraphics;

Interface

Procedure Display_scale( min_x, min_y, max_x, max_y: real);
Function Aspect_ratio: real;
Function Create_figure( x_size, y_size: real ): integer;
Procedure Delete_figure( screen: integer );
Procedure Viewport( min_x, min_y, max_x, max_y: real );
Procedure Fillscreen( screen: integer; shade: integer);
Procedure Background( screen: integer; shade: integer );
Function Read_pixel( screen: integer; x, y : real ) :integer;
Procedure Set_pixel( screen: integer; x, y: real; shade: color );
Procedure Getfigure( source_screen: integer, corner_x,
                     corner_y: real; mode: integer );
Procedure Putfigure( destination_screen: integer, corner_x,
                     corner_y: real; mode: integer );
Function Read_figure_file( title: string ): integer;
Function Write_figure_file( title: string ): integer;
Function Load_figure( index: integer ): integer;
Function Store_figure( figure: integer): integer;
Procedure Activate_Turtle( screen: integer );
Function Turtle_x: real;
Function Turtle_y: real;
Function Turtle_angle: real;
Procedure Move( distance: real );
Procedure Moveto( x,y: real );
Procedure Turn( rotation: real );
Procedure Turnto( heading: real );
Procedure Pen_mode( mode: integer );
Procedure Pen_color( shade: integer );
Procedure Wchar(c:char; copymode, shade: integer);
Procedure Wstring(var s:string; copymode, shade: integer);
```

V.2.11 Sample Program

Here is a sample program that illustrates a number of Turtlegraphics routines:

```
program Spiraldemo;

  uses Turtlegraphics, Applestuff;

  const  nop = 0;
      substitute = 1;

  var I, J, Mode: integer;
      C: char;
      Color: integer;
      Seed: integer;
      LX, LY, UX, UY: real;

  function Random_Num (Range: integer): integer;
     begin
       Seed:= Seed * 233 + 113;
       Random_Num:= Seed mod Range;
       Seed:= Seed mod 256;
     end;

  procedure ClearBottom;
     {clears bottom line of screen
      for prompts}
     begin
       Penmode (nop);
       Moveto (0, 0);
       WString ('                                ', substitute, 1);
     end;

  begin
    ClearBottom;      {various initializations}
    WString ('ENTER RANDOM NUMBER: ', substitute, 1);
    read(keyboard, Seed);
    ClearBottom;
    Display_Scale (0, 0, 200*Aspect_Ratio, 200);
       {Aspect_Ratio used so
        pattern will be round}
    Color:= 0;
    WString ('ENTER VIEWPORT LL CORNER: ', substitute, 1);
    read(keyboard, LX,LY);
    ClearBottom;
```

```
WString ('ENTER VIEWPORT UR CORNER: ', substitute, 1);
read(keyboard, UX,UY);
ClearBottom;
WString ('PENMODE= ', substitute, 1);
read(keyboard, MODE);

Palette (0);
   {0= black, 1=green, 2=red, 3=yellow}
ViewPort (LX, LY, UX, UY);   {create port}
PenMode (0);
   {use blank pen while moving it}
Moveto (100*Aspect_Ratio, 100);
   {put turtle in center of port}
   {Aspect_Ratio ensures that it will be
    correctly centered}
PenMode (Mode);
   {set pen to selected color}
J:= Random_Num(90)+90;
   {angle by which turtle will move
    note that turtle begins facing right
    and will move counterclockwise
    (J is positive)}

for I:= 2 to 200 do
   {draw spiral in 200 segments
    of increasing length}
   begin
        {cycle through the colors}
     Color:= Color+1;
     if Color > 3 then Color:= 1;
     PenColor (Color);
     Move(I);
     Turn(J);
   end;
```

```
    I:= Create_Figure (UX-LX, UY-LY);
       {create figure the size of the port}
    PutFigure (I, LX, LY, 1);
       {save it; mode overwrites
        old figure (if any)}
    ViewPort (0, 0, Aspect_Ratio*200, 200);
       {respecify viewport in
        the lower left corner}
    GetFigure (I, 0, 0, 1);
       {display finished spiral}
    readln;
       {clear user input buffer}
end.
```

IV.2.12 Memory Space Requirements

When running TURTLEGRAPHICS, the error message 'NO SPACE FOR HIRES DISPLAY' can occur when data or code encroaches on the area of memory set aside for the screen. During initialization, a 4K section is created in low memory from 2000 to 4000 hexadecimal.

The program is loaded after the creation of the hires screen. If the 'NO SPACE' error occurs during load time, it is due to too much data for the space between the current top of the heap and the hires screen. This can be alleviated by declaring fewer global variables or fewer segments. (Each segment has an entry in the segment identifier table which is created at load time. The term segment refers to either units or segment procedures.) Please note that the LOADER is also present at this time.

A good method for saving space is to use pointers to data structures. If you have buffers or other data structures, a pointer to the item can be declared globally and procedure NEW used to create the data structure when needed. This saves loading space as the pointer is only one word. Run-time space can be saved by use of procedure DISPOSE when the structure is not in use.

If the error occurs during execution time, some method will have to be found to make the resident portion of the program smaller. This can be done by making the program smaller overall or breaking the program into smaller segments.

V.3 The CONFIG Utility

The CONFIG Utility allows you to set certain parameters affecting how the p-System handles peripherals. When you X(ecute UTILITY:CONFIG, the following menu appears:

```
Apple-Config: G(et, S(ave, C(onsole, D(isk, P(rinter, R(emote ?
Apple-Config: T(clock, U(ser, X(Patch, B(ootstrap, Q(uiit
```

(The second half of the menu appears if you press '?'.)

G(et

If you press 'G' for G(et you are able to read the APPLEINFO table and the Primary Bootstrap parameters. The following promptline appears:

```
Read from M(emory or D(isk ?
```

If you press 'M', only the APPLEINFO table is read, and the copy that resides in RAM is what you will now be looking at as you continue to use CONFIG. If you press 'D', you will be asked to select which disk drive you want to read from. You will then be prompted to insert the disk you wish to examine and perhaps modify. Both the APPLEINFO table and the Primary Bootstrap parameters will be available for inspection if you read from disk.

You must do a R(ead before you can execute any of the other CONFIG commands.

S(ave

The S(ave option allows you to save any changes you have made. You are prompted to save to M(emory or D(isk. If you save to M(emory, only the APPLEINFO table is saved. If you save to D(isk and had originally read from disk, both the APPLEINFO table and the Bootstrap parameters will be saved. If you save to D(isk and had originally read from memory, only the APPLEINFO table will be saved.

C(onsole

If, from the CONFIG menu, you select C(onsole you will be able to alter the following console parameters:

HAS_LC_VIDEO
HAS_LC_KEYBOARD
SHIFT_KEY
FLIP_KEY
FOLLOW_KEY
USE_INVERSE_LC
HAS_SHIFT_WIRE
CURSOR_CHAR

HAS_LC_VIDEO, HAS_LC_KEYBOARD, USE_INVERSE_LC, and HAS_SHIFT_WIRE are booleans which take a value of TRUE or FALSE. In C(hange mode, simply press 'T' or 'F' followed by <return> for these values (indicating True or False). Alternatively, you may press 'Y' or 'N' followed by <return> (indicating Yes for true, or No for false). Press the actual keys you want to use for the other options, followed by <return>.

If you have an internal console with lowercase video capability, you will want to set HAS_LC_VIDEO to TRUE. If your keyboard will generate both uppercase and lowercase characters, you should set HAS_LC_KEYBOARD to TRUE.

If you want to change the "soft" shift key (see Section I.3), you may enter whatever CTRL-key combination you desire for SHIFT_KEY.

FLIP_KEY refers to the key that flips the 40 column screen from one side to the other. FOLLOW_KEY is the key that causes the 40 column screen to move with the cursor as it moves off the screen. These may be altered in the same manner as the SHIFT_KEY.

USE_INVERSE_LC, if TRUE, indicates that your APPLE system can't display lowercase, and that uppercase letters should, therefore, be displayed as inverse uppercase letters. Lowercase letters will be displayed as regular (not inverse) uppercase characters.

HAS_SHIFT_WIRE indicates that you have wired your APPLE computer so that the "hard" shift key can be recognized for alphabetic characters (see Section I.3).

CURSOR_CHAR may take on a value from 1 to 4 and indicates what type of cursor you desire to have. Its values have the following meanings:

0: No cursor
1: Solid block cursor
2: Flashing block cursor
3: Underline cursor
4: Flashing underline cursor

D(isk

Selecting D(isk will allow you to examine or alter the following disk driver parameters:

Number of TRACKS per disk
Number of SECTORS per track
Number of BYTES per sector
The INTERLEAVE of the sectors
The FIRST interleaved track
The track to track SKEW
The PHYSICAL drive (0..5) association to p-System
volumes (#4:, #5:, #9:, #10:, #11:, #12:)

Simply enter the volume number (4, 5, 9, 10, 11, 12) and then either enter new values for the various parameters followed by <return>, or simply press <return> to leave any value unchanged.

P(rinter

Selecting P(rinter allows access to the following parameters:

HAS_LC_PRINTER
NEEDS_LINEFEED
BAUD_RATE

If your printer has lowercase, you will want to set HAS_LC_PRINTER to TRUE.

NEEDS_LINEFEED may be TRUE or FALSE. If TRUE it indicates that the printer requires a line-feed after a carriage return is sent.

BAUD_RATE may take on a value from 0 to 255. The supplied printer driver assigns the following meaning to the value of BAUD_RATE:

1: 110
2: 150
3: 300
4: 600
5: 1200
6: 2400
7: 4800
8: 9600
9: 19200

R(emote

This option allows access the following serial line paramters:

CONVER_LC
ZERO_BIT7
BAUD_RATE

CONVER_LC indicates that all alphabetic characters should be converted to lowercase before being sent.

ZERO_BIT7 indicates that the parity bit should be set to zero before a character is sent.

BAUD_RATE has the same meaning as it does for P(rinter above.

T(clock and U(ser

These two options aren't currently implemented within the CONFIG Utiltity.

X(Patch

The examine and patch option displays a hexadecimal dump of the APPLEINFO table which you may alter directly. The fields are displayed on the left for convenience. C(hange allows you to select a particular byte offset into the table (in hexadecimal) and alter it (in hexadecimal). You may press <return> to leave the byte unchanged.

The following is an edited assembled listing of the APPLEINFO table showing offsets:

```
0200| INFOTABLE
0200| ;      ntrks,nscts,bytes,intrlv,first,skew
0200| .WORD   35.,  16., 256.,     2,    0,    0
0208|
020C| .WORD   35.,  16., 256.,     2,    0,    0
0214|
0218| .WORD   35.,  16., 256.,     2,    0,    0
0220|
0224| .WORD   35.,  16., 256.,     2,    0,    0
022C|
0230| .WORD   35.,  16., 256.,     2,    0,    0
0238|
023C| .WORD   35.,  16., 256.,     2,    0,    0
0244| ;physical disk translation table:
0248| PHYSDISK.BYTE 0,1,2,3,4,5
024E| ;16 bytes per section:
024E| CONSOLE .BYTE 0     ;HAS_LC_VIDEO (0=false,255=true)
024F|         .BYTE 0.    ;HAS_LC_KEYBOARD
0250|         .BYTE 17H   ;SHIFT_KEY (ascii code)
0251|         .BYTE 01H   ;FLIP_KEY
0252|         .BYTE 1AH   ;FOLLOW_KEY
0253|         .BYTE 0     ;USE_INVERSE_LC
0254|         .BYTE 0     ;HAS_SHIFT_WIRE
0255|         .BLOCK 9.   ;unassigned
025E| PRINTER .BYTE 25    ;HAS_LC_PRINTER
025F|         .BYTE 255.  ;NEEDS_LINEFEED
0260|         .BLOCK 14.  ;unassigned
026E| REMOTE  .BLOCK 16   ;unassinged
027E| USER    .BLOCK 16   ;unassinged
028E| CLOCK   .BLOCK 16   ;unassigned
```

B(ootstrap

If you then select B(ootstrap, the four Primary Bootstrap parameters are displayed, and you have the option of C(hanging them or Q(uitting this option. If you press 'C' to change them, a prompt for each one will appear. You may then enter in a new value followed by <return>, or you may leave the original value unchanged by simply pressing <return>. Pressing 'Q' returns you to the CONFIG menu.

If you are using an 80 column board (which doesn't use screen memory at BC00H-BFFEH) you may set HIWORD to BFFEH instead of BBFEH. This will give you an extra 1024 bytes of memory for use by the p-System.

MAXSECT (maximum number of sectors per track) must be set to correspond to the drive with the highest number of sectors per track of all drives. For example, if you have a mix of drives on-line—some with 16 sectors per track, 26 sectors per track and 32 sectors per track—you would set MAXSECT to 32.

Similarly, MAXBYTE (maximum number of bytes per sector) must be set to the number of bytes per sector of the drive which has the greatest number.

I)dent

If a G)et is done from M)emory, an I)dent will display the current card configuration for PRINTER:, REMOTE:, and CONSOLE:. A table will be displayed as follows:

Slot	Device	Card Type
0	PRINTER:	PRL
1	REMOTE:	SER
2	CONSOLE:	Internal

Where card type can be any of the following:

PRL - Parallel card (PRINTER: slot only)
SER - Serial card
Firm - Firmware card
Internal - Card in the CONSOLE: slot is recognized, internal console
Comm - Communication Card
None - Card is not recognized or no card is in slot

The table listed above signifies a parallel card in the PRINTER: slot, a serial card is in the REMOTE: slot and either no card or a card that isn't recognized is in the CONSOLE: slot. This information can be very useful when linking together an SBIOS. For more information, see Section III.2.

APPENDIX A

THE p-SYSTEM DEVICES AND VOLUMES

Device#	Volume name	Description
1	CONSOLE:	screen and keyboard with echo
2	SYSTERM:	screen and keyboard without echo
3	GRAPHIC:	<not implemented>
4	<disk name>:	the system disk
5	<disk name>:	the alternate disk
6	PRINTER:	a line printer
7	REMIN:	a serial input line
8	REMOUT:	a serial output line
9..12	<disk name>:	additional disk drives

APPENDIX B

APPLE II SLOT ASSIGNMENTS

Apple Slot	Interface Card	Volume	Device#
0	16k Ram Memory Card	n/a	n/a
1	Printer Interface	PRINTER:	6
2	Serial Interface	REMIN:/REMOUT:	7/8
3	Serial Interface to terminal or 80 Column Video Board	CONSOLE:	1
4	Disk Controller	<disk names>	11/12
5	Disk Controller	<disk names>	9/10
6	Disk Controller (required)	<disk names>	4/5
7	unassigned		

NOTE: All interface cards are optional except slot 6.

APPENDIX C

GENERIC APPLE DRIVERS

The console, printer, and remote SBIOS interfaces have been changed to accommodate the wide variety of peripherals used with Apple computers. These drivers will now support most serial, firmware, comm, and parallel printer cards. Most cards that follow Apple's signature protocol will operate on the p-System. The following table summarizes the Apple signature protocol.

Type of Card	Signature		
	CN05	CN07	CN0B (where N=slot number)
Comm	18	38	
Serial	38	18	<>1
Firmware	38	18	= 1
Parallel	48	48	

For example, using a firmware printer card, the address C105H contains 38H, C107H is 18H and location C10BH is 1. In the case of the console, if the 80-column card in slot 3 doesn't adhere to any of the signature descriptions listed in the preceding table, the console will default to the internal console interface, which is supplied in the code file CONSOLE.CODE.

GENERIC DRIVER TEST RESULTS

The Apple Upgrade package includes a code disk that contains a new set of drivers that were developed to widen the range of applicability of the p-System to the wide range of peripherals used with the Apple II computer. The following hardware configuration tables, which are organized according to the 80-column console cards, show the results of testing various card configurations.

The first row of each table contains the mother board slot number and the first column lists the device cards. The character "Y" located at the intersection of the two variables indicates that the particular device card was placed in that particular slot.

CONFIGURATION 1: Internal Console Card Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card		Y						
AIO comm card								
corvus card			Y					
smarterm card								
videx card								
double vision								
diskii card							Y	
micro-sci card						Y		

Errors detected : none
Error description :
Comments : Used with generic drivers in SBIOS

CONFIGURATION 2: Videx Card Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card		Y						
AIO comm card								
corvus card			Y					
smarterm card								
videx card				Y				
double vision								
diskii card							Y	
micro-sci card						Y		

Errors detected : will not boot
Error description :
Comments : used with generic console driver in SBIOS.

CONFIGURATION 3: Videx Card Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card								
AIO comm card								
corvus card			Y					
smarterm card								
videx card				Y				
double vision								
diskii card							Y	
micro-sci card						Y		

Errors detected : None
Error description :
Comments : Used with generic console driver in SBIOS.

CONFIGURATION 4: Videx Card Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card		Y						
AIO comm card								
corvus card			Y					
smarterm card								
videx card				Y				
double vision								
diskii card						Y		
micro-sci card							Y	

Errors detected : None
Error description :
Comments : VIDEX.CODE linked into SBIOS
NOTE: same result without the grappler card.

CONFIGURATION 5: Videx Card as CONSOLE Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card								
AIO comm card			Y					
corvus card								
smarterm card		Y						
videx card				Y				
double vision								
diskii card						Y		
micro-sci card							Y	

Errors detected : None
Error description :
Comments : VIDEX.CODE linked into sbios.

CONFIGURATION 6: Double Vision Card Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card		Y						
AIO comm card								
corvus card			Y					
smarterm card								
videx card								
double vision				Y				
diskii card							Y	
micro-sci card						Y		

Errors detected : Doesn't boot
Error description :
Comments : Using the generic console.code. With the generic console driver, the system will not boot with the double vision card in any hardware configuration.

CONFIGURATION 7: Double Vision Card Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card		Y						
AIO comm card								
corvus card			Y					
smarterm card								
videx card								
double vision				Y				
diskii card						Y		
micro-sci card							Y	

Errors detected : won't boot with grappler card
Error description : Grappler card not supported in combination with double vision card.
Comments : without grappler card will boot and NO ERRORS on console. Uses dblvision.code for the SBIOS.

CONFIGURATION 8: Smarterm Card Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card		Y						
AIO comm card								
corvus card			Y					
smarterm card				Y				
videx card								
double vision								
diskii card						Y		
micro-sci card							Y	

Errors detected : NONE
Error description :
Comments : SMARTERM.CODE linked into SBIOS.
note: same results without grappler card.

CONFIGURATION 9: Smarterm Card Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card								
AIO comm card		Y						
corvus card			Y					
smarterm card				Y				
videx card								
double vision								
diskii card						Y		
micro-sci card							Y	

Errors detected : None
Error description :
Comments : AIO card to SOROC as printer.

CONFIGURATION 10: Smarterm as Console

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card								
AIO comm card			Y					
corvus card								
smarterm card				Y				
videx card								
double vision								
diskii card						Y		
micro-sci card							Y	

Errors detected : None
Error description :
Comments : Used generic CONSOLE.CODE.

CONFIGURATION 11: AIO Comm Card as CONSOLE Driver Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card								
AIO comm card				Y				
corvus card								
smarterm card								
videx card								
double vision								
diskii card						Y		
micro-sci card							Y	

Errors detected: None
Error description:
Comments: Used generic CONSOLE.CODE.

CONFIGURATION 12: AIO Card as CONSOLE Driver Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card								
AIO comm card				Y				
corvus card								
smarterm card		Y						
videx card								
double vision								
diskii card						Y		
micro-sci card							Y	

Errors detected: None
Error description:
Comments: Used generic CONSOLE.CODE.

CONFIGURATION 13: AIO Card as CONSOLE Driver Test

SLOT--> / DEVICE	0	1	2	3	4	5	6	7
language card	Y							
grappler card								
AIO comm card				Y				
corvus card			Y					
smarterm card								
videx card								
double vision								
diskii card						Y		
micro-sci card							Y	

Errors detected: None
Error description:
Comments: generic console driver

APPENDIX D

BOOTING DIFFICULTIES

The p-System, as supplied for the Apple II, is intended to run a basic Apple II hardware configuration, and may not initially drive some hardware configurations. It may be necessary to reconfigure the system to operate in the particular environment defined by your Apple and peripheral cards.

NOTE: The printer driver supplied won't drive the Silentype Printer and the Apple Super Serial Card may not be driven if an Apple Language card is used.

The system will only boot from DISKII drives, or drives which properly emulate DISKII.

If the p-System won't boot on your present hardware configuration, we suggest that you attempt to configure the system as follows:

1. Try to boot the system with ONLY: a) 16K RAM card in slot 0; and b) DISKII drives in slot 6. The system is designed to boot in this configuration and should do so with no problems.

2. The supplied SYSTEM.SBIOS contains general PRINTER, REMOTE, and CONSOLE drivers, as well as a DISKII/MICRO-SCI disk driver. If needed, you should build a new SYSTEM.SBIOS for your system, as described in Chapter III, according to the following:

   slot 1: PRINTER: the supplied printer driver will work with most serial or parallel printer cards. If an 80 column card is used, a specific console driver may be necessary.

   slot 2: REMOTE: the supplied remote driver will work with most serial remote cards or RS-232 cards, including the D.C. HAYES MICROMODEM. If an 80 column card is used, a specific console driver may be necessary.

   slot 3: CONSOLE: the general console driver will work with several of the available 80 column cards, however, if certain printer or remote cards are used it is necessary to replace CONSOLE.CODE with one of the supplied specific drivers when relinking an SBIOS. Cars which have a soft switch will require a SYSTEM.STARTUP program which enables the switch. An example for use with the Videx 80 column card is attached.

slot 4: DISKII: the supplied vectors expect DISKII or MICRO-SCI drive controllers in this slot. (However, a Z-80 card will be ignored.)

slot 5: DISKII: the supplied vectors expect DISKII or MICRO-SCI drive controllers in this slot.

slot 6: DISKII: the supplied vectors expect DISKII or MICRO-SCI drive controllers in this slot. This is the boot slot and it will be necessary, at least initially, to have DISKII controller in this slot.

slot 7: May be used for SVA 8" disk controller (single sided, single density at present), but the system must first be configured by linking in the SVA8INCH and SVAVECT to the SBIOS.

3. Run the utilities CONFIG.CODE and SETUP.CODE and detailed in Chapter V and Chapter IV, respectively.

4. If any of the code files on the CODE: disk are used, it is necessary to transfer SYSTEM.BOOT from the CODE: disk to the boot disk.

5. When you have completed these steps, turnoff the computer, replace the cards you have configured the system for, and reboot the system.

Appendix D

The following source code is designed to activate the Videx Soft Switch on the 80 column card. The code appears to solve the Soft Switch problem.

Program Startup may be implemented in the following manner:

1. Enter in and compile the source code;

2. Save the source and code under some arbitrary name; and

3. Change the <NAME>.CODE file to SYSTEM.STARTUP.

Booting your system will now result in engaging the Soft Switch.

```
PROGRAM.STARTUP;
  (* flip Videx Soft Switch on system startup *)

 PROCEDURE POKE(ADDR,VALUE : INTEGER);

  TYPE MEMCELL = PACKED ARRAY[0..0] OF 0..255;

  VAR  MEMORY : RECORD CASE BOOLEAN OF
                       FALSE : (ADDRESS : INTEGER);
                       TRUE  : (CONTENTS : ^MEMCELL);
                END; (* memory *)

 begin (* poke *)
   memory.address ;= addr;
   memory.contents^[0] := value mod 256;
 end; (* poke *)

begin (* startup *)
  writeln('About to poke videx soft switch on....');
  poke(-16295,0);
  writeln('Soft Switch engaged.');
end.  (* startup *)
```

INDEX

APPLE Hardware Requirements 7
APPLEINFO Table 42
APPLEINTER.CODE 90
APPLE Slot Assignments 155
APPLESTUFF Unit 123
Arranging Disk Files 29
Assembler 20, 30

Backing up Diskettes 11
BASIC 19
Beginner's Guide 16, 31
Binding in the Debugger 35
BIOS.CODE 90
BIOS.C.CODE 90
BIOS.CR.CODE 90
BIOS.CRP.CODE 90
Boot Disks 9, 30
Bootstrapping 9, 116

CLKREAD 64
Clock Driver 78
Compilers 19
COMMCARD.CODE 38, 45
Configuring Disk Files 29
CONFIG Utility 146
Console Driver Example 79
CONINIT 55
CONREAD 55
CONSTAT 55
CONWRIT 56
CONSOLE.CODE 38, 43

DBL.VISION.CODE 49
Debugger 35
Disk Driver 74
Disk Files 29
Disk Formatting 10
DISKII.CODE 38, 45
DISKVECT.CODE 46
Diskvect Configuration 87
Driver Example (Console) 79
Driver Example (Disk) 74
Driver Examples (Other) 77
DRIVERS.CODE 39
DSKINIT 59
DSKREAD 57

DSKSTOP 60
DSKSTRT 59
DSKWRIT 58

Editor 19
EMPTY.CLK.CODE 47
EMPTY.COMM.CODE 38
EMPTY.CON.CODE 46
EMPTY.FIRM.CODE 38
EMPTY.INT.CODE 38, 47
EMPTY.PRL.CODE 38
EMPTY.PRN.CODE 46
EMPTY.REM.CODE 46
EMPTY.SER.CODE 38
EMPTY.USR.CODE 47
ENDMARK.CODE 39

Filer 19
FIRMCARD.CODE 38, 47
FORMATTER.CODE 11

GOTOXY 41, 104, 115

Hardware Requirements 7
HAS_LC_VIDEO 15, 147
HAS_SHIFT_WIRE 15, 147

INTCARD.CODE 38, 48
INTERRUPTS.CODE 39, 48
INTERP.CODE 90
Interpreter Configuration 89

Keyboard Notes 13
KEYBOARD.CODE 50

Linking an Interpreter 89
Linking an SBIOS 39

Memory Configuration 88

Operating System 9, 18

PATCHCARD.CODE 38, 48
Peripheral Drivers 37
Polling Peripherals 87
Primary Bootstrap 117, 119, 120
PRINTER.CODE 38, 43

PRNINIT 60
PRNREAD 61
PRNSTAT 61
PRNWRIT 62
p-System Devices 154

Reconfiguration Summary 37
REMINIT 62
Remote Driver 77
REMREAD 63
REMSTAT 62
REMWRIT 63
REMOTE.CODE 38, 43

Sample Disk Driver 74
Sample Console Driver 79
Sample GOTOXY 107
Sample Pascal Program 25
Sample Primary Bootstrap 119
Sample SYSTEM.MISCINFOs 5
SBIOS.CODE 38, 43
SBIOS Jump Vector 52
SBIOS Outline 68
SBIOS Routines 52
SBIOS Parameter Passing 70
SBIOS Variables 73
SCREENTEST 111
SENABLE 64
SERCARD.CODE 38, 49
SEVENT 64
SETBUFR 57
SETDISK 55
SETSECT 57
SETTRAK 56
SETUP.CODE 93
SMARTERM.CODE 50
SQUIET 64
STARTUP: Diskette 16
SUPRTERM.CODE 49
Supplied Peripheral Drivers 37
SVA8INCH.CODE 51
SVAVECT.CODE 51
SYSHALT 54
SYSINIT 54
System Disk Files 31
SYSTEM.BOOT 121
SYSTEM.LIBRARY 21, 31, 34

SYSTEM.MISCINFO 41, 96, 113
SYSTEM.PASCAL 18, 30, 35
SYSTEM.SYNTAX 19, 29

Terminal Configuration 37, 96, 104, 146
TURTLEGRAPHICS Unit 127

Units 33
User-Defined Devices 64
User Driver 77
USRINIT 65
USRREAD 66
USRSTAT 66
USRWRIT 67
Utilities 21

VIDEX.CODE 50

Writing Your Own Drivers 52
Writing Your Own Bootstrap 116