I. INTRODUCTION

I.1 Preliminaries

The first and most important thing to do when you receive your UCSD p-System is to back up the disks. This step is described below in Section I.3. We suggest that you read this introduction first, then go through the steps that Section I.3 details.

The UCSD p-System is intentionally machine-independent, and portable across a variety of microprocessor systems and peripheral devices. Because there is currently a lack of standard hardware protocols, differences between machines are dealt with by the System's software: most of this "tailoring" of software to hardware was done while the System was developed, and is part of the System as shipped, but in many cases, some further tailoring must be done by the user.

Microprocessors differ in their instruction sets, the way that they address main memory, and the way that they handle Input/Output devices.

The UCSD System deals with different instruction sets by providing an "interpreter" for each processor that is supported. In the System, Pascal and other high-level languages are compiled to a code called "P-code". This P-code is a set of instructions for a virtual machine; each interpreter takes this code and executes it upon a particular processor (often called the "host processor"). Some hardware systems execute P-code directly, and bypass the need for an interpreter.

Differences in addressing between processors ("byte sex" differences) are handled internally by the System, and need only concern the user when transferring data files from one sort of processor to another. See Section I.2.3.

Differences in I/O devices are dealt with by a portion of the System called the BIOS (for Basic I/O Subsystem). The BIOS handles all low-level device control. A portion of the "BIOS called the SBIOS (for Simplified BIOS) is a part of our Adaptable Systems, and may be modified by the user. For some hardware configurations, p-Systems are shipped with a BIOS ready to use, and for other hardware configurations, the user may have to write the SBIOS from scratch. The differences between various p-Systems are described below.

Since the p-System is intended for a single user working in an interactive mode, the System's terminal ('CONSOLE:') is a very important peripheral. Tailoring the System to a particular terminal is easily done: see Section I.2 and Chapter III.

Finally, each installation of a UCSD p-System must have a "bootstrap" program that starts the System running on a particular hardware configuration. As with the I/O-handling routines, a bootstrap may be shipped with your System, or you

may need to write one yourself.

Chapter V contains information about individual processors. For some processors, the System shipped is called the "Adaptable System." This Adaptable System is fully described in Chapter IV.

If your hardware uses a PDP-11 or LSI-11 processor, then the bootstrap and I/O routines are supplied with your p-System. More information about these processors (and about conversions to and from RT-11 format) is given in Chapter V.

If you have a Z80 or 8080 processor that runs the CP/M ® operating system, the p-System may be booted with the aid of CP/M. Initially, the System will use CP/M's CBIOS for its SBIOS. We call this System the CP/M Adaptable System; it is described in Section IV.3. After you have booted your p-System, you may write a simpler and faster bootstrap of your own, and add capabilities to your System by writing SBIOS routines yourself.

If you use a Z80 or 8080 without CP/M, or a 6502 processor, then the bootstrap and SBIOS must be written by you. This System is called the Full Adaptable System, and is described in Section IV.4

Finally, your p-System may be a System tailored to some particular processor, so that you need not do any adaptation (except to tailor screen control). If this is the case, the documentation you receive will include a supplement which describes your particular hardware; you will need to use this Installation Guide only rarely.

I.1.1 Summary

To sum up this introduction, these are the things which you must do when you receive your p-System:

1) Back up the disks. This is extremely important. See Section I.3.

2) If you have an Adaptable System, you must get it bootstrapped, and unpack the disks. Which order you do this in depends on your hardware; see Chapter IV. Section I.4 is an introduction to down-loading, and Chapter II an introduction to bootstrapping.

   A) For CP/M users, use the software provided to bootstrap directly.

   B) For other Adaptable System users, write your own bootstrap and SBIOS.

   C) Test your System to make sure that it works.

3) Provide the System with information for handling your interactive terminal. See Chapter III, which covers SETUP and GOTOXY.

4) Use your new p-System.

I.2 General Information

This section is an overview of the machine-specific details you must attend to when getting started with our System. Bootstrapping is described separately in Chapter II.

I.2.1 SETUP and GOTOXY

This section introduces the two basic mechanisms for controlling the System's console. With proper use of SETUP and GOTOXY, you may use the Screen Oriented Editor, which is much more flexible and easier to use than YALOE (Yet Another Line Oriented Editor). More sophisticated screen control and data capture can be achieved by using the Operating System's Screen Control Unit: this is described in the Internal Architecture Guide.

I.2.1.1 SETUP

When the System is booted, it reads a file called SYSTEM.MISCINFO (see Chapter 1 in the Users' Manual). SYSTEM.MISCINFO contains hardware-related information which the System needs; most of it concerns the System's interactive terminal, 'CONSOLE:'. The console is used extensively by the System, especially the Screen Oriented Editor. SYSTEM.MISCINFO specifies, among other things, the use of the console's keyboard, and its special functions such as cursor-control arrows, Editor accept, and Editor escape.

SETUP is a utility program which allows you to create a new SYSTEM.MISCINFO or modify an existing one. You must use SETUP to specify the characteristics of your interactive terminal before you can use the Screen Oriented Editor. Some MISCINFO files for some popular terminals have been included on the utilities disk. A MISCINFO file may be utilized by C(hanging its name to SYSTEM.MISCINFO in the F(iler. More about the use of SETUP and the MISCINFO files which are provided is given in Section III.2.Installation Guide
The Adaptable System

I.2.1.2 GOTOXY

The Screen Oriented Editor also requires a System intrinsic to position the console's cursor at an arbitrary position on the screen; SETUP cannot provide this. This intrinsic is called GOTOXY, and must be written by the user. Section III.3 tells how to write a GOTOXY procedure.

As with MISCINFO files, some sample GOTOXYs are shipped on the utilities disk, and may correspond to the terminal that you use. More about this is given in Section III.3.

I.2.1.3 Binding GOTOXY

To become a System intrinsic, your GOTOXY procedure must be "bound" into the Operating System. This is accomplished by using LIBRARIAN to replace the GOTOXY that is shipped with the GOTOXY that you have written yourself. See Section III.3 for full details.

I.2.2 The Adaptable System

If your hardware system does not use a PDP-11 or LSi-11 processor, and if your p-System was not pre-packaged for your particular hardware, then the System you receive will be an Adaptable System. The Adaptable Systems require that you do some programming before they can run on your machine. For systems that already use the CP/M operating system, this involves little or no work. Adapting other systems takes much more time and knowledge.

Figure 1 illustrates the I/O structure of the Adaptable System. Under the heading 'UCSD PASCAL I/O HIERARCHY' is a diagram of various portions of the System, and their interrelationships. The hexagon labelled 'screen I/O' represents the portion of the System that must be tailored with SETUP and GOTOXY. Of the machine-dependent portions of software, only the SBIOS need be written or modified by the user: the remaining software is already supplied with your System.

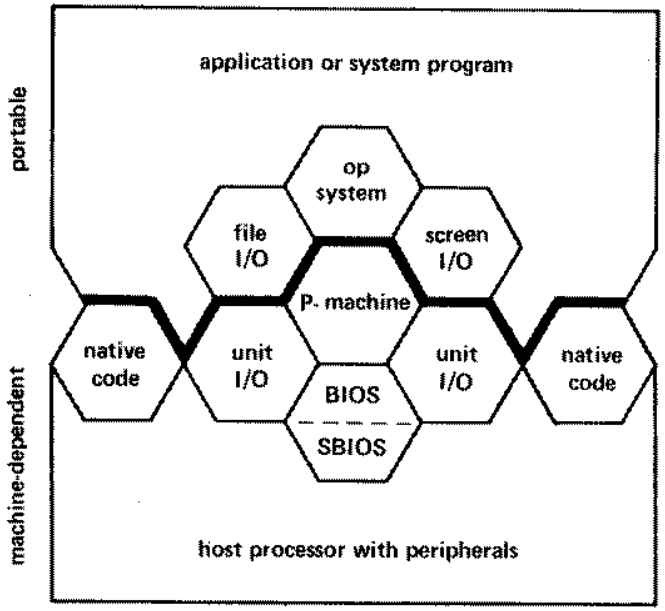
The Adaptable System requires that the user supply a bootstrap (for a CP/M Adaptable System, this is done automatically), and requires a user-supplied SBIOS to be loaded at bootstrap time. The SBIOS routines must be written in the native code of the host processor. This is usually done in assembly language under some other operating system: modifying that operating system's I/O routines to meet the p-System's requirements is often a convenient way of creating your own SBIOS.

Once the SBIOS routines have been written, they must be tested. A test program is provided for this purpose. When the integrity of the SBIOS has been established, the UCSD System may be bootstrapped.

After the System is bootstrapped, enhancements may be made to speed it up, produce a turnkey system, and add additional device drivers.

The Adaptable System is described in Chapter IV. Troubleshooting information is presented in Appendix E.

UCSD PASCAL I/O HIERARCHY



A SAMPLE SYSTEM

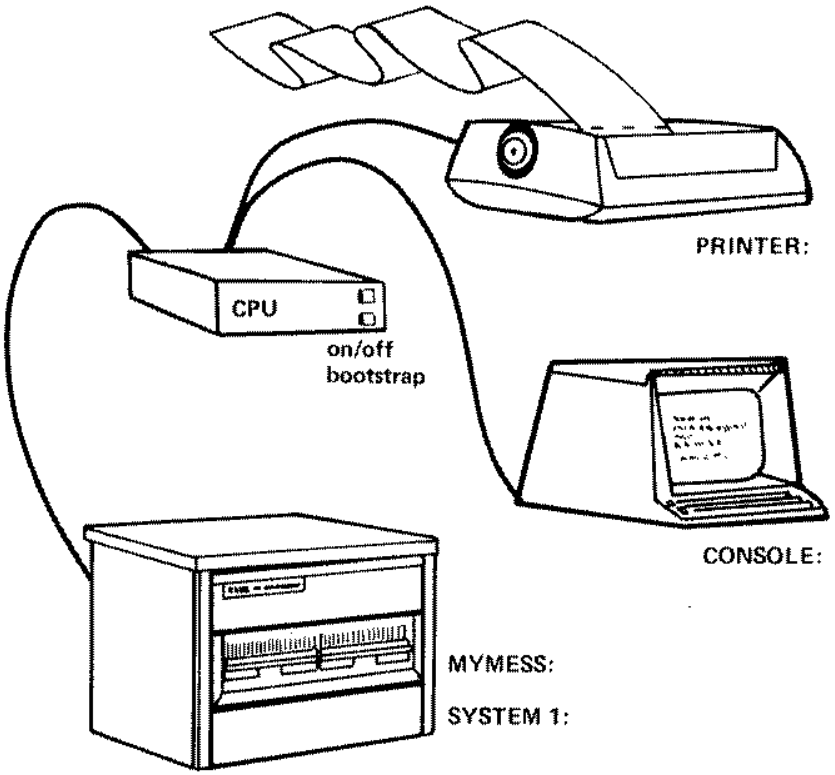


FIGURE 1

I.2.5 Byte Sex Handling

Different processors address words in one of two ways: most-significant-byte-first, or least-significant-byte-first. We use the term "byte sex" to refer to this difference in addressing. Data stored in a file on one machine will be "byte-flipped" when compared to the same data stored on a machine of opposite byte sex.

In general, this presents no problem. The System automatically detects the sex of a P-code codefile or a directory, and treats it appropriately. But the user should be conscious of byte sex when using assembled code (which is appropriate for only a particular machine), and when transferring data files from one machine to a machine of opposite sex. (Because only the user knows the format of data within the file, the user must take care to flip the data correctly; no general-purpose routine can be provided.)

Aside from those two considerations, UCSD System software is portable regardless of byte sex.

I.3 Backing Up Disks

Booting, unpacking, and downloading are dangerous operations that can destroy disks, so it is essential that you make backups of the disks that are shipped to you before doing anything else with (or to) your System.
If the disks you receive have write-protect slots, it is suggested that you write-protect the disks before backing them up. This is a further protection against damage.

I.3.1 Ready-to-use Systems

For PDP-11/LSI-11 Systems and other tailored Systems that come with a bootstrap and are ready to use, you may use the Filer to back up your disks. Follow these steps:

1) Bootstrap your System, then type 'F' for F(iler.

2) Once in the Filer, type 'T' for T(ransfer.

3) Place the disk you are backing up in drive #4
(the drive you booted from), and the blank (or useless)
disk in drive //5 (the alternate disk drive).

4) The Filer will prompt you with:
Transfer what file? #4,#5
... respond with the underlined portion.

5) The Filer then prompts:
Transfer 494 Blocks? Y
... tell it yes, as indicated. (The actual number of blocks may vary.)

6) The Filer will either proceed with the transfer, or prompt you with:
Destroy WHATSIS?
... or whatever the disk in drive #5 is called.

If you want it destroyed, type 'Y', otherwise try backing up onto a different disk.

After much clicking, the transfer will be complete, and you will be ready to back up the next disk. When you are through backing up the disks, you are ready to use your System. The next step will probably be configuring your terminal — see Chapter III.

If your hardware includes only one disk drive, in step (4) you must specify #4,#4. Backing up proceeds in the same way, but more slowly: the Filer will prompt you to swap disks back and forth.

For more information on T(ransfer in the Filer, see the **Users' Manual** Chapter III (Section III.6.3.14).

Warning: on some hardware, doing a T(ransfer does not transfer the bootstrap which is required on the main System disk. To transfer the bootstrap, use the utility BOOTER (described in Section II.3). This is not necessary for PDP-11s and LSI-11s. If it is necessary for your particular hardware, you will be told so in the supplemental brochure that comes with your p-System.

I.5.2 Adaptable Systems

Both CP/M and Full Adaptable Systems are shipped as 8" diskettes in IBM 3740 format: single-density, single-sided, 77 tracks, 26 sectors per track, 128 bytes per sector. To back up the disks, all 77 tracks must be copied to another disk. It may be possible to do this using an existing copy utility under an existing operating system, or you may have to copy disks by writing your own assembly language program under some operating system. This program must read all 26 sectors of a track into memory, and then copy them onto the same location on another disk. This should be done (in a loop) for all 77 tracks.

Once your disks have been backed up, you may proceed with bringing up your System: providing a bootstrap and device drivers, and configuring your terminal.

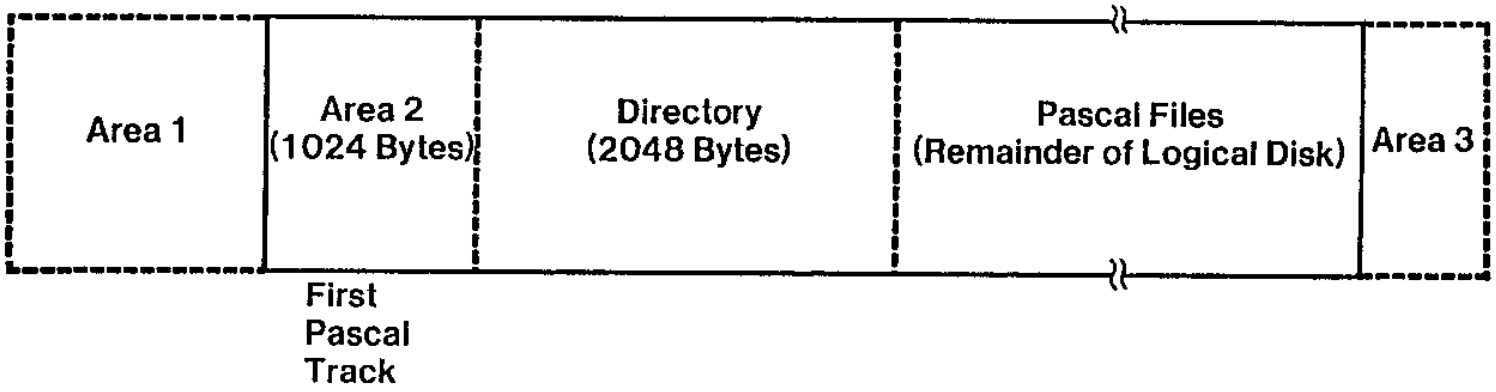
Once the System is up and running, it is possible to back up disks using the T(ransfer command in the Filer (as described in the previous section), or by using the utility DISKCHANGE (described in Section IV.2).

I.4 Downloading and Unpacking

This section is relevant only to users of the Adaptable Systems.
Once the disks have been backed up, the installation of the System can begin. If your disk drives do not support the 8" soft-sectored disks that the System is shipped on, you will need to "download", i.e., transfer the information on the disks that are shipped to your own media. Unpacking is part of the downloading operation.

Figure 2 shows the general format of a UCSD Pascal Disk. This format is the same regardless of the disk's size. Adaptable System disks are partitioned into three small disk images, as shown in Figure 3. Each of these "logical disks" has the same format as a full-sized disk, but only the first one (the "Default Disk" in the illustration) is visible to a UCSD System. The user must unpack these logical disks in order to utilize all of the files on them.

Logical Structure of a Generalized UCSD Pascal Disk



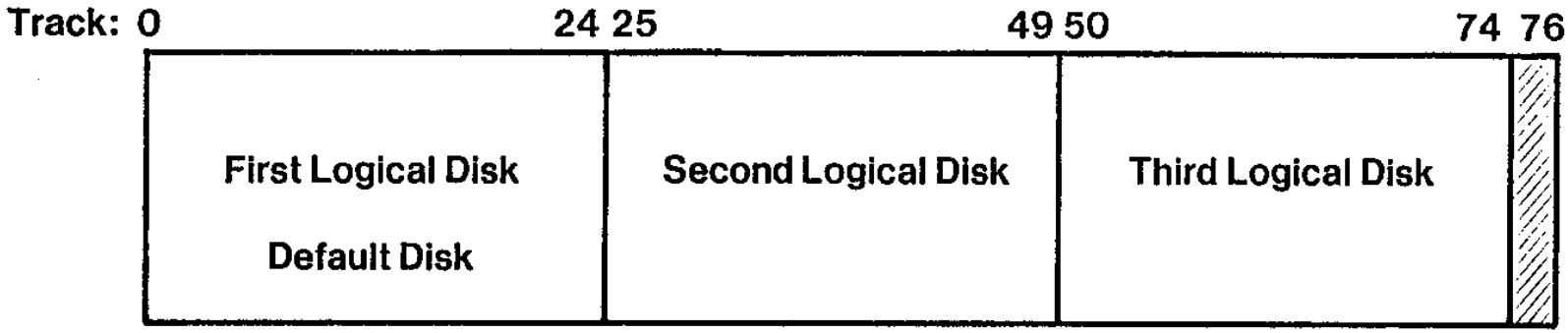
Area 1: Usually Track 0; may be more than one track or non-existent. Useful for bootstrapping and compatibility purposes.

Area 2: May be used for bootstrapping or test purposes.

Area 3: Remainder of Physical Disk; usually non-existent.

FIGURE 2

Physical Structure of an Adaptable System Distribution Disk



- 8 Inch Single Density Diskette
- Partitioned into three "logical disks" of 25 tracks each, intended to fit on mini-floppies
- No sector interleaving
- No track-to-track skew

FIGURE 3

When downloading, each image should be recorded on its own diskette. In this way the disks are unpacked as they are downloaded. One or two of the disks are bootable disks. They contain all of the information necessary to bootstrap the System. One disk is the System disk. It contains the files which are used during the normal operation of the System. The Utilities disk has miscellaneous applications and systems utilities which are used only from time to time. The Orienter/Startup disk has some programs that accompany Bowles' Beginner' Guide to the UCSD Pascal System. A catalog of these disks (describing their format and contents) may be found in Appendix B.

Downloading can be done either on a computer that supports both the source and target disk formats, or through a serial line and two computers.

In the former case, you may have an existing operating system or utility that is capable of copying tracks 0-24 onto one target disk, 25-49 onto another, and 50-74 onto a third.. If no such utility is available, you must write an assembly language program which reads each track, sector by sector, and then writes it to the target disk.

If two computers are involved, the one which supports 8" disks must read data from the source disk and send it out through a serial line; the other machine must be running a program which reads data from the serial line and writes it to the destination disk. The data should be read and written in contiguous areas: sector by sector.

If your hardware system supports 8" disks, and you are capable of booting the p-System off of a default disk (the first of the three disk images), then you may not need to unpack your disks before booting the System. In this case, the DISKCHANGE utility can be used to unpack the disks, once the System is running. This is a good deal easier than other methods. See Chapter IV to determine if this is possible. Documentation for the DISKCHANGE program is in Section IV.2.1.

In other cases, at least the bootstrap disk must be unpacked and downloaded before the System is bootstrapped. Bootstrapping in general is discussed in Chapter II, and bootstrapping Adaptable Systems is discussed in Chapter IV.

Important: These are the basic requirements for downloading disks:

1. Track 0 on the source disk must be transferred to Track 0 on the destination disk. Only the first 18 sectors contain information (2504 bytes).

2. If Track 0 on the destination disk contains less than 2504 bytes, copy Track 0 to Track 0 + Track 1 (do not change the order of the sectors). Track 1 of the source disk must now be transferred to Track 2 of the destination disk.

3. Copy Tracks 1..24 of the source disk to the destination disk. Do not change the order of the sectors or the bytes.

4. If the sectors of the destination disk are not the same size as sectors on the source disk, the information should still be transferred in order, ignoring sector and track boundaries. Exception: Track 0 must still be transferred to Track 0. If Track 0 on the destination disk is longer than 18 sectors, leave the remainder of that track unused.

5. The information which began at Track 1 of the source disk must begin at the start of a track on the destination disk (though not necessarily Track 1). Whichever track on the destination disk contains this information must be indicated in the 'first Pascal track' parameter on the bootstrap stack.

II. INTRODUCTION TO BOOTSTRAPPING

II.1 The Concept of Booting

"Booting" or "bootstrapping" is the problem of starting a software system on hardware which is running either no software at ail, 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. The nature of bootstrapping implies that bootstrap programs are machine-specific -- details about bootstraps for the various kinds of p-System are given below.

II.2 Primary, Secondary, and Tertiary Bootstraps

For the Adaptable System, the bootstrap is divided into three separate parts. This section summarizes the actions of each. Remember that BIOS stands for Basic I/O Subsystem, and SBIOS stands for Simplified BIOS.

The primary bootstrap ...

1. Loads the SBIOS by reading it off the System disk into memory.
2. Loads the secondary bootstrap.
3. Pushes hardware configuration parameters onto the stack.
4. Transfers control to the secondary bootstrap.

The secondary bootstrap ...

1. Initializes the BIOS (which is part of this bootstrap).
2. Reads the System disk's directory into memory.
3. Searches the directory for the Interpreter.
   (Interpreters may be called SYSTEM.INTERP, SYSTEM.PDP-11, SYSTEM.MICRO, etc.)
4. Loads the Interpreter. .
5. On the Z80: restacks the hardware configuration parameters for the benefit of the tertiary bootstrap and the Interpreter.
6. Transfers control to the tertiary bootstrap (which is part of the Interpreter).

The tertiary bootstrap (whose code is linked into the same codefile as the Interpreter) ...

1. Saves the BIOS initialization words (which are on the stack).
2. Initializes some hardware devices and peripherals.
3. Rereads the System disk's directory and 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.

II.3 Machine-Specific Bootstraps

For PDP-11s and LSI-11s, the primary and secondary bootstraps are recorded on blocks 0 and 1 of the System disk. The boot ROM (normally located at 1730D0) reads the first sector (128 bytes) into memory, and this code reads in the rest of the bootstraps. The 11 Interpreter is not 'adaptable', so there are no SBIOS routines or hardware configuration parameters for the user to set up; the Interpreter assumes standard 11 hardware and conventions. A disk of alternate interpreters is provided: different interpreters correspond to different hardware configurations (.i.e., single versus double density floppy drives, RK05 hard disks, etc.). The bootstrap itself discovers the size of main memory. More information on the 11 implementation may be found in Chapter V.

The primary bootstrap for the CP/M Adaptable System is the file PASBOOT on the CP/M-compatible disk. PASBOOT assumes that the CP/M BIOS ("CBIOS") is already in memory. Any customized primary bootstraps which the user may write must first load the CBIOS into memory. The current CP/M Adaptable System will only work with disks that have 128-byte sectors. If the sector length is different, the full Adaptable System must be used. More specific notes on booting the CP/M Adaptable System may be found in Section IV.3, and Chapter V.

All other Adaptable System users must write their own primary bootstrap loader. It must push the proper parameters onto the stack and load the primary bootstrap into memory at either 8000H or D000H. (The primary bootstrap is located on the System disk: track 0, sectors 1 and 2.) The loader must then jump to 8000H or D000H so the primary bootstrap will execute. Care must be taken to use the proper bootstrap (either 8000H or D000H) for the user's particular hardware configuration. Full details about which bootstrap to use are in Section IV.4.1.

The secondary bootstraps for all Adaptable Systems are located on track 0 sectors 5 - 18. The primary bootstrap loads the secondary bootstrap at either 8200H or D200H (depending on the primary bootstrap's location).

Figure 4 indicates the location of primary and secondary bootstraps, the directory, and other files on a System disk of the Adaptable System. This illustration should be compared to figures 2 and 3. System disks for Systems other than the Adaptable System look much the same though they do not include an SBIOS Tester program.

20

Installation Guide
Bootstrapping

Layout of an Adaptable System Logical Disk As Distributed by SofTech Microsystems

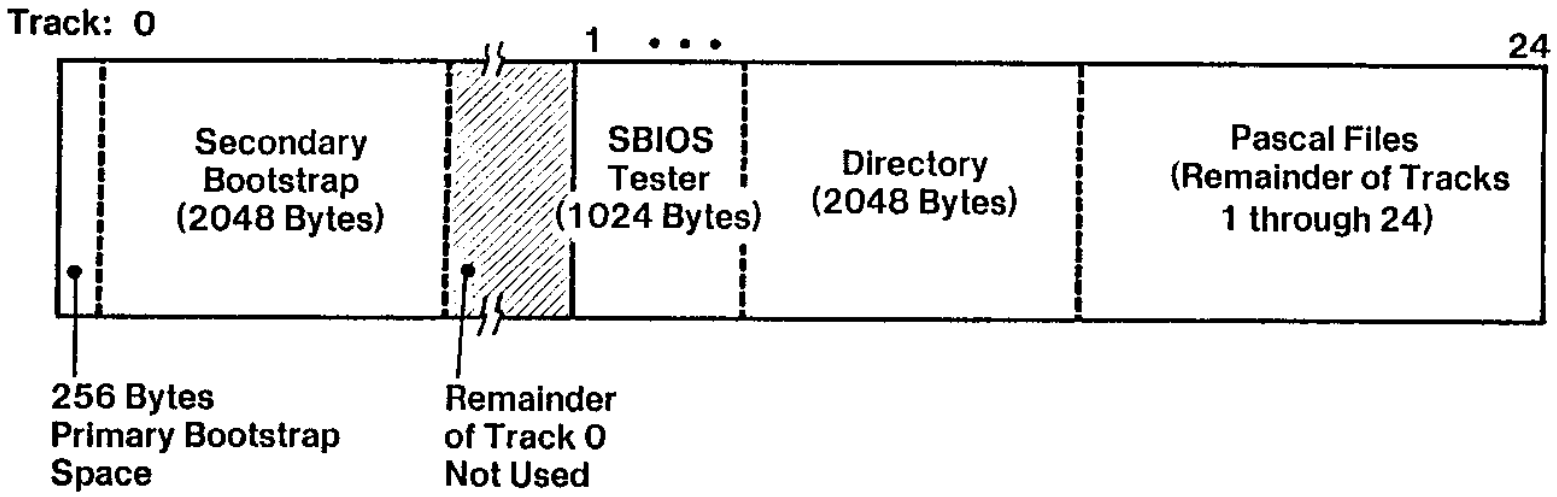


FIGURE 4

II.4 The BOOTER Utility

BOOTER is a utility which transfers a bootstrap from one disk to another. In normal System use, bootstraps are copied only when an entire disk is copied using the Transfer command in the Filer. If you have created a System disk by Transferring individual files to a new disk, BOOTER must be used. On many hardware configurations, Transfer is incapable of copying a bootstrap, and BOOTER must be used in any case (if you have such hardware, you will be told about this situation in the supplemental literature).

The code for BOOTER is on the Utilities disk under the name BOOTER.CODE or ABOOTER.CODE. To copy a bootstrap, eX(ecute the codefile.

On PDP-11, LSI-11, and 9900 systems, ABOOTER prompts for the name of the disk on which the bootstrap will be written, and the name of a file from which the bootstrap is to be read (if only a disk name is given, the first two blocks of that disk will be copied). Only two blocks are transferred: from the input disk or input file to the first two blocks of Track 0 of the output disk.

On Z80, 8080, and 6502 systems, BOOTER prompts for two disk names, and copies all of Track 0 from the input disk to the output disk.

III. TERMINAL HANDLING

III.1 Introduction

You should read this chapter if you are new to the System, want to change or improve the way the System handles your terminal, or want to convert to a new variety of terminal.

The first thing you will be concerned with is SETUP, a utility program that modifies some terminal handling information stored in a file called SYSTEM.MISCINFO. The next thing to tailor is GOTOXY, an intrinsic Pascal UNIT within the Operating System that provides random addressing for your terminal's cursor. The System comes with its own defaults, but for more convenient or more efficient use of your console, you will want to specify your own characteristics.

Changing SYSTEM.MISCINFO with SETUP does not require much knowledge or preparation. Changing the GOTOXY procedure requires a little more familiarity with your terminal, and a knowledge of UCSD Pascal.

To tailor terminal handling to your own needs, you will first run SETUP. SETUP creates a file called NEW.MISCINFO which contains information about your own terminal. You will then go into the Filer, change SYSTEM.MISCINFO to a backup file, and change the name of NEW.MISCINFO to SYSTEM.MISCINFO. After this, you reboot or I(nitialize: the new SYSTEM.MISCINFO is loaded into main memory, and your terminal is now controlled according to the information in this file. To see if you have run SETUP correctly, you might want to run the SCREENTEST diagnostic immediately, or you might want to wait until you have bound in a new GOTOXY. To create your own GOTOXY, you will write a Pascal procedure that does cursor addressing, create a codefile by C(ompiling it, and bind the codefile into the Operating System by using the Librarian utility. After binding, you should reboot, and then test the terminal handling by running SCREENTEST.

SCREENTEST checks that characters are being sent and received properly, and that the Screen Oriented Editor interface will work. If you encounter problems, it is easy to go back into SETUP and change your specifications, or modify your GOTOXY procedure and bind it in again.

If you don't feel confident, you might do a little more reading. Check your own terminal manual, and the following portions of the Users' Manual; the UNITWRITE intrinsic (Section VI.2.36), the introduction to the Screen Oriented Editor (Sections IV.0 and IV.1), and glance over the description of YALOE (Yet Another Line Oriented Editor, described in Chapter V). YALOE can be used on virtually any terminal, but the Screen Oriented Editor, which is more convenient and is usually used as the System editor, requires GOTOXY.

This chapter describes the care and feeding of SETUP, SCREENTEST, and GOTOXY. Users who wish to do more involved screen handling may use the

Operating System's Screen Control Unit, which is described in the Internal Architecture Guide.

III.2 SETUP

SETUP is provided as a System utility (on the Utilities disk) 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 is important to reboot (or I(nitialize) so that the System will start using the new information. It is also important to back up old data, at least until after you have run SCREENTEST, so that you can climb back out of any hole you dig for yourself!

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.

SYSTEM.MISCINFO contains three types of information:

Miscellaneous data about the System,

General information about the terminal, and

Specific information about the terminal's various control keys.

Section III.5.4 (Appendix D) contains a sample session with SETUP. You might look this over before you actually use the program.

III.2.1 Running SETUP

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

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

(The '[Dl]' is the SETUP version number, and may be different for your particular System.)

To invoke any command, just type its initial letter.

H(ELP gives you a description of the commands that are visible on any promptline 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 prompted 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. You will find that your particular terminal probably requires more sophisticated specifications.

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 System promptline.

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

Section III.2.2 contains a detailed description of the data items in SYSTEM.MISCINFO. An abbreviated list of all the data items, together with the System-supplied defaults, is in Section III.5, along with a list of sample settings for a variety of terminals (Appendices A and B for this chapter).

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

Once you have run SETUP, you should always backup SYSTEM.MISCINFO under some other name (OLD.MISCINFO is one suggestion; you might want to name your backups according to different terminals, e.g., TTY.MISCINFO, IQ120.MISCINFO, VT52.MISCINFO, etc.), then change the name of NEW.MISCINFO to SYSTEM.MISCINFO and reboot or I(nitialize. It is indeed possible to update to memory alone, and go on using the System without rebooting, but the results may not always be what you wanted, and the backup security is more risky. In general, M(EMORY) UPDATE is a Q(UIT option that you will use only when experimenting. If you do get into a bind, remember that the current in-memory SYSTEM.MISCINFO can be saved by running SETUP and doing a D(ISK) UPDATE before you change any data items.

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).

The only thing SETUP will not arrange for you, as far as terminal handling goes, is telling the System how to do random addressing for your terminal's cursor. This is a feature that the Screen Oriented Editor requires. To learn how to support this capability, see the section on GOTOXY.

III.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.

On the PDP-11, LSI-11, 8080, 9900, 6502, 6809, and Z-80 systems HAS WORD ORIENTED MACHINE is always FALSE.

HAS BYTE FLIPPED MACHINE is FALSE for all IV.0 systems except the 9900.

SETUP and the Manual refer to PREFIXED [DELETE CHARACTER]. This refers to the backspace function: read it as PREFIXED [BACKSPACE]. On most terminals it will be FALSE.

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 does not have, set the item to NUL (zero).

If you have a DEC VT-52 and a backspace won't move the cursor on the console, this is because you have KEY TO DELETE CHARACTER set to '_', the "rubout character". This is a printing character, so the Operating System does not echo a cursor move; the contents of memory are updated correctly. One workaround is to use the V(erify key to display the actual file contents, but to fix this for good use SETUP to change KEY TO DELETE CHARACTER to control-H or left-arrow - BACKSPACE should be set to the same character as well.

III.2.5 The Data Items in 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. Default values are shown, and sometimes our recommendations. When no suggested values are given, you should consult your own terminal's documentation. The items are ordered according to setup's menu. (See Section III.5.1, Appendix A.)

If you are using a hardcopy terminal or a storage screen rather than a CRT, you can ignore all the data items that are only used by the Screen Oriented Editor and leave them set to their defaults. In particular, if you are in doubt about a particular item, it is safest to leave it set to NUL. Always leave items set to NUL which concern features that your terminal does not have (ERASE LINE, for instance); the software will take care of these situations.

Please note that 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 System; on some terminals, the same function may be performed by two different characters. On other terminals, the key pressed and the character sent for a given function may be the same, but in any case, when you run SETUP you must be explicit and answer all questions, even if the information is redundant.

There are a few characters which you cannot change with SETUP. These are CARRIAGE RETURN «return>), LINE FEED (<lf>), ASCII DLE (control-P), and TAB (control-I). It is assumed that <return>, <lf>, and TAB are consistent on all terminals. ASCII DLE (data link escape) is used as a blank compression character. When sent to an output textfile, it is always followed by a byte containing the number of blanks which the output device must insert. If you try to use control-P for any other function, you will run into trouble. More information on DLE is given in the sections below on GOTOXY and SCREENTEST.

BACKSPACE

When sent to the screen, this character should move the cursor one space to the left. Default: ASCII BS.

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 NUL. Suggested: ASCII ETX (control-C or "Home").

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 (control-[).

EDITOR EXCHANGE-DELETE KEY

This key is also used by the Screen Oriented Editor. It operates only while doing an eX(change, and deletes a single character. Default: ASCII US (control-_).

EDITOR EXCHANGE-INSERT KEY

Like the EDITOR EXCHANGE-DELETE KEY, this only operates while doing an eX(change in the Screen Oriented Editor: it inserts a single space. Default: ASCII RS (control-^).

ERASE LINE

When sent to the screen, this character erases all the characters on the line that the cursor is on. Default; ASCII NUL.

ERASE SCREEN

When sent to the screen, this character erases the entire screen. Default: ASCII NUL.

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: ASCII NUL.

ERASE TO END OE 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: ASCII NUL.

HAS 8510A

May be TRUE or FALSE. Should be TRUE if and only if your hardware system is a Terak 8510a. Default: FALSE.

HAS BYTE FLIPPED MACHINE

May be TRUE or FALSE. On PDP-11, LSI-11, 8080, Z-80, and 6502 processors this bit is FALSE. On the 6800, 9900, and the GA440 system, it is TRUE. In general, it is TRUE only for implementations in which the IPC (Instruction Program Counter) is segment-relative. Default: FALSE.

HAS CLOCK

May be TRUE or FALSE. If your hardware has a line frequency (60 Hz) clock module, such as the DEC KW11, setting this bit TRUE will allow the Pascal system to optimize disk directory updates. It also allows you to use the TIME intrinsic: see Section VI.2 in the Users' Manual. If your hardware doesn't have a clock this must be FALSE. (Adaptable System users must write their own clock-handler; until it is installed, this item must be FALSE.) Default: FALSE.

HAS LOWER CASE

May be TRUE or FALSE. It should be TRUE if you do have lower case and want to use it. If you seem stuck in upper case even if this bit is TRUE, remember there is a soft alpha-lock: see KEY TO ALPHA LOCK. Default: FALSE.

HAS RANDOM CURSOR ADDRESSING

May be TRUE or FALSE. If your terminal is not a CRT, this should be FALSE. Default: FALSE.

HAS SLOW TERMINAL

May be TRUE or FALSE. When this bit is TRUE, the system's promptlines and messages are abbreviated. It is suggested that you leave this set at FALSE unless your terminal runs at 600 baud or slower. Default: FALSE.

HAS WORD ORIENTED MACHINE

May be TRUE or FALSE. If sequential addresses on your processor reference sequential 16 bit words, this should be TRUE. For PDP-11, LSI-11, 8080, Z-80, 9900, 6800, and 6502 systems, this should be FALSE. For the GA440 system it should be TRUE. Default: FALSE.

KEY FOR BREAK

When this key is pressed while a program is running, the program will terminate immediately with a runtime error. Default: ASCII NUL. Suggested: a key that is difficult to hit accidentally.

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 (control-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 DC5 (control-S).

KEY TO ALPHA LOCK

This character, when sent to the screen, locks the keyboard in upper case (alpha mode). It is usually a key on the keyboard as well. Default: ASCII DC2 (control-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 (control-H or "Backspace").

KEY TO DELETE LINE

Deletes the line that the cursor is currently on. Default and suggested: ASCII DEL ("Rubout").

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 (control-C or "Home").

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. If your keyboard has a vector pad, we suggest using those keys for these functions. If you have no vector pad, you might select four keys in the same pattern (such as, for example, '.', 'K',';', and '0', in that order) and use them as your vector keys, prefixing them or using the corresponding ASCII control codes. Default (in order): ASCII LF, ASCII BS, ASCII FS, ASCII US.

LEAD IN FROM KEYBOARD

On some terminals, 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 PREFlXED[<itemname>] to TRUE. An example of this is in Appendix B below. Default: ASCII NUL.

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 PREFlXED["<itemname>] to TRUE. An example of this is in Appendix B below. Default: ASCII NUL.

MOVE CURSOR HOME

When sent to the terminal, moves the cursor to the upper left hand corner of the screen (position (0,0)). If your terminal doesn't have a character which does this, this data item must be set to CARRIAGE RETURN; you will not be able to use the Screen Oriented Editor. Default: ASCII CR ("Return").

MOVE CURSOR RIGHT

When sent to the terminal, moves the cursor nondestructively one space to the right. If your terminal doesn't have this function, you will not be able to use the Screen Oriented Editor. Default: '!'.

MOVE CURSOR UP

When sent to the terminal, moves the cursor vertically up one line. If your terminal doesn't have this function, you won't be able to use the Screen Oriented Editor. Default: ASCII NUL.

NON PRINTING CHARACTER

The character that will be displayed on the screen when a non-printing character is typed 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 PREFlXED[<itemname>] to TRUE. See the explanations for LEAD IN FROM KEYBOARD and LEAD IN TO SCREEN. An example of the use of two-character sequences is given in Appendix B.

SCREEN HEIGHT

The number of lines in your display screen, starting from 1. If you are using a hardcopy terminal, this should be set to 0. Default: 24 (base ten).

SCREEN WIDTH

The number of characters in one line on your display, starting from 1. Default: 80 (base ten).

STUDENT

May be TRUE or FALSE. On IV.0 Systems, should always be FALSE. Default: 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: 5 (base ten).

III.5 GOTOXY

When you have tailored SYSTEM. MISCINFO with SETUP, you should write your own GOTOXY. GOTOXY is a Pascal UNIT embedded in the Operating System. It provides random addressing for your terminal's cursor. There is a GOTOXY that is provided with the System we ship, (the source for this code, along with other examples, is in Appendix C below), but as it is a general routine for any terminal, it is not fast. When you create your own GOTOXY, you will write a Pascal procedure, compile it, then bind it into the Operating System using the utility LIBRARY.

If you are not yet ready to write your own GOTOXY, you should skip down to the next section, which describes SCREENTEST.

If you intend to do all your work on a line-oriented terminal, you never need to write a GOTOXY.

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. In Section III.5.5 (Appendix C) of this Installation Guide are a few sample versions of GOTOXY, including the source for the GOTOXY code which comes with the System, and the SAMPLEGOTO.TEXT that is also on your System disk. You should look this appendix over.

III.3.1 Writing Your Own GOTOXY

III.3.1.1 A Discussion

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 co-ordinates (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 does not look like one of the following lines:

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

... your GOTOXY will not compile. In particular, there must be no spaces within the comment, and the 'U' must be capitalized.

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 cannot 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.

Please note; some terminals take a bias on X and Y. That is, 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 (control-P) character, which is always interpreted as a blank-compression character. You don't want to send this value as a cursor control character. See the section below on SCREENTEST.

The following section contains a more detailed description of GOTOXY. Section III.5.3 (Appendix C) contains specific examples for a variety of terminals.

III.5.1.2 A Recipe for GOTOXY

This section walks you through a sample GOTOXY, and demonstrates the best way of writing a GOTOXY. To see some more specific examples, see Appendix C (Section III.5.3).

The sample program here is commented like a Pascal program.

```
{$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 [O..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+OEFSET;

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

END {UNIT GOTOXY}.
```

III.2 Binding GOTOXY

The first thing to do, once you have written your own GOTOXY, is to compile it to a codefile. Any filename will do, provided its suffix is .CODE. Choose a name you will remember.

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

You should also make sure that the STUDENT bit in SYSTEM.MISCINFO is set to FALSE -- otherwise GOTOXY binding will not work, and you will get the message "No proc in seg table" when you try to reboot the System.

III.2.1 Using LIBRARY to Bind GOTOXY

First, back up 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.

The LIBRARY is a utility program which is shipped on the Utilities disk under the name LIBRARY.CODE. To run it, eX(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 70 to 100 blocks long. If there is not 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 shouid 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).

Type 'T', The INTERFACE part of your unit will not be copied.

Type '0'. LIBRARY prompts:

Copy from slot O?

... type a space. LIBRARY prompts:

Copy to which slot? 16

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

Now type 'N' for N(ew. This causes a repeat of the prompt;

Input file? SYSTEM.PASCAL

... type in the name of your Operating System, as shown. This is the new input file

Finally, type '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 will not be destroyed because it is already there.

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

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

Input file: SYSTEM.PASCAL
  0 u KERNEL    1481     9 u SCREENOP   918     18 u DEBUGGER   187
  1 s PRINTERR   695    10 s SEGSCINI   416     19 s EXTRALEX  4872
  2 s INITIALI  1358    11 u SOFTOPS    559     20 u SYSCOMD    119
  3 s GETCMD    2779    12 u OSUTIL     511
  4 u HEAPOPS    314    13 u REALOPS    752
  5 u EXTRAHEA   736    14 u CONOJRRE   140
  6 u EXTRAI0    772    15 s USERPROG  1549
  7 u PASCALI0   304    16 u FILEOPS   2146
  8 u STRINGOP   259    17 u GOTOXY      31

Output file: NEWSYS.CODE
  0                      9                      18
  1                     10                      19
  2                     11                      20
  3                     12
  4                     13
  5                     14
  6                     15
  7                     16 u GOTOXY      29
  8                     17
```

... note that 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.

Typing '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, never move slot 0 or slot 15

III.2.2 Problems

If your newly created System will not bootstrap at all, it may be because you moved the USERPROG segment when you used LIBRARY. USERPROG must 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, 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. SCREENTEST is described immediately below.

III.4 SCREENTEST

Now that you have changed your SYSTEM.MISCINFO with SETUP (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.

III.4.1 Running SCREENTEST

Type X for eXecute, 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 upper or lower case; all other characters are ignored), and will then prompt you for the name of an error log file.

If you hit <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 hardcopy 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. A sample log is in Section III.5.5 (Appendix E).

III.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. If you find yourself at a loss and decide to consult Pascal Support, you will need to refer to this log.

III.5.2.1 Problems that can be Fixed by Changing SETUP

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 does not 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 OE 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 Pascal 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;

III.5.2.2 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;
```

III.5.2.3 Other Problems

test_basic:

not all characters written out
-> there is a problem with the Pascal system intrinsic UNITWRITE, or, if you are using the Adaptable System, with the SBIOS. You should call Pascal Support; 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; this is the case with Terak systems; DLE is an ASCII character used as a blank-compression code to save space in output strings;

III.5.3 Miscellaneous Notes on SCREENTEST Problems

The System interprets an ASCII DLE or chr(16) (base ten) within a textfile 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+52 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.

ERASE LINE will have difficulty if there are bugs in the screen emulator for memory-mapped screens. This is applicable primarily to Terak systems. In particular, Teraks have trouble with blank-compression sequences (DLE-expansions) of 64 or longer.

Some terminals will not scroll at all, or scroll two lines at a time. The IV.0 System's Screen Oriented Editor unfortunately cannot handle these terminals -- you must use YALOE for SYSTEM.EDITOR.

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

III.5 Appendix A — SETUP Menu and Defaults

In the defaults shown below, 'T' means true and 'F' means false as per the input conventions in SETUP. The numbers shown are in base ten, literal characters are quoted, and ASCII abbreviations are used for nonprinting characters. When you use SETUP, these values are shown in several formats, so the meaning is clear. {Note: must add the eX(change INSERT CHAR and DELETE CHAR items.}

BACKSPACE	BS
EDITOR ACCEPT KEY	NUL
EDITOR ESCAPE KEY	ESC
EDITOR EXCHANGE-DELETE KEY	US
EDITOR EXCHANGE-ACCEPT KEY	RS
ERASE LINE	NUL
ERASE SCREEN	NUL
ERASE TO END OF LINE	NUL
ERASE TO END OF SCREEN	NUL
HAS 8510A	F
HAS BYTE FLIPPED MACHINE	F
HAS CLOCK	F
HAS LOWER CASE	F
HAS RANDOM-CURSOR ADDRESSING	F
HAS SLOW TERMINAL	F
HAS WORD ORIENTED MACHINE	F
KEY FOR BREAK	NUL
KEY FOR FLUSH	ACK
KEY FOR STOP	DC3
KEY TO ALPHA LOCK	DC2
KEY TO DELETE CHARACTER	BS
KEY TO DELETE LINE	DEL
KEY TO END FILE	ETX
KEY TO MOVE CURSOR DOWN	LF
KEY TO MOVE CURSOR LEFT	BS
KEY TO MOVE CURSOR RIGHT	FS
KEY TO MOVE CURSOR UP	US
LEAD IN FROM KEYBOARD	NUL
LEAD IN TO SCREEN	NUL
MOVE CURSOR HOME	CR
MOVE CURSOR RIGHT	'!'
MOVE CURSOR UP	NUL
NON PRINTING CHARACTER	'?'
PREFIXED [DELETE CHARACTER]	F
PREFIXED [EDITOR ACCEPT KEY]	F
PREFIXED [EDITOR ESCAPE KEY]	F

```
PREFIXED [ED EXCH-DELETE KEY]          F
PREFIXED [ED EXCH-ACCEPT KEY]          F
PREFIXED [ERASE LINE]                  F
PREFIXED [ERASE SCREEN]                F
PREFIXED [ERASE TO END OF LINE]        F
PREFIXED [ERASE TO END OF SCREEN]      F
PREFIXED [KEY TO DELETE CHARACTER]     F
PREFIXED [KEY TO DELETE LINE]          F
PREFIXED [KEY TO MOVE CURSOR DOWN]     F
PREFIXED [KEY TO MOVE CURSOR LEFT]     F
PREFIXED [KEY TO MOVE CURSOR RIGHT]    F
PREFIXED [KEY TO MOVE CURSOR UP]       F
PREFIXED [MOVE CURSOR HOME]            F
PREFIXED [MOVE CURSOR RIGHT]           F
PREFIXED [MOVE CURSOR UP]              F
PREFIXED [NON PRINTING CHARACTER]      F
SCREEN HEIGHT                          24
SCREEN WIDTH                           80
STUDENT                                F
VERTICAL MOVE DELAY                    5
```

III.5.2 Appendix B — Sample setups for Some Terminals

Here is a list of SYSTEM.MISCINFO data items followed by some sample values for four popular terminals. Some items in the SETUP menu haven't been included; these are data items that refer to your 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:	LSI ADM-3A	HAZELTINE 1500/1510	SOROC IQ120	HEATH H19
Data Items :				
BACKSPACE	left-arrow	backspace	ctrl-H	ctrl-H
EDITOR ACCEPT KEY	ctrl-C	ctrl-C	home	ctrl-C
EDITOR ESCAPE KEY	esc	esc	esc	ctrl-[
ERASE LINE	NUL	NUL	NUL	I
ERASE SCREEN	ctrl-Z	ctrl-\	'*'	E
ERASE TO END OF LINE	NUL	ctrl-0	T	K
ERASE TO END OF SCRN	NUL	ctrl-X	Y	J
HAS LOWER CASE	TRUE	TRUE	TRUE	TRUE
HAS RAND CURS ADDR	TRUE	TRUE	TRUE	TRUE
HAS SLOW TERMINAL	FALSE	FALSE	FALSE	FALSE
KEY FOR BREAK	ctrl-B *	break **	break	break
KEY FOR FLUSH	ctrl-F	ctrl-F	ctrl-F	ctrl-F
KEY FOR STOP	ctrl-S	ctrl-S	ctrl-S	ctrl-S
KEY TO ALPHA LOCK	ctrl-R	NUL	ctrl-R	ctrl-R
KEY TO DELETE CHAR	ctrl-H	backspace	l-arrow	ctrl-H
KEY TO DELETE LINE	rubout	shift-DEL	rubout	DEL
KEY TO END FILE	ctrl-C	ctrl-C	ctrl-C	ctrl-C
KEY TO MV CURS DOMM	ctrl-J	ctrl-K	d-arrow	B
KEY TO MV CURS LEFT	ctrl-H	backspace	l-arrow	D
KEY TO MV CURS RGHT	ctrl-L	ctrl-P	r-arrow	C
KEY TO MV CURS UP	ctrl-K	ctrl-L	u-arrow	A
LEAD IN FRCM KEYBD	NUL	NUL	NUL	esc
LEAD IN TO SCREEN	NUL	~	esc	esc
MOVE CURSOR HOME	Ctrl-^	ctrl-R	Ctrl-^	H
MOVE CURSOR RIGHT	ctrl-L	ctrl-P	r-arrow	C
MOVE CURSOR UP	ctrl-K	ctrl-L	u-arrow	A
NON PRINTING CHAR	'?'	'?'	'?'	'?'
PREF [DELETE CHAR]	FALSE	FALSE	FALSE	FALSE
PREF [ED ACCEPT KEY]	FALSE	FALSE	FALSE	FALSE
PREF [ED ESCAPE KEY]	FALSE	FALSE	FALSE	TRUE

PREF [ERASE LINE]	FALSE	FALSE	FALSE	TRUE
PREF [ERASE SCREEN]	FALSE	TRUE	TRUE	TRUE
PREF [ERASE TO EOLN]	FALSE	TRUE	TRUE	TRUE
PREF [ERSE TO EOSCN]	FALSE	TRUE	TRUE	TRUE
PREF [KEY DEL CHAR]	FALSE	FALSE	FALSE	FALSE
PREF [KEY DEL LINE]	FALSE	FALSE	FALSE	FALSE
PREF [KEY MV CRS DN]	FALSE	FALSE	FALSE	TRUE
PREF [KEY MV CRS LT]	FALSE	FALSE	FALSE	TRUE
PREF [KEY MV CRS RT]	FALSE	FALSE	FALSE	TRUE
PREF [KEY MV CRS UP]	FALSE	FALSE	FALSE	TRUE
PREF [MOVE CRS HOME]	FALSE	TRUE	FALSE	TRUE
PREF [MOVE CURS RT]	FALSE	FALSE	FALSE	TRUE
PREF [MOVE CURS UP]	FALSE	FALSE	FALSE	TRUE
PREF [NONPRINT CHAR]	FALSE	FALSE	FALSE	FALSE
SCREEN HEIGHT	24	24	24	24
SCREEN WIDTH	80	80	80	80
STUDENT	FALSE	FALSE	FALSE	FALSE
VERTICAL MOVE DELAY	5	5	10	10

* The BREAK key can also be used, but it's perilously close to RETURN.

** Break is also control-(a) on Hazeltines.

Terminals:	DEC VT-52	HEWLETT/ PACKARD	DATA- MEDIA
Data items :			
BACKSPACE	backspace	backspace	backspace
EDITOR ACCEPT KEY	ctrl-C	ctrl-C	cntrl-C
EDITOR ESCAPE KEY	esc	esc	esc
ERASE LINE	Ctrl-@	cntrl-@	Ctrl-@
ERASE SCREEN	Ctrl-@	cntrl-@	ctrl-L
ERASE TO END OF LINE	K	K	Ctrl-]
ERASE TO END OF SCRN	J	J	ctrl-K
HAS LOWER CASE	TRUE	TRUE	TRUE
HAS RAND CURS ADDR	TRUE	TRUE	TRUE
HAS SLCW TERMINAL	FALSE	FALSE	FALSE
KEY FOR BREAK	Ctrl-@	cntrl-@	cntrl-@
KEY FOR FLUSH	ctrl-F	ctrl-F	ctrl-F
KEY FOR STOP	ctrl-S	ctrl-S	ctrl-S
KEY TO ALPHA LOCK	ctrl-R	ctrl-R	ctrl-R
KEY TO DELETE CHAR	ctrl-H	backspace	backspace
KEY TO DELETE LINE	del	del	del
KEY TO END FILE	ctrl-C	ctrl-C	ctrl-C
KEY TO MV CURS DOMM	B	d-arrow	d-arrow
KEY TO MV CURS LEFT	D	l-arrow	l-arrow
KEY TO MV CURS RGHT	C	r-arrow	r-arrow
KEY TO MV CURS UP	A	u-arrow	u-arrow
LEAD IN FRCM KEYBD	esc	cntrl-A	ctrl-@
LEAD IN TO SCREEN	esc	esc	Ctrl-@
MOVE CURSOR HCME	H	H	ctrl-Y
MOVE CURSOR RIGHT	C	C	ctrl-\
MOVE CURSOR UP	A	A	ctrl-_
NON PRINTING CHAR	'?'	'?'	'?'
PREF [DELETE CHAR]	FALSE	FALSE	FALSE
PREF [ED ACCEPT KEY]	FALSE	FALSE	FALSE
PREF [ED ESCAPE KEY]	TRUE	FALSE	FALSE
PREF [ERASE LINE]	FALSE	FALSE	FALSE
PREF [ERASE SCREEN]	FALSE	FALSE	FALSE
PREF [ERASE TO EOLN]	TRUE	TRUE	FALSE
PREF [ERSE TO EOSCN]	TRUE	TRUE	FALSE
PREF [KEY DEL CHAR]	FALSE	FALSE	FALSE
PREF [KEY DEL LINE]	FALSE	FALSE	FALSE
PREF [KEY MV CRS DN]	TRUE	FALSE	FALSE
PREF [KEY MV CRS LT]	TRUE	FALSE	FALSE
PREF [KEY MV CRS RT]	TRUE	FALSE	FALSE
PREF [KEY MV CRS UP]	TRUE	FALSE	FALSE
PREF [MOVE CRS HCME]	TRUE	TRUE	FALSE

PREF [MOVE CURS RT]	TRUE	TRUE	FALSE
PREF [MOVE CURS UP]	TRUE	TRUE	FALSE
PREF [NONPRINT CHAR]	FALSE	FALSE	FALSE
SCREEN HEIGHT	24	24	24
SCREEN WIDTH	80	80	80
STUDENT	FALSE	FALSE	FALSE
VERTICAL MOVE DELAY	0	0	0

III.5.5 Appendix C — GOTOXY Source Examples

The following example is shipped on your System disk as SAMPLEGOTO.TEXT. It is about as simple a GOTOXY as can be written. It is not the code which is shipped in your Operating System: that is the next example, which on one hand is a much more general program, and on the other hand is also much longer. Since GOTOXY is a frequently used I/O routine, you want it to be efficient: it should be tailored to your particular terminal. This brief example works for a DEC VT-52. For an efficient example, see the Datamedia sample.

```
(*The following is a sample gotoxy procedure for the VT-52*)
(*$U-*)
UNIT GOTOXY;

INTERFACE

PROCEDURE AGOTOXY(X,Y:INTEGER);

IMPLEMENTATION

PROCEDURE AGOTOXY;

BEGIN
  IF X<0 THEN X:=0;
  IF X>79 THEN X:=79;
  IF Y<0 THEN Y:=0;
  IF Y>23 THEN Y:=23;
  WRITE (CHR(27),'Y',CHR(Y+32),CHR(X+32));
END;


END.
```

This example works for a DEC VT-50. It uses writes embedded in WHILE loops, and is not fast.

```
{$U-}
UNIT GOTOXY;

INTERFACE

PROCEDURE AGOTOXY(X,Y: INTEGER);

IMPLEMENTATION
PROCEDURE AGOTOXY;

BEGIN
{Check the input data to see that it is within the screen
dimensions. On some smarter terminals, if a cursor position
command is sent for a position that does not exist, the
results are unpredictable.}
  IF X < 0 THEN X := 0
  ELSE
    IF X > 79 THEN X := 79;
  IF Y < 0 THEN Y := 0
  ELSE
    IF Y > 11 THEN Y := 11;
  {For a DECscope VT-50, GOTOXY needs to be implemented by:}

  {Send the cursor home, 0,0}
  WRITE(CHR(27),'H');

  {While TAB is meaningful, use it to move the cursor}
  WHILE X > 8 DO
    BEGIN
    WRITE(CHR(9));
    X := X-8;
    END;

  {Finish off what portion of the x coordinate could not be
  absorbed with the TAB characters.}
  WHILE X > 0 DO
    BEGIN
    WRITE(CHR(27),'C');
    X := X-1
    END;

  {Send line-feeds to access the y coordinate.}
WHILE Y > 0 DO
```

```
    BEGIN
    WRITE(CHR(10));
    Y := Y-1
    END
  END;


END.
```

This example is for a Datamedia 1520, and demonstrates the quickest form of GOTOXY: using a UNITWRITE to send one single command stream to the terminal. As mentioned above, this method can speed up your terminal I/O by as much as 10%; we recommend it.

```
{$U-}

UNIT GOTOXY;

INTERFACE

PROCEDURE AGOTOXY(X,Y: INTEGER);

IMPLEMENTATION

PROCEDURE AGOTOXY;
VAR
   T; PACKED ARRAY[0..2] OF CHAR;
BEGIN
  T[0] := CHR(30); {chr(30) is an ASCII RS, which is Datamedia's
                    absolute cursor address flag.}
  {Set appropriate character for x coordinate.}
  IF X < ^ THEN T[1] := CHR(32)      {Note the offset of 32.}
  ELSE
    IF X > 79 THEN T[1] := CHR(32+79)
    ELSE
      T[1] := CHR(X+32);

  {Set appropriate character for y coordinate.}
  IF Y < 0 THEN T[2] := CHR(32)
  ELSE
    IF Y > 23 THEN T[2] := CHR(32+23)
    ELSE
      T[2] := CHR(Y+32);

  {Send the cursor where it belongs.}
  UNITWRITE(1,T,3)      {1 is the device number of CONSOLE:}
  END;

END.
```

Here are two more examples using UNITWRITE. They are for a Soroc and a Hazeltine terminal, respectively.

```
(*$U-*)

UNIT GOTOXY;

INTERFACE

PROCEDURE AGOTOXY(X,Y: INTEGER);

IMPLEMENTATION

PROCEDURE AGOTOXY;

(* FOR A 50ROC IQ 120 *)

VAR TELL: PACKED ARRAY [0..3] OF 0..255;

BEGIN
  IF X>79 THEN X:=79
    ELSE IF X<0 THEN X:=0;
  IF Y>25 THEN Y:=25
    ELSE IF Y<0 THEN Y:=0;
  TELL[0] := 27;              (* LEAD-IN FOR SOROCS *)
  TELL[1] := ORD('=');
  TELL[2] := 32+Y;            (* NOTE THE OFFSET *)
  TELL[5] := 32+X;
  UNITWRITE(1,TELL,4)
END;

END.
```

```
{$U-}

Unit gotoxy;

Interface

Procedure agotoxy(x,y: integer);

Implementation

Procedure agotoxy;

{gotoxy for the Hazeltine 1500 and 1510}

var tell: packed array [0..3] 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;
  tell[0] := 126;             {the lead-in for a Hazeltine}
  tell[1] := 17;              {also a DC1}
  if x<30 then
    tell[2] := x+96         {different offset for these terminals}
  else
    tell[2] := x;
  tell[3] := y+96;
  unitwrite(1,tell,4)
End;


End.
```

III.5.4 Appendix D — Sample SETUP Session with Comments

The following is a sample of part of a session with SETUP. The data is being changed from the System defaults to the specifications for a Soroc terminal, as in Appendix B above. All underlined text like this is user input, and all text enclosed in curly brackets {like this} is commentary. Angle brackets <these> are used to enclose the names of non-printing characters {like <return>}. All else is setup's output to the terminal.

```
{to begin, you must eXecute SETUP}

XSETUP<return>

INITIALIZING……
……
SETUP: C(HANGE T(EACH H(ELP Q(UIT [Dl]

{H(ELP tells you about the other commands, and T(EACH
 describes the use of SETUP.  Now is the most profitable
 time to use these commands.
 Suppose you have read H(ELP and T(EACH, and decide
 to change data items by going through the menu.
 You must hit C for C(HANGE.}

C   {Note: these single-character commands don't echo.}
CHANGE: S(INGLE) P(RCMPTED) R(ADIX)
    H(ELP) Q(UIT)

{H(ELP) describes the commands on this particular line,
R(ADIX) allows you to change the base of the numbers
you enter, and Q(UIT) returns you to the SETUP: prompt.
What you want to do now is go through the prompted menu

P



FIELD NAME = BACKSPACE
 OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
    10       8      8         BS      ^H
WANT TO CHANGE THIS VALUE? (Y,N,!)
<return>
WANT TO CHANGE THIS VALUE? (Y,N,!)
```

```
{<return> or <space> will cause this prompt to be repeated
 ! causes an escape to the CHANGE: prompt.
 Since control-H (^H) is indeed the Soroc's backspace
you want to go on.}


N



FIELD NAME = EDITOR ACCEPT KEY
 OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
     0       0      0         NUL     ^@
WANT TO CHANGE THIS VALUE? (Y,N,!)
Y
NEW VALUE: <home>

{When <home > or any other non-printing key
is pressed, ? is displayed.}

OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
    3       3      3         ETX     '@
WANT TO CHANGE THIS VALUE? (Y,N,!)
N



FIELD NAME = EDITOR ESCAPE KEY
OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
    0       0      0         NUL     '@
WANT TO CHANGE THIS VALUE (Y,N,!)
Y
NEW VALUE: <return>

{Any unexpected input here causes the
 relevant section of T(EACH to be output,
 followed by this:}
C(ONTINUE)
{All characters are ignored except C, and
 then the prompt is repeated.}
```

```
C
NEW VALUE: <rubout>  {Again, a ? is echoed.}
OCTAL  DECIMAL  HEXADECIMAL  ASCII
  177     127     7F         DEL
WANT TO CHANGE THIS VALUE? (Y,N,!)

{(Note that there is no corresponding control key.)
 DEL is not the key you meant, so you must
 change it again.}


Y
NEW VALUE: <esc>  {? is echoed.}
OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
   33      27     1B         ESC     ^[
WANT TO CHANGE THIS VALUE? (Y,N,!)
N  {This is what it should be.}



{The menu continues in this way for the rest of
 the data items.  Suppose you have gone ahead and
 answered all of the questions according to the
 Soroc specifications.  After the last data item,
 you again get the menu:}


CHANGE: S(INGLE) P(RCMPTED) R(ADIX)
H(ELP) Q(UIT)

{you realize that you left the prefix for
 ERASE LINE at FALSE, when it should be
 TRUE.  You want to change just this one
 data item.}

S {For S(INGLE)}
NAME OF FIELD: PREFIXED [ERASE]
DIDN'T FIND PREFIXED [ERASE]  {Oops}
NAME OF FIELD: PREFIXED [ERASE LINE]

FIELD NAME = PREFIXED [ERASE LINE]
 CURRENT VALUE IS FALSE
WANT TO CHANGE THIS VALUE? (Y,N,!)
Y
NEW VALUE: TRUE  {T would also work.}
 CURRENT VALUE IS TRUE
```

```
WANT TO CHANGE THIS VALUE? (Y,N,!)
N

CHANGE: S(INGLE) P(ROMPTED) R(ADIX)
H(ELP) Q(UIT)
Q
SETUP: C(HANGE T(EACH H(ELP Q(UIT [D2]
Q  {You're through changing data now.}
QUIT: D(ISK) OR M(EMORY) UPDATE,
R(ETURN) H(ELP) E(XIT)

{you want to do a disk update to create
 NEW.MISCINFO on your disk for future use.}

D
QUIT: D(ISK) OR M(EMORY) UPDATE,
R(ETURN) H(ELP) E(XIT)
E
{And now you're done.  The Pascal system prompt
will appear.}
```

III.5.5 Appendix E — Sample SCREENTEST Log

This is a sample of a SCREENTEST log for a terminal that has some problems.

```
      1 test_DLE_expansion: expansion not happening properly
      2 test_DLE_expansion: expansion not happening properly
      3 test_DLE_expansion: expansion not happening properly
      4 test_DLE_expansion: expansion not happening properly
      5 test_DLE_expansion: expansion not happening properly
      6 test_DLE_expansion: expansion not happening properly
      7 test_DLE_expansion: expansion not happening properly
      8 test_DLE_expansion: expansion not happening properly
      9 test_keyboard: backspace key not correct
     10 test_keyboard: line feed key not correct

***** End Diagnostic; 10 errors encountered.
```

IV. THE ADAPTABLE SYSTEM

IV.1 Introduction

The UCSD p-System can run on any hardware system that in some way emulates the idealized "P-machine" (which is described in the Internal Architecture Guide). In most cases, emulating the P-machine means running a P-code interpreter. Such interpreters have been written for many current microprocessors. However, the hardware in which these microprocessors are packaged varies enormously, and though an interpreter may run on a particular processor, to run on the full hardware system requires tailoring to its requirements: procedures for bootstrapping may vary, disk drives may vary, other remote devices may vary.

The p-System has been configured for many hardware packages. But if your hardware is not one of these packages, you may still be able to use the p-System by using an Adaptable System.

The Adaptable System comes in two flavors. One is for users of Z80/8080/8085 systems that run the CP/M operating system. If you have such a system, the Adaptable System allows you to boot directly from CP/M, and later tailor the p-System to run more efficiently. The other Adaptable System is for users of Z80/8080/8085 systems that do not run CP/M, and users of 6502 systems. This version of the Adaptable System requires more initial programming on the part of the user, and more familiarity with the hardware.

There are two main problems that the Adaptable System handles. One is creating a bootstrap that will bring up the p-System on a particular hardware package (see Chapter II). The other is configuring an SBIOS (Simplified Basic I/O Subsystem) that will handle the peripherals of that hardware package.

Once these problems have been solved, the Adaptable System can also be used to extend the I/O capabilities of the p-System by adding more disk drives, more remote devices, user-defined devices, a system clock, and so forth.

The CP/M Adaptable System is described in Section IV.3, and the Full Adaptable System is described in Section IV.4. Details of various machines are discussed in Chapter V.

Section IV.2 describes a utility called DISKCHANGE. DISKCHANGE allows you to change the sector skew and interleaving of a disk. This is a necessary utility for the Adaptable System; since disk drives vary widely, the Adaptable System is shipped on disks that are uninterleaved, and it is up to the user to choose the interleaving that best suits his or her hardware. For most disk drives, there is one configuration that is vastly more efficient than any other.

IV.1.1 Creating a useful System Disk

The Bootstrapping disks that are provided with Adaptable Systems contain a minimal System. They are intended for booting a System from scratch, not for day-to-day use. Other pieces of the System that you may wish to use frequently are shipped on the disks labelled SYSTEM and UTILITIES.
Once you have booted your System, it is an easy matter to T(ransfer files from one disk to another using the System's Filer. Any disk may be used as a bootable System disk, provided it contains both a bootstrap and the following files;

SYSTEM.PASCAL
SYSTEM.INTERP
SYSTEM.MISCINFO

(Interpreter names may vary from processor to processor, e.g., SYSTEM.INTERP, SYSTEM.PDP_11).
Depending on the work you are doing, you may also want the disk you boot from to contain SYSTEM.EDITOR, or SYSTEM.COMPILER (accompanied by SYSTEM.SYNTAX), or both. SYSTEM.FILER is almost always needed, and SYSTEM.LIBRARY is needed if you use Long Integers or routines you have put in the library yourself. You may also be frequently using an assembler.
All of these files are described in a bit more detail in Chapter 1 of the Users' Manual.

An 8" floppy disk often has enough room to contain all the System components that you frequently use, plus some working space. 5-1/4" floppies are not so roomy, and you may want to make yourself a "working set" of minifloppies.
Note: If you intend to use the Debugger, you must add it to the System. Use the utility LIBRARY to add DEBUGGER.CODE to the file SYSTEM.LIBRARY. Do not alter any segment numbers. Both DEBUGGER and LIBRARY are described in the Users' Manual, Chapter X. Neither of them should be used until you have acquired some familiarity with the System and its use.

IV.2 Relevant Utilities

This section describes DISKCHANGE and DISKSIZE, which are two utilities provided with all Adaptable Systems. DISKCHANGE has two basic purposes. One is to unpack Adaptable System disk images on Systems that have already been booted (see Chapter 1), and the other is to change disk formats. The purpose of DISKSIZE is to change disks that have been unpacked by DISKCHANGE so that their full memory capacity may be used.

Changing disk formats is chiefly done in two situations: when a different format allows your system's disk hardware to function more efficiently, and when you wish to use p-System disks that have been prepared on someone else's hardware using a different disk format.

IV.2.1 Using DISKCHANGE

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

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 the disk formats, to determine what combination of interleaving and skew yields the fastest disk access. The utility FINDPARAMS that is supplied with Adaptable Systems is meant to 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, eX(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, typing an exclamation point ('!') will abort the program.
The first two prompts ask for which disk will be transferred to which disk. It is possible to transfer a disk to itself, but do this only if you have first backed it up, otherwise you are in danger of losing your entire disk.
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 cannot use the answer you give. This value only affects the speed at which DISKCHANGE operates.

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 Adaptable System disks come with one-to-one interleaving and zero skew.
You should bootstrap your system without changing these values; once you are able to run DISKCHANGE, it should be safe to change them to something more efficient.

When DISKCHANGE displays the final prompt of 'Insert system disk and press return', typing 'R' instead of return causes a transfer to be done again with the same parameters. That is, when you type '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 cannot change the parameters, however, without finishing a DISKCHANGE run and starting over again.

CP/M users beware: the CP/M documentation uses the word 'skew' to mean what we call 'interleaving'. The CP/M operating system does not perform what we call 'skew'.

IV.2.2 Using DISKSIZE

If DISKCHANGE is used to unpack a disk by transferring one Adaptable System disk image (1/5 of an 8" floppy) onto another (blank) 8" floppy, the new floppy's directory will still indicate a small disk size (155 blocks). 8" floppies can generally hold about 494 blocks (the exact figure depends on your hardware). In order for your System to access che entire disk, you must use DISKSIZE to change the recorded size of the new disk.

When you eX(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 on your disk.

You can calculate the number of blocks available on your disks by using the bootstrap parameters for your system (these parameters are described in the following sections). 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 )
```

IV.3 The CP/M Adaptable System

The CP/M Adaptable System is intended for Z80 and 8080 microprocessor systems that are already running the CP/M operating system. CP/M provides the facilities for easily bootstrapping the p-System. Once the p-System is booted, it is a self-contained software environment: at no time does p-System software run "under" CP/M. Once the p-System has been booted, it may be improved in a number of ways: device-handling may be made more efficient, and a new bootstrap may be created which eliminates the need for using CP/M to boot the System.

The software supplied with the p-System creates an SBIOS from the CP/M BIOS (the "CBIOS"). It is possible to do this with CBIOS versions 1.4, 2.0, and 2.2 provided they allow 128-byte sectors. It is also possible to boot with the CDOS operating system from Cromemco, provided it is compatible with CDOS version 1.07.

No other versions of CP/M or CDOS may be used to boot the p-System. If this is the case, you must use the Full Adaptable System (described in Section IV.4). CP/M configured for Dynabyte disk drives is not compatible with the CP/M Adaptable System: you must use the Full Adaptable System.
If you have a compatible version running on an IMS 8000 with double density floppy drives, refer to Appendix C for instructions on how to bootstrap.
To bootstrap your System, your hardware must include 48K contiguous bytes of RAM, at least 175K bytes of disk storage, and a CRT or teletype that can send and receive ASCII characters (these requirements are spelled out in full detail in Appendix A).

Once your System has been bootstrapped, you may provide a "cold bootstrap", and otherwise improve the System's performance.

The CP/M Adaptable System is shipped on 8" floppy disks. They are IBM 5740 format: soft-sectored, single sided, single density, and each p-System disk contains the images of three logical disks (see Figures 3 and 4). The data on each disk is not interleaved. Each logical disk can fit on one 5-1/4" minifloppy; if you use such hardware, you must download data from the 8" disk to 5-1/4" disks (see Section I.4).

The three p-System disks that you are shipped are called CPMADAP, SYSTEM, and UTILITIES. The latter two contain System programs, as the names indicate. The CPMADAP disk contains a minimal p-System, intended for booting on 5-1/4" disks. The fourth disk you are shipped is labelled CPMDISK (BOOTER). This disk, unlike the others, is in a CP/M-readable format, and contains a program,

PASBOOT, that runs under CP/M. PASBOOT bootstraps a p-System.
The use of these disks is described below. A catalog of release disks is in Appendix B.

You should read through all of the next section before attempting to bring up your p-System. You should also remember to back up your disks before doing anything else with them.

IV.5.1 Assessing your System

The three critical resources involved in bootstrapping the p-System are RAM memory, floppy disk storage, and I/O drivers.

IV.5.1.1 Memory Configurations

It is possible to bootstrap the p-System with 48K contiguous bytes of RAM devoted exclusively to the System. Little can be done with so little memory, however, and if the system has more memory than this, its performance will be better.

IV.5.1.2 Floppy Disk Requirements

It is necessary that any machine that runs the p-System have at least 175K (550 512-byte blocks) of floppy disk storage. Again, more disk space is needed for most applications of the System.

The p-System is designed to work on any type of floppy disk, including: minifloppies, soft-sectored floppies, hard-sectored floppies, double-density floppies, and double-sided, double-density floppies. The Adaptable System disks are IBM 5740-format: soft-sectored, single-sided, single-density. The information on them is uninterleaved. If the target configuration does not include floppy drives capable of reading the disks that are shipped, the information must be downloaded onto floppies (the "target medium") that your hardware can read (see Section I.4). Whichever floppy disk format you use, you must have 128-byte sectors. If your sectors are of some other size, you must use the Full Adaptable System (see Section IV.4).

IV.5.1.2.1 Format of the CP/M Adaptable System Disks

Each disk shipped (except for CPMDISK (BOOTER)) is divided into three logical disk images. If these disks are used without being unpacked, only the first disk image is visible to the p-System: the second and third disks must be unpacked before they can be used.
The first disk image starts at Track 0, the second at Track 25, and the third at Track 50.

Each logical disk is 25 tracks long: the tracks are logically numbered 0..24. Each track contains 26 sectors (1..26), and each sector is 128 bytes long.
Logical track 0 is reserved for bootstrapping purposes: Sectors 5..18 contain the

secondary bootstrap. Sectors 1 and 2 are empty, but may be used for a primary bootstrap should you write one of your own (see Section IV.3.4). Sectors 19..26 are unused.

Logical track 1 contains a p-System file directory in sectors 9..26.
Logical tracks 2..24 are available for file storage (each disk that is shipped already contains several files).
The information on these disks is uninterleaved. Once the System has been booted, disk formats may be changed to improve performance.

IV.3.1.2.2 Contents of the CP/M Adaptable System Disks

CPMDISK is a CP/M-readable disk with the programs PASBOOT and SAMBOOT. It has a third disk image that bootstraps a System with two disk drives, using CP/M.

CPMADAP contains three logical p-System disks:
1) A System that boots with one disk drive, using CP/M.
2) A System that boots using SBOOT8
(from the Full Adaptable System).
3) Interpreter codefiles and the program CPMBOOT.

SYSTEM also contains three disk images:
1) System files, SETUP, and STARTUP.
2) Pascal Compiler and 8080 Assembler.
3) Some utilities, Linker, and Z80 Assembler.

UTILITIES contains two disk images:
1) COPYDUPDIR, MARKDUPDIR, DECODE, PATCH,
and SCREENTEST.
2) Other utilities.

IV.5.1.5 I/O Drivers

To boot a CP/M Adaptable System, you must be running a CP/M 1.4, 2.0, or 2.2, with a standard CBIOS. CDOS systems will work if and only if they are compatible with CDOS version 1.07 (later systems usually work; earlier systems do not). If you use an Intel MDS (IMS) machine, you must recopy your disks with CRC-checking turned off: see Appendix C for details.

A p-System booted using CP/M will do low-level (SBIOS-level) I/O using CBIOS routines defined in the CBIOS jump table. This is why compatibility is imperative: if the routines are in a different location, the bootstrap program will not find them. In addition, the CBIOS must not use any memory between 0100H and the base of the CBIOS jump table: this is where the bootstrap will load the P-machine Interpreter.

If all the conditions stated in this section are met, you should be able to proceed to the next section, and attempt to bootstrap your System.

IV.3.2 Bootstrapping

To bootstrap your p-System, perform the following steps:
1) Bring up your CP/M system.
2) Insert the CP/M-compatible disk (CPMDISK (BOOTER))
into Drive A.
3) Type 'PASBOOT'<return> to execute the program PASBOOT.

PASBOOT should display the following message:

UCSD PASCAL (IV.O) BOOTER VERSION [zz]

INSERT PASCAL DISK INTO DRIVE A, THEN TYPE <RETURN>

... you should insert the Bootstrapping disk from CPMADAP
(the first disk image on CPMADAP) into Drive A, and then
type <return>.

4) PASBOOT does its work, and displays the following messages:
READING SECONDARY BOOTSTRAP
BOOTING TO UCSD PASCAL

... after this message, you should see the Operating System
promptline, and be able to use your p-System. See the next
section for details on checking the System.

Problems: If PASBOOT encounters any problems, it will halt. The error
message:

Can't find SYSTEM.INTERP

... indicates a problem in the Secondary Bootstrap (See Chapter II). The error
message:

Can't find SYSTEM.PASCAL

... indicates a problem in the Tertiary Bootstrap.

Booting with CP/M should take about two minutes (give or take some time). This
is slow: Section IV.3.4 tells how to supply your own (faster) bootstrap.

IV.3.3 Checking your System

Once your System has been bootstrapped, you can generally tell whether it is working by a few simple observations:

1) The console should display the System promptline, and below that there should be a welcome message.

2) When you type "F" for F(iler, the System disk should do some clicking (it is reading several sectors), and then the console should display the Filer's promptline.

3) While in the Filer, type 'D' for D(ate, and enter the current date, followed by <return>. Then type 'D' again. The second time you use the D(ate command, it should display the date that you entered the first time.

... if (1) fails, almost anything could be wrong: reread Section IV.1 to make sure your hardware and CP/M software conform to requirements. You might want to check that PASBOOT creates the correct booting parameters, and that the CBIOS disk read and console write routines are correct (more information about the bootstrap stack and these routines may be found in Section IV.4). If (2) fails, check your disk read and console read/write routines. If (3) fails, check your disk routines.

Some more hints about troubleshooting appear in Appendix C. If you are truly stuck, you should contact the supplier of your software for support.

IV.5.3.1 Notes

When your System is bootstrapped, you should be able to use all devices that CP/M communicates with:

the line printer is PRINTER:, Device 6
the tape reader is REMIN:, Device 7
the tape punch is REMOUT:, Device 8

... however, you will only be able to communicate with one disk drive. To communicate with more than one disk drive, bootstrap with the third disk image on CPMDISK (BOOTER). When the System is booted using the Interpreter on this disk, both disk drives must contain a disk (otherwise, the System will "hang" while booting

The following are p-System numbers for disk drives:

CBIOS	p-System
0	4
1	5
2	9
3	10

... the p-System is always booted from disk drive #/4.

The keys for STOP/START, FLUSH, and BREAK do work on the p-System that is booted using CP/M. Input from the keyboard is queued if there is some other I/O going on.

The Bootstrapping Disk from CPMADAP contains only a minimum System, intended for bootstrapping, nothing more. Many useful files are on the other disks supplied: once your System is booted, you may use the Filer to T(ransfer frequently-used files onto frequently-used disks, and arrange things to your own convenience.

IV.3.4 Improvements

Once the p-System has been bootstrapped using CP/M, it is possible to speed up disk accesses and to provide an automatic bootstrap (a cold boot). Disk access speed may be improved by changing disk formats to better match the disk drives being used.

IV.3.4.1 PASBOOT

PASBOOT is an assembly language program that runs under CP/M and boots the p-System. It reads the Secondary Bootstrap from the Bootstrapping Disk (the first disk image on CPMADAP; the Secondary Bootstrap is on Track 0, Sectors 3..18) into main memory, starting at 8200H. It then pushes parameters that describe the target machine onto the processor stack, and initiates the bootstrap by jumping to 8200H.

The source for PASBOOT is supplied on the disk CPMDISK (BOOTER). Several equates in the source may be modified to change the conditions of bootstrapping:

DDT - normally FALSE. If set to TRUE, the System may be traced and debugged using DDT.

BOOT - is the address of the JMP WBOOT for CP/M's CBIOS. The default is 0000H. This is used when DDT = FALSE.

BDOS - is the address of the JMP BDOS for CBIOS. The default is 0005H. This is used when DDT = TRUE.

TPA - the address of the start of a user program when assembled under CP/M. The default is 0100H.

INTERP$BASE - is the starting address of the Interpreter. Normally equal to TPA. Must always be on a page boundary (i.e., the low byte must equal 00H). Default is 0100H.

LOW$MEMORY - the lowest available RAM address. It must be the base of a contiguous block of at least 48K of RAM, must be greater than or equal to INTERP$BASE, and must be on a page boundary. The default is 0100H.

TRACKS - the number of tracks on the Bootstrapping Disk. The default is 77 (as on standard 8" floppies).

SECTORS - the number of sectors per track on the Bootstrapping Disk. The default is 26.

BYTES - the number of bytes per sector on the Bootstrapping Disk. The default is 128. To boot using CP/M, this must be 128.

INTERLEAVE - the ratio for interleaving sectors on a track. This parameter is termed 'skew' in CP/M documentation; in UCSD p-System parlance, 'skew' refers instead to a track-to-track offset: see below. The INTERLEAVE parameter is interpreted as INTERLEAVE:1. The default is 1; the p-System disks as shipped have 1:1 interleaving, i.e., none at all.

FIRST$TRACK - the track on which Block 0 of the p-System starts. p--System code is typically stored after bootstrap code. The default is 1.

SKEW - is the track-to-track sector offset. This does not mean what 'skew' means in CP/M documentation. In the p-System, skew is a track-to-track offset: if SKEW=6, then Sector 1 on one track is adjacent to Sector 7 on the next track, which is adjacent to Sector 13 on the next track, and so forth.

MAX$SECTORS - is the maximum number of sectors per track that an online disk drive may have. For example, if a system supports one single-density floppy drive (with 26 sectors/track) and one double-density floppy drive (with 52 sectors/track), this value should be 52. MAX$SECTORS is used to allocate a sector translation-table. The default is the value of SECTORS.

MAX$BYTES - is the maximum number of bytes per sector that an online disk drive may have. MAX$BYTES is used to allocate a partial-sector read buffer. The default is the value of BYTES (= 128).

IV.5.4.2 Allowing Empty Disk Drives

The standard CBIOS method of handling an empty disk drive is to emit error messages until a disk is placed in the drive. The p-System, on the other hand, is able to handle empty disk drives. This is a good deal more convenient.
For this to be the case, all the disk drive routines in the CBIOS/SBIOS must be modified so that they return a status (IORESULT) in the A register. This status must be:

```
0  ... if read or write successful
9  ... if disk not online
1  ... if any other error
```

The CBIOS/SBIOS routines should not under any circumstances display error messages: that is the responsibility of the p-System.

IV.3.4.5 Changing Disk Recording Formats

The CP/M Adaptable System disks as shipped are formatted with a sector interleaving of 1:1 and a track-to-track sector skew of 0. For most disk drives, these values are inefficient. The FINDPARAMS utility can be used to determine more efficient parameters for your particular disk drives.
There is a copy of FINDPARAMS on each Bootstrapping Disk. When you run it, it presents you with a series of prompts and tests which allow you to judge which combination of parameters works best with your disk drives. Once you know what these parameters are, you may use DISKCHANGE to alter the disks you use (DISKCHANGE is described in Section IV.2.1).

On some slow processors, FINDPARAMS is incapable of determining the proper interleaving. In such cases, it is suggested that the user take the time to determine optimum interleaving and skew by hand. The best way to do this is to do a "binary search:" logging the bootstrap time for different interleavings, and successively improving the interleave values. The lowest interleaving that is markedly faster than the next smaller interleave value is the one that should be used. Optimum skew can be determined in a similar manner. However, skew does not affect disk access speed as much as interleaving does, so the difference between skew values will be less apparent.
When you have used DISKCHANGE to alter your Bootstrapping Disk, you must change the relevant parameters in PASBOOT:

```
INTERLEAVE
FIRST$TRACK
SKEW
```

... and then reboot your System. If you change the Bootstrapping disk but neglect to alter these parameters, your System will no longer bootstrap. This is one reason it is important to keep backups of all your System disks.
Note that at this point in the use of your System, all disks must have the same recording format; if you optimize your Bootstrapping Disk, you must optimize all other disks that you use.

Warning: Before you use DISKCHANGE on a disk, you should back it up. DISKCHANGE may not do exactly what you wanted it to. You may have forgotten to change the parameters in PASBOOT. Also, DISKCHANGE will lose all information on a disk that is not part of the disk image it is altering: you must unpack your CP/M Adaptable System disks before you optimize them with DISKCHANGE. Section I.4 describes how to unpack Adaptable System disks.

Many soft-sectored 8" floppies in the field are formatted with 2:1 interleaving, sector skew of 6, and first Pascal track of 1. This format is recommended if vou wish to exchange software with other users. But DISKCHANGE can be used to convert p-System disks to any desired format.

IV.5.4.4 Creating an Automatic Bootstrap

It is possible to write a bootstrap (a primary bootstrap) and place it on a System disk so that the disk will boot without the aid of CP/M. The hardware you use must. be able to read a primary bootstrap stored at Track 0, Sector 1 of the System disk. The bootstrap is loaded into some predetermined location in main memory. On many hardware systems, an operation of this sort takes place when a bootstrap button is pressed.

If your hardware is capable of loading a bootstrap in this manner, you must write a new primary bootstrap. This primary bootstrap must read the CBIOS/SBIOS into memory, load the secondary bootstrap, and start it.
The CPMBOOT utility is provided to transfer the primary bootstrap and a CBIOS onto Track 0 of a Bootstrapping Disk (starting at Sector 1).
The primary bootstrap that you write may be based on the PASBOOT program, but it must load the CBIOS/SBIOS: it cannot assume it is already in memory. The primary bootstrap must do the following things;

1) Read the secondary bootstrap from Track 0, Sectors 3..18, into main memory starting at 8200H.

2) Read the CBIOS/SBIOS from Track 0, Sectors 19..26, into main memory (the actual location is optional; it is best to follow the example of PASBOOT).

3) Load the configuration parameters onto the processor stack (the source for PA5BOOT indicates how to do this).

4) Perform a JP (not a CALL) to the secondary bootstrap (at 8200H).

The primary bootstrap must be on disk wherever the bootstrap button will read it: the preferred location is Track 0, Sector 1.
For more information on the primary bootstrap, refer to the SAMBOOT source program on CPMDISK.

Once you have written and tested a primary bootstrap, you may load it onto a Bootstrapping disk, along with a copy of CBIOS/SBIOS, by running the utility CPMBOOT (located on the disk CPMADAP). CPMBOOT prompts you for the names of a file to load as the primary bootstrap and a file to load as CBIOS; it then copies these files onto Track 0 of a Bootstrapping disk. Once CPMBOOT has been run, the Bootstrapping disk it created should be able to boot without using CP/M.

CPMBOOT can only write to a standard 8" disk. If your disk drives are not IBM 5740 format, you must load the primary bootstrap and CBIOS onto the disk in some other manner.

IV.5.4.5 Changing the P-Machine Interpreter

The Interpreter that is shipped with the System may not have the characteristics you desire. A different Interpreter may be created by linking together codefiles that support the features you wish to support. How to do this is described in detail in Section V.I.4.

The only exception from Section V.I.4 that you must note if you are using the CP/M Adaptable System is that the CBlOS-interface files are different. Instead of INTER.CODE and INTER.X.CODE, you have your choice of INTER.CPM1.CODE, INTER.CPM2.CODE, and INTER.CPM4.CODE: these codefiles interface with CBIOSes that support 1, 2, and 4 disks (they must use BIOS.C.CODE). The Interpreter that is shipped is INTERP.CODE linked with RSP.CODE, BIOS.C.CODE, INTER.CPM1.CODE, and TERTBOOT.CODE.

IV.3.4.6 Using the Full Adaptable System

The CP/M Adaptable System is merely a simple interface to the Full Adaptable System. You can take advantage of the features of the Full Adaptable System, but to do this you must take the time to read the following section, Section IV.4, and possibly do some SBIOS programming of your own.

The second disk image on CPMADAP contains a secondary bootstrap that is the standard bootstrap used with the Full Adaptable System. If you decide to use more Full Adaptable System features, use this disk image.

With the Full Adaptable System, you may add device drivers to; support more disks, support disks of different formats, drive printer and remote devices, drive user-defined devices, and read a hardware clock.

IV.4 The Full Adaptable System

The full Adaptable System of Version IV.0 is intended for Z80, 8080/85, and 6502 microprocessor systems. With the full Adaptable System, it is possible to boot the p-System on machines that have at, least 48K bytes of RAM of which at least 56K are contiguous. They must also have a minimum disk storage capacity of 175K bytes (550 512-byte blocks). The actual floppy drives may be of any type (single density, double density, mini, hard sectored, etc). The hardware must also include a teletype or CRT display that can both send and receive ASCII characters. In addition to a UCSD p-System, the Adaptable System disk contains special bootstrapping and testing utilities that enable the user to bootstrap the p- System quickly and reliably. They are set up to bootstrap on particular memory configurations. If your hardware is not so configured, an alternate bootstrap is provided that should execute on the target hardware configuration. An I/O0 module that communicates with the floppy disks and console must be provided by the user to run on the target configuration.

Once the the p-System has been bootstrapped, additional facilities may be provided to communicate with a printer, remote device, other user-defined devices, and a system clock. The I/O configuration may be extended to provide access to different types of floppy drives on line at the same time.

The Adaptable System is shipped on four 8" diskettes. They are IBM 5740 format: soft-sectored, single-sided, single-density, and they contain the virtual images of three logical disks (see Figures 5 and 4). The data on each disk is uninterieaved. Each logical disk is small enough to fit on one 5-1/4" minifloppy, and can thus be downloaded (see Section I.4).

One of the Adaptable System disks is called SYSTEM. Another is called UTIL. The other two are called ADAP disks; you will probably need to use only one of them. If your System has a Z80, 8080, or 8085 processor, the ADAP disks are ADAPZ and ADAP8; use ADAPZ for the Z80 and ADAP8 for the 8080/5. If you have a system with a 6502 processor, the ADAP disks are called HI PAGE and LO PAGE; which one you must use depends on your memory configuration: see below, Section V.5.5.

Further, each ADAP disk contains two Bootstrap disks, and one Interpreter disk. Which of the Bootstrap disks you must use also depends on your memory configuration; see the following section.

You should read through all of the next section before attempting to bring up your p-System. You should also remember to back up your disks before doing anything else with them.

IV.4.1 Assessing the Situation

The three critical resources involved in bootstrapping the p-System are RAM memory, floppy disk storage, and I/O drivers.

IV. 4.1.1 Memory Configurations

It is possible to bootstrap the p-System with 48K bytes of exclusively devoted memory. A minimum of 36K contiguous bytes must be available for user data space (this is referred to throughout the following discussion as 'the large contiguous RAM space'). There are three software modules that at certain times must coexist in RAM. They are the SBIOS, the Interpreter, and the Bootstrap. The SBIOS is an I/O package customized to the target configuration (see Section IV.4.3). It can be located wherever it fits best in RAM. From the p-System's point of view, it is best situated in a small, isolated RAM space of its own. If such an area is not available, the SBIOS may occupy as much of the high-address memory space (of the targe contiguous RAM) as is necessary.

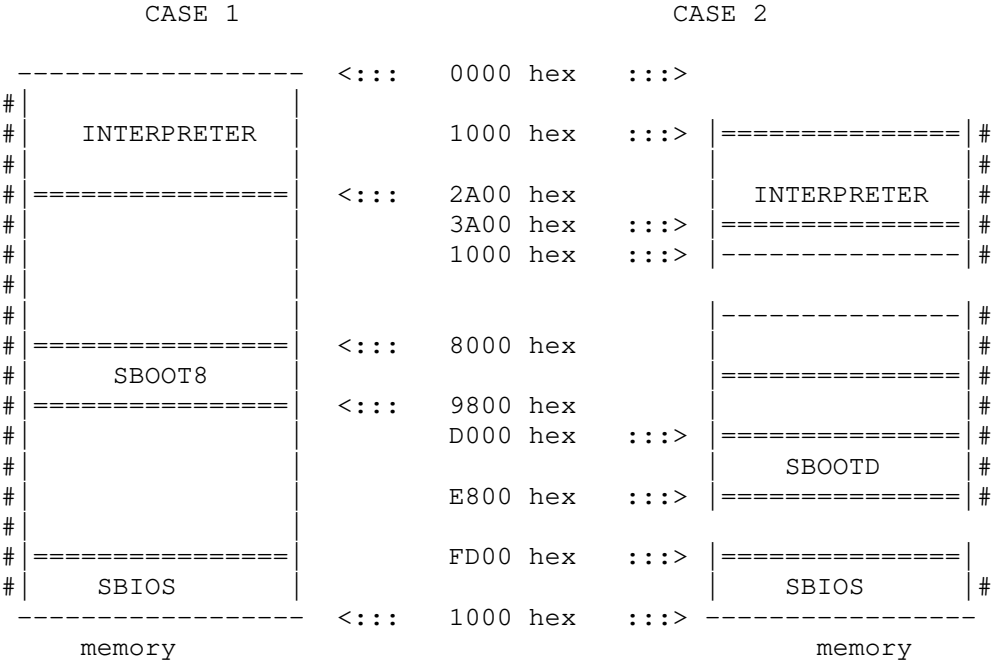
The primary and secondary bootstraps occupy 1800 hex bytes (including data storage), and execute in the large contiguous RAM space. If this contiguous RAM space starts at or before location 3000 hex, the default bootstrap may be used. It runs at 8000 hex and is called SBOOT8. If the large contiguous RAM starts after 3000 hex, an alternate copy of the bootstrap must be used. It runs at DOOO hex and is called SBOOTD. Once it is finished running, the memory space occupied by the bootstrap is returned to the large contiguous memory pool.

Finally, the Interpreter is approximately 12K bytes long, and runs either at the start of the large contiguous memory space, or in a separate RAM space. If there is a separate RAM space large enough to accommodate the Interpreter, it should be used rather than the large contiguous space. (The exact size of the Interpreter can be obtained by examining the last word of the SYSTEM.INTERP file after the System has been bootstrapped.)

Details about various processors, including possible memory configurations, are given in Chapter V.

IV.4.1.1.1 Sample Configurations

To illustrate these requirements, we show two cases. The first is the simple case of a configuration that has 64K of RAM available. In the second case, the configuration provides a 16K RAM between 1000 hex and 5000 hex, a 36K RAM between 6000 hex and F000 hex, and a 300H-byte RAM between FD00 hex and FFFF hex. In both cases, assume the SBIOS is 300H bytes long, the Bootstrap is 1800H bytes long, and the Interpreter is 2AOOH bytes long.



('#' indicates the presence of memory)

In the first case, there is no separate RAM space for the Interpreter. Therefore, it must start where the large contiguous RAM space starts. Since there is RAM at 8000 and the large contiguous RAM starts before 3000 hex, the bootstrap SBOOT8 is used. Finally, the SBIOS is located so as not to fragment the large contiguous memory space.

In the second case, there is a separate RAM space that is large enough to hold the Interpreter. The Interpreter is therefore located there. There is also a separate RAM space large enough to hold the SBIOS. The SBIOS is therefore located there. Finally, since the large contiguous space does not start before 3000 hex, the SBOOTD bootstrap must be used at D000 hex.

IV.4.1.2 Floppy Disk Requirements

It is necessary that any machine that runs the p-System have at least 175K bytes (350 512-byte blocks) of floppy disk storage. It is possible to bootstrap with less disk space, but virtually impossible to do anything else.

The p-System is designed to work on any type of floppy medium. This includes mini-floppies, soft-sectored floppies, hard-sectored floppies, double-density floppies, and double-sided, double-density floppies. The Adaptable System disks are IBM 3740-format; soft-sectored, single-sided, single-density. The information on them is uninterleaved. If the target configuration does not include floppy drives capable of reading the bootstrap disk, a copy of the Bootstrapping disk must be created on a disk (called the "target medium") that the available floppy drives can read (see Section I.4).

IV.4.1.2.1 Format of the Adaptable System Disk

The Adaptable System disk is logically divided into three disk images of 25 tracks apiece (see Figure 3). The first disk image (tracks 0 .through 24) contains a Bootstrapping disk that boots with the SBOOT8 bootstrap. The second disk image (tracks 25 through 49) contains a Bootstrapping disk that boots with the SBOOTD bootstrap. The third disk image (tracks 50 through 74) contains the Interpreter disk. The first disk image is the only one of the three logical disks that is accessible to the p-System when it is first booted. For this reason, the SBOOT8 bootstrap is considered the default bootstrap.
A logical disk image has 25 tracks, numbered 0 through 24. Each track contains 26 sectors, numbered 1 through 26, with 128 bytes per sector.

Logical track 0 is reserved for the bootstrap. Sectors 1 and 2 contain the Primary bootstrap, and sectors 3 through 18 contain the Secondary bootstrap, which is overlayed into memory as bootstrapping progresses. Sectors 19 through 26 are not used.

Logical track 1, sectors 1 through 8 are reserved for the SBIOS tester (see Section IV.4.2). Sectors 9 through 24 are occupied by the disk's directory.
The remainder of the logical disk (as shipped) may contain other System files.

IV.4.1.2.2 Preparing the Disk for Bootstrapping

To boot the p-System, an appropriate Bootstrapping disk image must occupy tracks 0 through 24 of the target medium.

In this case, "appropriate" means that the bootstrap must be for the proper address (i.e., either SBOOT8 or SBOOTD, as explained elsewhere), and the bootstrap must be in its proper location (i.e., if downloading was necessary, it must have been done correctly).

If the target medium is an 8" soft sectored floppy, and SBOOT8 should be used, there is no work necessary. The SBOOT8 Bootstrapping disk image is already on tracks 0 through 24 of the Adaptable System disk.

If instead it is necessary to bootstrap with SBOOTD, the contents of the second disk image (tracks 25 through 49) must be copied onto tracks 0 through 24 of some disk (it is far safer to copy from the disks provided onto fresh disks; in any case, your disks should have already been backed up). The copying may be done by any means available to you (such as possibly a native operating system, a ROM monitor, an assembly language program, et cetera).

If the target medium is not an 8" soft sectored floppy, you must download your disks onto the appropriate media (see Section I.4), unpacking them in the process. When you have done, this, you should be able to boot using either SBOOT8 or SBOOTD, depending on your memory configuration as described above.

When downloading, you must be careful not to change the information you are transferring: the order of bytes or sectors must not be changed, and address boundaries must not be disturbed.

Remember that for the purposes of initial bootstrapping, the information on a logical disk image is recorded in contiguous sectors (i.e., the sectors of the disk are not interleaved).

Once you have bootstrapped your System, you will probably want to change the disk so that sectors are interleaved, since this is far more efficient on most floppy drives. The DISKCHANGE utility allows you to do so with little difficulty (see Section IV.2).

IV.4.2 SBIOS

IV.4.2.1 Introduction

The Simplified Basic I/O Subsystem is a collection of low-level input/output routines. Fifteen of them are required for the p-System's operation; an additional thirteen routines may be added at the user's discretion (see below, Section IV.4.4). The SBIOS routines must perform device-level I/O functions. While the P- machine is running, they are called by the BIOS. Since they are machine- specific, they cannot be provided with an Adaptable System: the user must write them from scratch, or adapt them from low-level I/O routines already in the user's possession.

A correct SBIOS and a correct bootstrap enable the p-System to run on the user's hardware.

This section first describes the SBIOS routines and their requirements, in order. Then it describes how SBIOS routines are called, and concludes with a section on how to test an SBIOS. The following section describes how to bootstrap the System, with suggestions about writing a bootstrap.

IV.4.2.2 The SBIOS Routines

These are the names of the fifteen essential SBIOS routines, along with a brief description of each routine. SBIOS routines are called through a jump table (see Section IV.4.2.2.5); the center column shows each routine's number in the table (the "jump vector"),

Routine Name	Vector Number	Description
SYSINIT	0	Initialize machine
SYSHALT	1	exit UCSD PASCAL
CONINIT	2	console initialize
CONSTAT	3	console status
CONREAD	4	console input
CONWRIT	5	console output
SETDISK	6	set disk number
SETTRAK	7	set track number
SETSECT	8	set sector number
SETBUFR	9	set buffer address
DSKREAD	10	read sector from disk
DSKWRIT	11	write sector to disk
DSKINIT	12	reset disk
DSKSTRT	13	activate disk
DSKSTOP	14	de-activate disk

IV.4.2.2.1 The Individual SBIOS Routines

SBIOS routines are called by the primary bootstrap and by the BIOS; rarely, if ever, by the user's programs. They are sometimes passed parameters on the stack, and sometimes return results in registers or main memory. The conventions of parameter passing vary (necessarily) from processor to processor: see Chapter V for details.

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 is not online. (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 (numbered 0..5 for SBIOS purposes) 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 ways 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 is not used by the bootstrap; it is provided for use with some routines in the Extended SBIOS: see Section IV.4.4.2.

SYSHALT

SYSHALT is called when the p-System terminates (through a H(alt). It should shut down all devices in an orderly manner. If the user so desires, SYSHALT may also start another operating system on the host machine.

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. Any interrupt vectors associated with the console should be set in SYSINIT, not CONINIT.

If CONINIT encounters no problems in initializing the console, it should return a 0 (zero). If it detects that the terminal is offline, 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 online, the parameter should return 0; if the console is offline (disconnected), the parameter should return 9.

The second parameter describes the state of the console input channel. If a character has been typed on the keyboard, the parameter should return FF hex; otherwise it should return 0. (Note: CONSTAT does not 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 online and a character is pending, CONREAD reads that character. If the console is online but no character is pending, CONREAD waits, 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 offline, 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 exactly 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 online, the character is sent, and CONWRIT returns 0. If the console is offline, CONWRIT returns 9. If there is a transmission problem, CONWRIT returns 1: the System will assume that the character was lost.

CONWRIT should not 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 will not 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..5.

SETDISK merely changes a value; it does not 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 highest numbered track on the disk.

Like SETDISK, SETTRAK merely changes a value; it does not 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 highest numbered sector on a track.
SETSECT does not 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 does not 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 should not change these values.

If the read was successful, the status should return 0. If the disk was offline 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 should not 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

DISKWRIT 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 should not change these values.

If the write was successful, the status should return 0. If the disk was offline 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 should not 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 should not 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 online (i.e., 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 should not alter the values of CURDISK, CURTRAK, CURSECT, and CURBUFR.

DSKINIT is only called when the System is booted or re-initialized (i.e., after SYSINIT is called). It is not 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 init 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 are not done automatically (by the hardware) as consequences of read, write, and init operations.

DSKSTRT does not return a status.

This routine is intended for use with certain mini-floppy drives (5-1/4"). Most 8" floppies will not 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 init 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 are not done automatically (by the hardware) after read, write, and init operations. DSKSTOP does not return a status.

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

IV.4.2.2.2 Where to Get the SBIOS Routines

The SBIOS routines are deliberately simple. Similar low-level I/O handling routines may be found in most operating systems or ROM monitors. While you may if you choose write all of the SBIOS from scratch, it may be possible to find these or similar routines in your hardware's ROM, and if not in ROM, they may be included in software that you already have for your system, or may be available from the manufacturer of your hardware.

IV.4.2.2.3 What to Do with SBIOS Routines

The SBIOS must be edited and assembled on whatever operating system or other computer is available. It may also be assembled by hand.
The SBIOS routines should adhere to the specifications given in this section. Be careful that the routines which return status values do return the correct value: the System relies on this information.

IV.4.2.2.4 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:

```
SBIOS                           ;Beginning of the SBIOS
                JUMP  SYSINIT   ; Jump to SYSINIT routine
                JUMP  SYSHALT   ; Jump to SYSHALT routine
                JUMP  CONSTAT   ; Jump to CONSTAT routine
                JUMP  CONREAD   ; Jump to CONREAD routine
                ..
                ..
                ..
                JUMP  DSKSTRT   ; Jump to DSKSTRT routine
                JUMP  DSKSTOP   ;Jump to DSKSTOP routine

CURDISK         .WORD           ;Temporary area
CURTRAK         .WORD


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

SYSHALT         HALT            ; Dying on this machine is simple
```

IV.4.2.2.5 How to Call the 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.
For the simple SBIOS, there are 15 jumps: each one to the start of a different SBIOS routine. The jumps are arranged in vector number order (see the list of SBIOS routines in Section IV.4.2.2).

The following steps show how to call an SBIOS routine:

```
      STEP 1:
      Calculate the offset to the jump instruction.  This is:
      (the SBIOS routine's vector number) *
      (the number of bytes in a jump instruction);
      STEP 2:
      Add the offset from STEP 1 to the address of the SBIOS
      (that is, the start of the jump vector);
      STEP 3:
      Execute the jump instruction (and the subsequent routine).
```

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 be a subroutine call (whatever that means on your hardware). Each individual SBIOS routine is responsible for returning to its caller.
Paramter-passing conventions for SBIOS routines vary from processor to processor. See Chapter V for details on your particular machine.

IV.4.2.5 Testing the SBIOS

The conditions for testing an SBIOS are the same as the conditions for bootstrapping the System. Namely, you must have a complete SBIOS, you must have selected the appropriate disk to bootstrap from, and you must have set up some parameters on the processor's stack.

Building an SBIOS is described in the previous section. Selecting the appropriate disk to bootstrap from is described in Section IV.4.1. Setting up parameters on the stack is described below in Section IV.4.2.5.1; the section on bootstrapping (Section IV.4.3) refers back to this section.

It is unwise to try to bootstrap without first testing the SBIOS. Should there be problems with the SBIOS which cause your System to fail, there will be no way to tell what went wrong. Running SBIOSTESTER is therefore an important step: if your SBIOS passes these tests, it is likely to work when you bootstrap your System and run it.

SBIOSTESTER is a utility program that resides on the Bootstrapping disk. It is located in track 1, and therefore does not appear in the directory. SBIOSTESTER includes tests for each SBIOS routine, including very thorough tests of each disk drive.

Before SBIOSTESTER can be run, it must be given a set of parameters that describe the configuration of the host hardware. These parameters are placed on top of the processor's stack (which differs from machine to machine: see Chapter V).

Once the SBIOS has passed its tests, you will be ready to bootstrap your System. The parameters reguired by the bootstrap are the same as the parameters required by SBIOSTESTER. Thus, Section IV.4.5 on bootstrapping refers back to the following section, Section IV.4.2.3.1.

IV.4.2.5.1 Loading Parameters on the Stack

A number of parameters must be passed on the processor stack to SBIOSTESTER (and later, when you are ready, to the secondary bootstrap). These parameters describe the configuration of the target machine: the characteristics of the Bootstrapping disk, the current memory configuration, and other miscellaneous items.

Each parameter is a 16-bit word. Hardware stacks differ from processor to processor: you must refer to Chapter V for full details about your own machine.

The parameters must appear on the stack in the following order:

```
top of stack -->  highest numbered floppy drive to test
                  address of the Interpreter
                  address of the SBIOS
                  address of the lowest word of contiguous memory
                  address of the highest word of contiguous memory
                  number of tracks per disk
                  number of sectors per track
                  number of bytes per sector
                  interleaving factor
                  first Pascal track
                  track-to-track skew
                  maximum number of sectors per track for all disks
                  maximum number of bytes per sector on any disk
```

IV.4.2.3.1.1 Individual Parameters

Here is a description of each parameter, in order:

Highest Numbered Floppy Drive to Test

SBIOSTESTER tests all disk drives. Disk drives are numbered from 0 (which is the drive from which the System must be bootstrapped). For example, if this parameter is 1, SBIOSTESTER tests drives 1 and 0. For practical testing purposes, this parameter should always be 5. This is to ensure that proper error messages are generated when the System attempts to access a disk that is not there.

When the System is actually booted, this parameter should be 0 (on 6502 Systems and Z80/8080 Systems with the supplied bootstrap) or not present (on Z80/8080 Systems with a user-written bootstrap).

Address of the Interpreter

The appropriate address for the Interpreter depends on your memory configuration. See Section IV.4.1.

Address of the SBIOS

The appropriate address for the SBIOS also depends on your memory configuration. See Section IV.4.1.

Bounds of the Large Contiguous RAM

The next two parameters are the addresses of the first and last words in the large contiguous RAM space. (I.e., these addresses must be even.)

As described in Section IV.4.1, the p-System requires a minimum of 36K contiguous bytes of RAM (Random Access Memory). When running SBIOSTESTER or booting the System, this space must be absolutely free for the System's use. It can of course be larger than 56K.

The SBIOS must not reside within this space. The Interpreter need not reside within this space, but if it does, it must start at the beginning of this space, and be wholly contained within it.

These issues are discussed in Section IV.4.1, and details about individual processors are given in Chapter V.

Tracks per Disk

The number of tracks on a single floppy disk. At this stage of bringing up your System, all disk drives must be identical. Section IV.4 describes how to maintain several different floppy disks formats on one System at one time.

Sectors per Track

The number of sectors on each track of a floppy disk.

Bytes per Sector

The number of bytes in a single floppy disk sector. This must be either 128, 256, or 512. The SBIOS routines DSKREAD and DSKWRIT transfer information to and from memory a sector at a time.

Miscellaneous Parameters

The remaining four parameters are for use after your System has already been bootstrapped at least once. They allow for more efficient use of the floppy disk drives.

When you run SBIO5TESTER, these parameters must have the following values:

```
interleaving factor   =   1
first Pascal track    =   1
track-to-track skew   =   0
maximum number of sectors per track   =   sectors per track
maximum number of bytes per sector    =   bytes per sector
```

IV.4.2.5.1.2 Sample Configurations

The memory-configuration parameters in this example are taken from the example in Section IV.4.1.1.1.

In Case 1 there are two 8" floppy drives online, with 77 tracks/disk, 26 sectors/track, and 128 bytes/sector. In Case 2, there are six mini-floppy drives online, with 55 tracks/disk, 10 sectors/track, and 256 bytes/sector.

These are the two parameter stacks (values are shown in hex):

	CASE1		CASE 2
top of stack --->			
0005	=	highest numbered floppy drive to test	= 0005
0000	=	address of Interpreter	= 1000
FD00	=	address of of SBIOS	= FD00
0000	=	address of low word of contiguous RAM	= 6000
FDFE	=	address of high word of contiguous RAM	= F000
004D	=	number of tracks per disk	= 0023
001A	=	number of sectors per track	= 000A
0080	=	number of bytes per sector	= 0100
0001	=	interleaving factor	= 0001
0001	=	first Pascal track	= 0001
0000	=	track-to-track skew	= 0000
001A	=	maximum number of sectors per track	= 000A
0080	=	maximum number of bytes per sector	= 0100

... note that the interleaving, first Pascal track, and track-to-track skew are the same in both cases.

IV.4.5.2 Running SBIOSTESTER

IV.4.3.2.1 Loading SBIOSTESTER into Memory

The program SBIOSTESTER is present on the first 1024 bytes of track 1 of all Adaptable System Bootstrapping disks (on 8" disks, it occupies sectors 1..8). It must be loaded into the same location in memory as you would load your bootstrap: either 8000 hex or D000 hex.

SBIOSTESTER may be loaded in any way you choose. Perhaps a small assembly language program could be written to do this (you have already provided disk read routines when you constructed your SBIOS), or perhaps you have another operating system that enables you to load code into memory.
If you are not sure at which location you should load SBIOSTESTER, refer to Section IV.4.1.

IV.4.5.2.2 Executing SBIOSTESTER

Once SBIOSTESTER has been loaded into main memory, and the proper parameters have been pushed on the processor stack, execute it by performing a JUMP to SBIOSTESTER (either to 8000H or D000H). Do not call SBIOSTESTER with a subroutine call.

While SBIOSTESTER is running, there should be a blank disk in every disk drive. SBIOSTESTER simply reports most problems and goes on. Some problems are serious enough to cause it to abort. Since SBIOSTESTER displays its progress on the console, you must watch the console while SBIOSTESTER is running to know whether your SBIOS is passing all tests (SBIOSTESTER displays more information than will fit onto a single screen).

SBIOSTESTER performs the following actions:

Step 1: SYSINIT is called to initialize the SBIOS, and CONINIT is called to initialize the console.

Step 2: If step 1 is successful, the following prompt appears on the console:
Insert blank disks into all drives then hit the <return> key.

Warning: The contents of all disks in the tested drives will be destroyed by the SBIOS read/write tests!

Step 3: CONREAD should read the <return> key when you hit it. If this is the case, the highest numbered drive is declared the current disk (CURDISK) by a call to SETDISK.

Step 4: DSKSTRT and then DSKINIT are called to initialize the floppy drive. If DSKINIT returns a status of 0,

```
      Testing disk x
```

... is displayed on the console (where x is the current disk).

If DSKINIT returns a status of 9,

```
      Disk x is not on line
```

is displayed on the console, and the test jumps ahead to step 9.

If neither of these messages appear on the console, there is a problem with either CONWRIT, DSKSTRT, or DSKINIT

Step 5: A data pattern is written on each sector of each track of the current disk, using SETTRAK, SETSECT, and DSKWRIT Before each call to DSKWRIT

```
      Writing track xx, sector yy
```

... appears on the console, where xx is CURTRAK and yy is CURSECT (both numbers appear in hex).

If DSKWRIT returns a status of 16,

```
      Bad data transfer on track xx, sector yy
```

... appears on the console. The values are the same as above.

Step 6: Each sector of each track on the current disk is read, using SETSECT, SETTRAK, and DSKREAD. Before each call to DSKREAD,

```
      Reading track xx, sector yy
```

... appears on the console (values as in Step 5).

If DSKREAD returns a status of 9, or the pattern read is not the same as the pattern written in Step 5, an error message appears (the same as in Step 5).

Step 7: This is a more elaborate disk write test.

SBIOSTESTER writes a unique data pattern on one sector of each track using SETTRAK, SETSECT, and DSKWRIT. The tracks are accessed starting from the disk's middle track and alternating from one side of the middle track to the other until the outer tracks are reached. As each sector is recorded, its location is displayed on the console (as in Step 5). If DSKWRIT returns an error status, the error message of Step 5 appears on the console.

Step 8: This is a more elaborate disk read test, using the information generated by Step 7.

SETTRAK, SETSECT, and DSKREAD are used to read the sectors written in Step 7. As each sector is read, its location appears on the console (as in Step 6). If DSKREAD returns an error, or the sector's contents do not correspond to what was written, the error message of Step 5 appears on the console.

Step 9: DSKSTOP is called to de-activate the current disk. CURDISK is decremented. If CURDISK is 0 or greater, SBIOSTESTER branches back to Step 4, and a new floppy drive is tested.

Step 10: The following message appears on the console:

```
      Test complete
```

... and SYSHALT is then called.

IV.4.2.3.2.5 After Running SBIOSTESTER

If SBIOSTESTER runs through to completion, and if it displays no error messages, you may attempt to bootstrap your System with some degree of confidence. Otherwise, debug your SBIOS and try again. Do not attempt to boot your System if you know there are bugs in your SBIOS.

IV.4.3 Bootstrapping

IV.4.5.1 Loading a Bootstrap

If the SBIOS is not already in memory, it must be loaded (see the preceding section). The bootstrap must then be loaded into memory at either 8000 hex or D000 hex, depending on the machine's memory configuration (see Section IV.4.1). The bootstrap is recorded on the first 256 bytes of Track 0 of each Bootstrapping disk. Any available method may be used to load this code into the appropriate memory space. Possible methods include using a manufacturer-supplied operating system or a small assembly language program that calls the already-resident SBIOS to read the code. An example of such a program for each type of CPU is provided in Chapter V.

IV.4.3.2 Executing a Bootstrap

Each bootstrap requires that the parameters described in Section IV.4.2.3.1. be on the top of the processor stack (the number of disk drives to test must be zero). Once the configuration parameters are on the stack, and the SBIOS and bootstrap are n' memory, the bootstrap is ready to execute. This is done by executing a jump instruction to the beginning of the bootstrap code (i.e., either 8000 hex or D000 hex).

The bootstrapping process may take as long as two or three minutes. This is only for the time being: Section IV.4.4 explains how to write a faster bootstrap.
Note: On Z80/8080 Systems, the 'number of drives to test' parameter must equal 0 only if the supplied bootstrap is used: with a user-written bootstrap, it must not be pushed. On 6502 Systems, it should equal 0.

IV.4.5.5 Checking the System

Once the System appears to have bootstrapped, there are a few simple tests that help verify the interaction bet.ween the SBIOS and the p-System.
If the System has booted correctly, the console should display a welcome message, followed by the System version number, and the date on which the Bootstrap disk was created. After that, the outer System promptiine should appear (see the Users' Manual).

if these messages do not appear, the bootstrap has not worked. First check the values that are on the stack before the bootstrap is run. If these appear to be correct, the bootstrap code may not have functioned. If the bootstrap appears to be correct, then the SBIOS routines that handle either disk reads or console output

may be at fault.

Once the System appears to have booted, the next test is to type 'F'. This should call the Filer. Several sectors will be read off the System disk, and another promptline will appear. If these actions do not occur, the SBIOS disk read routines or console I/O routines may not work.

The final quick test is to type 'D' while in the Filer. When prompted, type the current date (e.g., 12-JAN-79) followed by <return>. Finally, type 'D' again. If the correct date (that is, the one you just typed) is not displayed, the disk write routines may be wrong.

IV.4.3.4 Accessing Other System Programs

The sole purpose of the Bootstrapping disk is to aid in the development of bootstraps. There is no need for most System programs in this process. Hence, they are provided on the System disk rather than on the Bootstrapping disk. The System disk as shipped contains three disk images, and may be unpacked as described in Section I.4. The first disk image, SYSTEM1, contains a p-System that may be booted once a working bootstrap and Interpreter are transferred to it. The other two disk images contain other System programs (Appendix C contains a full catalog).

Bootstraps may be transferred using the utility BOOTFR (see Section II.5). Interpreters may be transferred as any normal file, by using the Filer's T(ransfer command. Once it has a bootstrap, Interpreter, and Operating System, SYSTEM1 may be booted like the Bootstrap disk.

For any disk to be booted, it must contain a bootstrap, SYSTEM.INTERP, SYSTEM.PASCAL, SYSTEM.SYNTAX (if you intend to compile programs), SYSTEM.MISCINFO (if you intend to use the Screen Oriented Editor), and SYSTEM.LIBRARY (if you intend to use Long Integers or code that you yourself have placed in the Library).

In order to use an assembler, its name must be SYSTEM.ASSMBLER (no 'E'). Assemblers are shipped with the name of the processor they generate code for, so it is necessary to enter the Filer and use the C(hange command to name the desired assembler SYSTEM.ASSMBLER. Any assembler information files (e.g., Z80.0PCODES) must reside on the same disk as SYSTEM.ASSMBLER.

IV.4.3.5 Writing a Bootstrap

Chapter II discusses bootstraps in some detail. This section is only meant to provide some reminders about the same topics.

The Adaptable System comes with primary, secondary, and tertiary bootstraps. They are located on each Bootstrapping disk as diaqrammed in Figure 4 (Section II.5). The primary bootstrap that is supplied must be loaded by a bootstrap loader which you yourself supply; at some point in your use of the System you will almost certainly want to replace it with a simpler bootstrap (a "cold boot"). Section IV.4.1.2 describes how to do this.

Until you write your own primary bootstrap, you must load the supplied primary bootstrap with a bootstrap loader: this can be either a small program that you write yourself, or some other operating system that you are already using. The supplied primary bootstrap must be loaded into either 8000H or D000H, depending on your memory configuration, as described in Section IV.4.1. The processor's stack must also be loaded with parameters, as described in Section IV.5.2.5.1; the number of drives to test must be 0.

Once the primary bootstrap is loaded and the parameters are on the stack, execute the bootstrap by doing a JUMP to its location (either 8000H or D000H). Do not call the bootstrap as you would a subroutine.

The supplied primary bootstrap will load the SBIOS and the secondary bootstrap, push some parameters on the stuck, and initiate the secondary bootstrap. If you have prepared things correctly, within a short time you should have a running p-System.

IV.4.4 Improvements

When the p-System is first booted with a minimal SBIOS and the primary bootstrap supplied on the Bootstrapping disk, it does not have the speed or the extended I/O capabilities of a fully-implemented System. This section describes the improvements that can be made to your System, once you have jumped the initial hurdle of bootstrapping it for the first time.

Among the capabilities that you may add to your System are: a fast turnkey bootstrap (a "cold boot"), more efficient disk recording formats, the ability to communicate with a printer, serial line, and system clock, and the ability to use disk drives that are formatted differently from the System disk.

IV.4.4.1 Simple Improvements

This section discusses changing disk recording formats and writing a simpler bootstrap. Both of these improvements are relatively simple, and can substantially improve the speed of your System.

IV.4.4.1.1 Changing Disk Recording Formats

The Adaptable System disks as shipped are formatted with a sector interleaving of 1:1 and a sector track-to-track skew of 0 (interleaving and skew are described in Section IV.2.1). For most disk drives, these values are inefficient. The utility called FINDPARAMS can be used to determine more efficient parameters for your particular hardware.

There is a copy of FINDPARAMS on each Utilities disk. When you run it, it presents you with a series of prompts and tests which allow you to judge which combination of parameters works best with your disk drives. Once you know what these parameters are, you may use DISKCHANGE to alter the disks you use (DISKCHANGE is described in Section IV.2.1).

When you have used DISKCHANGE to alter your Bootstrapping disk, you must change the relevant parameters on the bootstrap stack:

```
interleaving factor
first Pascal track
track-to-track skew
```

... and then reboot your System. If you change the Bootstrapping disk and fail to alter these parameters, your System will no longer bootstrap. This is one reason it is important to keep backups of all your System disks.

Remember that you must use BOOTER to copy the bootstrap on Track 0.

Note that you cannot do this optimization until you have booted your System with a working SBIOS, as SBIOSTESTER requires that disks be formatted with 1-1 interleaving and 0 skew.

Note also that (at this point in your use of the System) all disks must have the same format; if you optimize your Bootstrapping disk, you must optimize all other disks that you use.

Warning: Before you use DISKCHANGE on a disk, you should back it up. DISKCHANGE may not do exactly what you wanted to. You may have forgotten to change the bootstrap parameters. Also, DISKCHANGE will lose all information on a disk that is not part of the logical disk it is altering: you must unpack your Adaptable System disks before you optimize them with DISKCHANGE. Section I.4 describes how to unpack Adaptable System disks.

Many soft-sectored 8" floppies in the field are formatted with 2-to-l interleaving, sector skew of 6, and first Pascal track of 1. This format is recommended if you wish to exchange software with other users. But DISKCHANGE can be used to convert p-System disks to any desired format.

IV.4.4.1.2 Simplifying the Bootstrap

In order to produce a turnkey p-System, (that is, one which boots as soon as you power your machine up, or perhaps power it up and then push a bootstrap button), your hardware must have some mechanism to read the contents of a pre-defined area of a disk into a pre-defined area in memory. On many machines, this mechanism takes the form of a boot-button that transfers control to a boot-ROM. It is the program in ROM that reads the contents of a disk sector into memory and causes that code to execute.

If your hardware has such a bootstrap feature, a primary bootstrap may be written. This primary bootstrap must push the appropriate configuration parameters onto the processor's stack, load the SBIOS into memory from a pre-defined location on the Bootstrapping disk, and then load the secondary bootstrap and start its execution.

The primary bootstrap which you write must reside on the Bootstrapping disk, along with the SBIOS. Neither of these may overwrite the System itself (which should not be surprising). The available areas on the Bootstrapping disk are:

1) Track 0: sectors 1 and 2

2) Track 0: sectors 19 through the end of track 0

3) Track 1: sectors 1 through 8

... this scheme assumes 1:1 interleaving. If you have already changed the interleaving of your disks, then the sectors available on Track 1 are the logical sectors 1..8, in other words, the areas on disk into which the first eight sectors of Track 1 are mapped (by the BIOS).

The primary bootstrap which you write must:

1) Read the SBIOS from the Bootstrapping disk into the memory space in which it is intended to execute. (Both the location on disk and the location in memory are determined by the user. See Section IV.4.1.1, and the preceding portion of this section.)

2) Load the secondary bootstrap from the Bootstrapping disk into the memory space in which it is intended to execute. The secondary bootstrap is 2048 bytes long, and is located on Track 0 starting at Sector 3 (see Figure 4). It must be loaded into memory following the primary bootstrap (i.e., it is loaded at either 8200 hex or D200 hex, depending on the primary bootstrap's location).

3) Load the configuration parameters onto the stack (see Section IV.4.2.3.1).

4) Do a JUMP (not a procedure call) to the beginning of the secondary bootstrap (which is at either 8200H or D200H).

... note that SETDISK must be called before the secondary bootstrap begins execution. SETDISK is usually called in Step (1) or Step (2), but if it is not, it must be called before Step (4).

IV. 4.4.1.2.1 Alternate Floppy Locations for the SBIOS

If there is not room for your SBIOS on the Bootstrapping disk, the p-System can be reconfigured to start on Track 2 instead of Track 1. This leaves Track 1 un-interleaved (i.e., 1-fco-l) and available for storage of the SBIOS. The reconfiguration procedure is;

1) Use DISKCHANGE to change the first Pascal track to 2. Do not alter the disk's interleaving or skew.

2) Change the 'first Pascal track' parameter on the booting stack to 2.

IV.4.4.1.2.2 Alternate Locations for the Secondary Bootstrap

If your hardware has a boot-ROM that must read a primary bootstrap from somewhere on Track 0, sectors 3 through 18, you must move the Secondary Bootstrap to a different location on the floppy. It may be moved to a different area of Track 0, or onto tracks 1 or 2. If it is moved to Track 1 or 2, the Bootstrapping disk must be reformatted as described in the preceding section. The primary bootstrap must be altered, so it will read the secondary bootstrap from its new location.

IV.4.4.2 Improving the SBIOS

In addition to a console and floppy disk drives, as handled by the simple SBIOS, the p-System may also interface to a remote port (serial line), a printer, a real it-time clock, floppy disks of dissimilar formats, and even devices whose interface is defined by the user. To obtain these facilities, you must create an Extended SBIOS with the appropriate drivers, and reconfigure your System's Interpreter.

IV.4.4.2.1 Communicating with the Interpreter

When the System calls the SBIOS routine SYSINIT, it passes a pointer to the Interpreter's jump vector (this is mentioned in Section IV.4.2.2.1). This allows the Extended SBIOS to do certain I/O operations that require handshaking with the Interpreter.

Exactly how this parameter is passed depends on your processor: see the appropriate section of Chapter V.

SBIOS routines call Interpreter routines in the same way SBIOS routines are called. This mechanism is described in Section IV.4.Z.2.5.

The Interpreter routines that the Extended SBIOS may need to use are:

Routine Name	Vector Number	Description
POLLUNITS	0	polls character-oriented devices
DSKCHNG	1	changes disk format values

Note: If you have an Extended SBIOS that uses these routines, and it is called by a machine-level program of your own (i.e., before the Interpreter has been bootstrapped), then these routines are, naturally, unavailable. In this case, you must pass the address of a "dummy" jump vector to SYSINIT: this dummy jump vector must point to "stubs" for the routines POLLUNIT and DSKCHNG. In other words, the program of yours which calls SYSINIT passes an address of a jump vector (which you have created), and this jump vector points to instructions which do nothing but return to their caller.

POLLUNITS

POLLUNITS may be called by DSKINIT, DSKREAD, and DSKWRIT.

POLLUNITS checks the console, remote, and printer input drivers for available data. Any available data is read from the appropriate device and saved in that device's input queue.

POLLUNIT5 does not alter any registers.

DSKCHNG

The Interpreter assumes that the disks it is communicating with are formatted according to the "current format": CURFORM. CURFORM is initialized by the secondary bootstrap according to the values it is passed on the processor stack.

This must be the format of the disk that bootstraps the System.
DSKCHNG changes CURFORM. It may be called by SETDISK, thus allowing the System to support multiple disk formats.

DSKCHNG is passed a pointer to a disk information record. This record contains six 16-bit words:

Word	Definition
0	number of tracks per disk
1	number of sectors per track
2	number of bytes per sector
5	interleaving factor
4	first Pascal track
5	track-to-track skew

These parameters should be familiar: they correspond to some of the parameters on the bootstrap stack. For details of passing the pointer to this record, see Chapter V.

DSKCHNG destroys all processor registers except the stack pointer.
When SETDISK is called and the new CURDISK has a different format from the previous CURDISK, SETDISK must make the appropriate call to DSKCHNG. It is the SETDISK routine that must know (usually by keeping a table) which disk number corresponds to which disk format.

IV.4.4.2.1.1 Enhancing the Floppy Disk Drivers

This section explains two improvements to the System based on the use of DSKCHNG and POLLUNIT5.

IV.4.4.2.1.1.1 Allowing Multiple Floppy Disk Formats

You may enable your System to support more than one disk format by rewriting SETDISK so that it calls DSKCHNG whenever CURDISK refers to a disk with a different format than the previous CURDISK (as explained above). SETDISK must know which disk drive has which format: this can be accomplished by a table lookup.

If any floppy drive has more sectors per track than the System disk (the bootstrapping disk), you must be careful to change the 'maximum sectors per track' parameter on the bootstrap stack to reflect the new situation.

IV.4.4.2.1.1.2 Polling During Disk Accesses

DSKINIT, DSKREAD, and DSKWRIT may take advantage of any wait loops (such as waiting for a SEEK to terminate) to use POLLUNITS to poll any character-oriented input devices.

The advantage of calling POLLUNITS frequently is that is ensures that each device's type-ahead queue is up to date. In particular, the user will be able to type ahead more commands more rapidly.

IV.4.4.2.2 The Extended SBIOS

Section IV.4.2 describes a simple SBIOS which includes only those routines that are absolutely necessary for booting the System. This section describes an additional thirteen SBIOS routines that communicate with a printer, a remote serial line, a hardware clock, and user-defined devices.

Routine Name	VECTOR NUMBER	DESCRIPTION
PRNINIT	15	printer initialize
PRNSTAT	16	printer status
PRNREAD	17	printer read
PRNWRIT	18	printer write
REMINIT	19	remote initialize
REMSTAT	20	remote status
REMREAD	21	remote read
REMWRIT	22	remote write
USRINIT	23	user devices initialize
USRSTAT	24	user devices status
USRREAD	25	user devices read
USRWRIT	26	user devices write
CLKREAD	27	system clock read

Note that in the jump table, these routines appear in this order, after the basic SBIOS routines.

The routines for a printer and remote port parallel those for the console: character-oriented devices are all handled in the same general manner, though internal details will differ for each device.

The user-defined device handlers (described below) are intended for peripheral hardware that the System does not customarily support (such as, for instance, grain elevators (yes, it has been done!)). A Pascal program may access these devices through the intrinsics UNITREAD, UNITWRITE, UNITCLEAR, and UNITSTATUS. See the Users' Manual for further information on these intrinsics. The device numbers 128..255 are available as numbers of user-defined devices.

IV.4.4.2.2.1 Additional SBIOS Routines

Each of the Extended SBIOS routines is described below. For information about parameter passing on a particular processor, see Chapter V.

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 offline, it should return a 9.

PRNINIT should not 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 online, the status must be 0, if the printer is offline, the status must be 9.

The second status is the state of the printer input channel (if there is one). If a character is pending on the printer input channel, PRNSTAT returns FF hex, otherwise it returns 0. (Note: PRNSTAT does not read the pending character, but merely reports its presence.)

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 online and a character is pending on the input channel, PRNREAD reads that character. If the printer is online 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 offline, 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 online, 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 offline, the status returned is 9.

PRNWRIT should not 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 will not 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 offline, 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 offline or there is no driver, the status is 9.

The second parameter returns FF hex if a character has been received on the remote channel, and 0 if no character has been received. (Note that REMSTAT does not 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 online and a character is pending, REMREAD reads that character. If the port is online 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 offline 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 online, the character is sent and the status is 0. If the remote port is offline or has no driver, the status is 9. If there is a transmission problem, the character is sent and the status is 1.

REMWRIT should not 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 will not 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 online and enabled, CLKREAD returns the time, and a status of 0. If the clock is offline, CLKREAD returns a status of 9, and sets the time equal to 0.

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

IV.4.4.2.2.1.1 User-defined Devices

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

The System may support a number of user-defined devices. Yet the Extended SBIOS has only one set of USRxxxx routines: USRINIT, USR5TAT, 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..255. Pascal programs may access user-defined devices by using the appropriate device numbers when calling the UNITREAD, UNITWRITE, ... family of intrinsics (see the Users' Manual, Chapter VI).

Note that 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 online (and all correct), and 9 for offline. 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 100..255 are available as user-definable status codes. The responsibility for handling these non-standard status codes belongs entirely to the user's software. If the System receives an I0RESULT in the range 100..255, it will halt with an I/O error (and then reboot) unless I/0 checking has been turned off (with the {$I-} compile-time option).

USRINIT

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

If the specified device is online, 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 online, USRINIT returns a status of 0. If the device is offline (or just plain nonexistent), USRINIT returns a status of 9. Other status codes may be defined by the user.

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 online, 9 for offline. Other status codes may be defined by the user.

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 the user.

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 the user's 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 the user.

USRREAD returns 0 for online, 9 for offline, 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 online, 9 for offline, or a user-defined status number.

IV.4.4.2.2.2 Testing the Extended SBIOS

Since the Extended SBIOS is intended to handle a wide variety of hardware, no automatic testing routines are provided. The user is responsible for testing Extended SBIOS routines and seeing that they work. It should be possible to test these routines either outside the p-System (using a different operating system) or within the p-System (the Users' Manual describes how to write load, and run assembly language routines).

IV.4.4.2.2.5 Bootstrapping with the Extended SBIOS

Before the System may be bootstrapped with an Extended SBIOS, the Interpreter must be "reconfigured." This operation is described in Chapter V. Once the Interpreter has been reconfigured, your normal bootstrapping procedure may be followed, substituting the new Extended SBIOS for the original simple SBIOS, and making any necessary changes to the parameters on the bootstrap stack.

V. MACHINE-SPECIFIC NOTES

V.1 Z80 and 8080 Systems

V.1.1 Vector Lists and Register Assignments

SBIOS routines must return their status (I0RESULT) in the A register. Parameters are passed to SBIOS routines in the B and C registers. The read routines write into a buffer in main memory. The stack pointer should not be modified (except as necessary to return from each routine in a standard manner). The following table shows the parameters for each routine in the basic SBIOS, along with each routine's vector offset (i.e., the position in the jump table of the instruction that jumps to that routine):

(The vector offsets are shown in hex.)

Routine	Vector Offset	Parameters
SYSINIT	00	passed: BC = pointer to Interpreter' s jump table
SYSHALT	03	<none>
CONINIT	06	returns: A = IORESULT
CONSTAT	09	returns: A = IORESULT C = 0 if no char pending = FF if char pending
CONREAD	0C	returns: A = IORESULT C = input char
CONWRIT	0F	passed: C = output char returns: A = IORESULT
SETDISK	12	passed: C = disk no. (CURDISK)
SETTRAK	15	passed: C = track no. (CURTRAK)
SETSECT	18	passed: C = sector no. (CURSECT)
SETBUFR	1B	passed: BC = buffer addr. (CURBUFR)
DSKREAD	1E	returns: A = IORESULT
DSKWRIT	21	returns: A = IORESULT
DSKINIT	24	returns: A = IORESULT
DSKSTRT	27	<none>
DSKSTOP	2A	<none>

Some Extended SBIOS routines are passed parameters on top of the stack. The 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 ‘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.

The following table continues the above table, showing parameters for routines in the Extended SBIOS:

```
PRNINIT        2D          returns: A = IORESULT
PRNSTAT        30          returns: A = IORESULT
                                    C = 0 if no char pending
                                      = FF if char pending
PRNREAD        33          returns: A = IORESULT
                                    C = input char
PRNWRIT        36          passed:  C = output char
                           returns: A = IORESULT
REMINIT        39          returns: A = IORESULT
REMSTAT        3C          returns: A = IORESULT
                                    C = 0 if no char pending
                                      = FF if char pending
REMREAD        3F          returns: A = IORESULT
                                    C = input char
REMWRIT        42          passed:  C = output char
                           returns: A = IORESULT
USRINIT        45          passed:  C = device number
                           Returns: A = IORESULT
USRSTAT        48          passed: SP = return address
                                        input/output toggle
                                        pointer to status rec
                                        device number
                           returns: A = IORESULT
USRREAD        4B          passed: SP = return address
                                        extra parameter 2
                                        extra parameter 1
                                        pointer to buffer
                                        device number
                                        extra parameter 5
                           returns: A = IORESULT
USRWRIT        4E          passed: SP = return address
                                        extra parameter 2
                                        extra parameter 1
                                        pointer to buffer
                                        device number
                                        extra parameter 5
                           returns: A = IORESULT
CLKREAD        51          returns: A = IORESULT
                                   DE = least significant word
                                   HL = most significant word
```

The following table shows offsets and parameters for the two Interpreter routines which SBIOS routines may access:

POLLUNITS	0	<none>
DSKCHNG	3	passed: BC = pointer to disk format values

V.1.2 Sample Bootstrap Loader

```
; This routine loads the primary bootstrap from SBOOT8.
; The primary bootstrap is located at Track 0, sectors 1 and 2.
; The SBIOS must be resident before this program may be executed:
; SBIOS routine s are used to read the bootstrap from the disk.
; If there is any problem, SYSHALT is called.
; Note the use of the SBIOS jump table: the formula used corresponds to the
; instructions in Section IV.4.2.2.5; a jump instruction is 5 bytes long.

        .PROC   LOAD

BIOSJP  .EQU      0FD00H      ; we assume the SBIOS is at this location
BOOTAD  .EQU      8000H       ; we are using SBOOT8 (not SBOOTD)
SECSIZE .EQU      80H         ; number of bytes in a sector

SYSINIT .EQU      00H         ; these are SBIOS jump table offsets
SYSHALT .EQU      03H
SETDISK .EQU      12H
SETTRAK .EQU      15H
SETSECT .EQU      18H
SETBUFR .EQU      1BH
DSKREAD .EQU      1EH
DSKINIT .EQU      24H
DSKSTRT .EQU      27H
DSKSTOP .EQU      2AH

      .MACRO      SBIOS       ; calls an SBIOS routine
      CALL        BIOSJP + %1
      .ENCM

LOADR                         ; the code to load the bootstrap
      SBIOS       SYSINIT     ; initialize the SBIOS
      LD          C,0         ; the bootstrap disk is drive 0
      SBIOS       SETDISK
      SBIOS       DSKSTRT
      SBIOS       DSKINIT     ; now ready to use disk (if no error)
      AND         A           ; check for I0 error
      JP          NZ,CALLHLT  ; ... halt system if problem
      LD          C,0         ; bootstrap is in Track 0
      SBIOS       SETTRAK

      LD          BC,BOOTAD   ; memory buffer is bootstrap location
      SBIOS       SETBUER
      LD          C,1         ; first read Sector 1
      SBIOS       SETSECT
```

```
      SBIOS       DSKREAD
      AND         A           ; check for I/0 error
      JP          NZ,CALLHLT  ; ... halt system if problem
      LD          BC,BOOTAD+SECSIZE ; prepare to read rest of bootstrap
      SBIOS       SETBUFR
      LD          C,2         ; ... which is in Sector 2
      SBIOS       SETSECT
      SBIOS       DISKREAD
      AND         A           ; check for I/0 error
      JP          NZ,CALLHLT  ; ... halt if problem
      SBIOS       DSKSTOP     ; we're through with the disk
      RET                     ; return to caller

; now the program that calls LOADR must set up the parameter stack
; and then jump to the bootstrap, which is at 8000H

CALLHLT                       ; the error routine
      SBIOS       SYSHALT     ; stops the system
      JP          CALLHLT     ; if SYSHALT fails,
                              ; don't go elsewhere!
      .END
```

V.1.3 Memory Configuration Notes

All parameters which are memory addresses must be word quantities, and the low byte of the address must be even (for example, the highest word in memory is FFFE hex; the highest byte is FFFF).

The Interpreter must start on a page boundary. This means that the low byte of its starting address must be 00. If you wish the Interpreter to be located at the start of the large contiguous RAM space, then the large RAM space must start on a page boundary.

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

To push bootstrap parameters onto the processor stack, set the stack pointer to the highest even address in the large contiguous RAM space, and then push the parameters (the stack grows downward).

V.1.4 Reconfiguring the Interpreter

The Interpreter disk in each Adaptable System contains codefiles which may be linked together to form an Interpreter configured differently than the SYSTEM.INTERP that is shipped already linked. When you create an Extended SBIOS, you must reconfigure the Interpreter by choosing the appropriate codefiles and linking them together yourself.

These are the relevant files:

Name	Description
INTERP. CODE	Interpreter with no real numbers
INTERP.FP.CODE	Interpreter with real number operations (FP stands for Floating Point)
RSP.CODE	interface between Interpreter and BIOS
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
INTER.CODE	SBIOS interface
INTER.X.CODE	Extended SBIOS interface
TERTBOOT.CODE	tertiary bootstrap

The SYSTEM.INTERP that is shipped is INTERP.CODE linked with RSP.CODE, BIOS.CODE, INTER.CODE, and TERTBOOT.CODE.

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

1) Link the codefile.

You must make the following choices:

Whether to use INTERP or INTERP.FP. INTERP.FP allows your programs to use real numbers and transcendental functions, but it is much larger than INTERP.

Note: If your system has a hardware clock, and you are using it (i.e., the HAS CLOCK data item in SYSTEM.MiSCINFO must be set to TRUE using SETUP), then you must use INTERP.FP. The reason is that if there is a hardware clock, the Compiler uses it to calculate compile times, and uses real arithmetic to do so.

Whether to use BIOS, BIOS.C, BIOS.CR, or BIOS.CRP. These are

progressively larger BIOS'es. Queuing allows more efficient I/0. Use the BIOS that most closely matches your hardware configuration. BIOS.CR and BIOS.CRP can only be used with an Extended SBIOS.

Whether to use INTER or INTER.X. This depends on which SBIOS you are using.

Once you know what the pieces of your new Interpreter will be, you can link them together with the System's Linker. The Interpreter codefile you choose will always be the 'Host file?', and the remaining codefiles will be entered as 'Lib file?'s, always in the following-order;

RSP
the BIOS you have chosen
the SBIOS interface you have chosen
TERTBOOT

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

Example:

At the System command level, type 'L' for L(ink. The following prompts appear (<return> means the carriage return key, and comments are in {}):

```
Host file? lNTERP<return>   {or INTERP.FP}
Lib file? RSP<return>
Opening RSP.CODE
Lib file? BIOS.CRP<return>   {or other BIOS}
Opening BIOS.CRP.CODE
Lib file? INTER.X<return>   {or simply INTER}
Opening INTER.X.CODE
Lib file? TERTBOOT<return>
Opening TERTBOOT.CODE
Lib file? <return>

... {more Linker output}

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

2) Compress the codefile.

At the System command level, type 'X' for eX(ecute, then 'COMPRESSOR<return>'. the utility COMPRESSOR shows a series of

prompts; answer them as follows:

Assembly Code File Compressor

Type '!' 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 COMPRESSOR will either complete its work, or issue an error message, in which case you must try again.

(COMPRESSOR is described in the Users' Manual, Section X.1.)

3) Change filenames

At the System command level, type T' 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.

V.1.5 Miscellaneous Notes

When booting the System, the 'number of drives to test' parameter must be 0 only if you use the primary bootstrap that is shipped with the System. If you write your own primary bootstrap, this parameter must not be pushed onto the processor stack.

V.2 PDP-11 and LSI-11 Systems

V.2.1 Vector Lists and Register Assignments

PDP-11 and LSI-11 p-Systems are ready to run as shipped. There is no BIOS or SBIOS per se; I/0 routines are embedded in the Interpreter. This is feasible because of the consistency of I/0 handling in '11' systems.

Addresses and interrupt vectors are assigned to the standard p-System devices as follows:

Device	Address	Interrupt Vector
CONSOLE:	177560	060
KEYBOARD:	"	"
PRINTER:	177510	200 (parallel)
	"	204 (serial)
REMOUT:	177520	120
REMIN;	"	"

... the numbers are octal (base eight).

For more information about low-level device handling on 11's, it is best to refer to the hardware documentation.

V.2.2 Sample Bootstrap Loader

All current PDP-11 and LSl-il Systems are shipped as ready-to-run software packages. There is no need to rewrite the bootstrap that is shipped, nor to write any special-purpose program to load the bootstrap.

On 11's, the utility BOOTER copies the first two blocks of the disk.

V.2.3 Memory Configuration Notes

Not applicable.

V.2.4 Reconfiguring the Interpreter

Not applicable.

Note that PDP-11/LSI-11 Systems come with several different interpreters. Each interpreter is intended for a particular set of disk devices. Interpreters that use the hardware extended instruction set (EIS) have '.EIS' in their filename.

Thus, each interpreter is named either PDP... or LSI..., where ... are the mnemonics for features supported by that particular interpreter.

Supported disk drives are indicated in the names of '11' interpreters by the following mnemonics:

RX	floppy disks (RX-0l's)
DY	double density floppy disks (RX-02's)
RK	RK-05 hard disks
RL	RL-01 hard disks

For examples of interpreter names, refer to Appendix B.

V.2.5 Miscellaneous Notes

The utilities RT11TOEDIT and EDITTORT11 are provided for converting RT-11 files into p-System textfiles or visa versa.

RT11TOEDIT first prompts the user for a device number. This should be the number of a disk drive that contains an RT-11-format disk, with a directory in blocks 6..7. If the disk is present, its directory is displayed on the screen, showing each file with its name, type, size in blocks, and position on the disk (a block number in base ten). Unused portions of the disk are also shown. The user is then prompted for the name of an RT-11 file and the name of a p-System file to which it will be written. The user may specify a bitwise transfer (i.e., the file is unchanged), or a textfile transfer (the new file is supplied with a standard textfile header block).

EDITTORT11 is similar to RT11TOEDIT. The user must specify the number of a disk drive with an RT-11-format disk in it, then name a p-System file to be transferred, and an RT-11 file to be created.

With both of these utilities, all prompts must be answered with upper case characters only.

V.3 6502 Systems

V.3.1 Vector Lists and Register Assignments

The System assumes that the SBIOS destroys 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 read routines write into a buffer in main memory. The stack pointer should not be modified (except as necessary to return from each routine in a standard manner).

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

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: A = 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
DSKSTRT	27	<none>
DSKSTOP	2A	<none>

Some Extended SBIOS routines are passed parameters on top of the stack. The 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.

The following table continues the above table, showing parameters for routines in the Extended SBIOS:

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
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

The following table shows offsets and parameters for the two Interpreter routines which SBIOS routines may access:

POLLUNITS	0	<none>
DSKCHNG	3	passed: XA = pointer to disk format values

V.5.2 Sample Bootstrap Loader

```
; This routine loads the primary bootstrap from SBOOT8.
; The primary bootstrap is located at Track 0, sectors 1 and 2.
; The SBIOS must be resident before this program may be executed:
; SBIOS routines are used to read the bootstrap from the disk.
; If there is any problem, SYSHALT is called.
; Note the use of the SBIOS jump table: the formula used corresponds to the
; instructions in Section IV.4.2.2.5; a jump instruction is 3 bytes long.

        .PROC   LOAD

BIOSJP  .EQU    0FD00H          ; we assume the SBIOS is at this location
BOOTADR .EQU    80H             ; we are using SBOOT8 (not SBOOTD)
                                ; (this is the high byte of the address)
SECSIZE .EQU    80H             ; number of bytes in a sector

SYSINIT .EQU    OOH             ; these are SBIOS jump table offsets
SYSHALT .EQU    03H
SETDISK .EQU    12H
SETTRAK .EQU    15H
SETSECT .EQU    18H
SETBUFR .EQU    1BH
DSKREAD .EQU    1EH
DSKINIT .EQU    24H
DSKSTRT .EQU    27H
DSKSTOP .EQU    2AH

        .MACRO  SBIOS           ; calls an SBIOS routine
        JSR     BIOSJP + %1
        .ENDM

LOADR                           ; the code to load the bootstrap
        SBIOS   SYSINIT         ; initialize the SBIOS
        LDA     #0              ; the bootstrap disk is drive 0
        SBIOS   SETDISK
        SBIOS   DSKSTRT
        SBIOS   DSKINIT         ; now ready to use disk (if no error)
        TXA                     ; check for I/O error
        BNE     CALLHLT         ; ... halt system if problem
        LDA     #0              ; bootstrap is in Track 0
        SBIOS   SETTRAK

        LDA     #BOOTADR        ; memory buffer is bootstrap location
        SBIOS   SETBUFR
        LDA     #1              ; first read Sector 1
```

```
        SBIOS   SETSECT
        SBIOS   DSKREAD
        TXA                     ; check for I/O error
        BNE     CALLHLT         ; ... halt system if problem
                                ; prepare to read rest of bootstrap
        LDA     #SECSIZE % 100H             ; this is the low byte ...
        LDX     #BOOTADR + <SECSIZE / 100H> ; and the high byte
                                ; ... of the bootstrap address
        SBIOS   SETBUFR
        LDA     #2              ; rest of bootstrap is in Sector 2
        SBIOS   SETSECT
        SBIOS   DISKREAD
        TXA                     ; check for I/O error
        BNE     CALLHLT         ; ... halt if problem

        SBIOS   DSKSTOP         ; we're through with the disk
        RET                     ; return to caller

; now the program that calls LOADR must set up the parameter stack
; and then jump to the bootstrap, which is at 8000H

CALLHLT                         ; the error routine
        SBIOS   SYSHALT         ; stops the system
        JMP     CALLHLT         ; if SYSHALT fails,
                                ; don't go elsewhere!
        .END
```

V.5.5 Memory Configuration Notes

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

Pages 0 and 1 of main memory (addresses 0000H through 01FFH) are used for data storage by the Interpreter. The SBIOS, Interpreter, and p-System software may not occupy these pages.

There are two bootstrap disks for the 6502: LO PAGE and HI PAGE. The System on LO PAGE assumes that it has exclusive use of Page 0 addresses 0000H through 007FH. The System on HI PAGE assumes exclusive use of 0080H through 00FFH. Which of these systems you choose depends on whether you have other (hardware or software) requirements for one of these halves of Page 0; if you do not, choose either bootstrap.

The Interpreter must start on a page boundary (the low byte of the address must be 00). If the Interpreter is to be located at the start of the large contiguous RAM space, the large RAM space must start on a page boundary.

The SBIOS may use any interrupt vectors it needs without fear of conflicting with the p-System. However, the System disables interrupts when it would be unsafe to get an interrupt on an attached semaphore.

The stack pointer must be initialized to 00FFH before bootstrap parameters are pushed onto the stack.

V.3.4 Reconfiguring the Interpreter

Reconfiguring the 6502 Interpreter is equivalent to reconfiguring the Z80/8080 Interpreter. Refer to Section V.1.4.

V.3.5 Miscellaneous Notes

When the System is bootstrapped, the 'number of drives to test' parameter on the processor stack must be 0.

6502 Systems use an expression stack that is limited to 128 words of data. When this stack overflows, the System is halted and re-initialized (though it may simply hang). 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.

2) Nested set expressions must be written so that there are never more than 5 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(C+D)+B)+A

V.4 9900 Systems

V.4.1 Vector Lists and Register Assignments

All current 9900 Systems are shipped as ready-to-run software. There is therefore no need for the user to alter the SBIOS or write a bootstrap.

V.4.2 Sample Bootstrap Loader

Not applicable.

V.4.3 Memory Configuration Notes

Not applicable.

V.4.4 Reconfiguring the Interpreter

Not applicable.

V.4.5 Miscellaneous Notes

None.

VI.A APPENDIX A -- A. S. Hardware Requirements

Memory:

At least 48K of RAM memory, of which at least 36K must be in one contiguous block. There must be room for the bootstraps (6144 = 1800H bytes) at either 8000 hex or D000 hex. The Interpreter must start at a page boundary (one page is FF hex bytes).

For CP/M Adaptable Systems, the entire 48K must be contiguous.

For 6502 systems, at least one-half of Page 0 must be unoccupied and contiguous (either 00-7F hex or 80-FF hex).

Disk Drive:

At least one floppy disk drive with at least 175K bytes of available space. Either 8" or 5-1/4" floppy drives may be used, and they may be of any format (soft- sectored or hard-sectored, etc.).

Console:

A console that sends and receives ASCII characters. Either a teletype or a CRT may be used. If a CRT is used, it must be able to scroll one line at a time.

Downloading:

If your system floppies are other than 8" IBM 3740-format soft-sectored single-density single-sided disks, you must have access to some facility that can download the disks that are shipped to disks of your own format.

VI.B APPENDIX B — Disk Catalog for Current Releases

Note: This shows the catalog for Z80/8080 Systems only. More information will appear in future printings of this Guide.

ADAPZ

```
Z8SYS:
SYSTEM.INTERP     26  7-Jan-81     6   512 Datafile
SYSTEM.MISCINFO    1  7-Jan-81    32   194 Datafile
SYSTEM.FILER      32 26-Jan-81    33   512 Codefile
< UNUSED >         2              65
SYSTEM.PASCAL     85  4-Feb-81    67   512 Datafile
< UNUSED >         1             152
4/4 files<listed/in-dir>, 150 blocks used, 3 unused, 2 in largest



ZDSYS:
SYSTEM.INTERP     26  7-Jan-81     6   512 Datafile
SYSTEM.MISCINFO    1  7-Jan-81    32   194 Datafile
SYSTEM.FILER      32 26-Jan-81    33   512 Codefile
< UNUSED >         2              65
SYSTEM.PASCAL     85  4-Feb-81    67   512 Datafile
< UNUSED >         1             152
4/4 files<listed/in-dir>, 150 blocks used, 3 unused, 2 in largest



INTZ80:
SYSTEM.LIBRARY     9  7-Jan-81     6   512 Datafile
INTERP.Z.CODE     22  7-Jan-81    15   512 Codefile
INTERP.ZF.CODE    25  7-Jan-81    37   512 Codefile
TERTBOOT.CODE      7  7-Jan-81    62   512 Codefile
RSP.CODE           6  7-Jan-81    69   512 Codefile
BIOS.CODE          8  7-Jan-81    75   512 Codefile
BIOS.C.CODE        8  7-Jan-81    83   512 Codefile
BIOS.CR.CODE       8  7-Jan-81    91   512 Codefile
BIOS.CRP.CODE      9  7-Jan-81    99   512 Codefile
INTER.CODE         4  7-Jan-81   108   512 Codefile
INTER.X.CODE       4  7-Jan-81   112   512 Codefile
< UNUSED >        37             116
11/11 files<listed/in-dir>, 116 blocks used, 37 unused, 37 in largest
```

ADAP8

```
88SYS:
SYSTEM.INTERP     27  7-Jan-81     6   512 Datafile
SYSTEM.MISCINFO    1  7-Jan-81    33   194 Datafile
SYSTEM.FILER      32 26-Jan-81    34   512 Codefile
< UNUSED >         2              66
SYSTEM.PASCAL     85  4-Feb-81    68   512 Datafile
4/4 files<listed/in-dir>, 151 blocks used, 2 unused, 2 in largest



8DSYS:
SYSTEM.INTERP     27  7-Jan-81     6   512 Datafile
SYSTEM.MISCINFO    1  7-Jan-81    33   194 Datafile
SYSTEM.FILER      32 26-Jan-81    34   512 Codefile
< UNUSED >         2              66
SYSTEM.PASCAL     85  4-Feb-81    68   512 Datafile
4/4 files<listed/in-dir>, 151 blocks used, 2 unused, 2 in largest



INT8080:
SYSTEM.LIBRARY     9  7-Jan-81     6   512 Datafile
INTERP.8.CODE     22  7-Jan-81    15   512 Codefile
INTERP.8F.CODE    25  7-Jan-81    37   512 Codefile
TERTBOOT.CODE      7  7-Jan-81    62   512 Codefile
RSP.CODE           6  7-Jan-81    69   512 Codefile
BIOS.CODE          8  7-Jan-81    75   512 Codefile
BIOS.C.CODE        8  7-Jan-81    83   512 Codefile
BIOS.CR.CODE       8  7-Jan-81    91   512 Codefile
BIOS.CRP.CODE      9  7-Jan-81    99   512 Codefile
INTER.CODE         4  7-Jan-81   108   512 Codefile
INTER.X.CODE       4  7-Jan-81   112   512 Codefile
< UNUSED >        37              116
11/11 files<listed/in-dir>, 116 blocks used, 37 unused, 37 in largest
```

CPMADAP

```
SYSCPM1:
SYSTEM.INTERP     27  7-Jan-81     6   512 Datafile
SYSTEM.MISCINFO    1  7-Jan-81    33   194 Datafile
SYSTEM.FILER      32 26-Jan-S1    34   512 Codefile
< UNUSED >         2              66
SYSTEM.PASCAL     85  4-Feb-81    68   512 Datafile
4/4 files<listed/in-dir>, 151 blocks used, 2 unused, 2 in largest


88SYS:
SYSTEM.INTERP     27  7-Jan-81     6   512 Datafile
SYSTEM.MISCINFO    1  7-Jan-81    33   194 Datafile
SYSTEM.FILER      32 26-Jan-81    34   512 Codefile
< UNUSED >         2              66
SYSTEM.PASCAL     85  4-Feb-81    68   512 Datafile
4/4 files<listed/in-dir>, 151 blocks used, 2 unused, 2 in largest


INTCPM:
SYSTEM.LIBRARY     9  7-Jan-81     6   512 Datafile
INTERP.8.CODE     22  7-Jan-81    15   512 Codefile
INTERP.8F.CODE    25  7-Jan-81    37   512 Codefile
TERTBOOT.CODE      7  7-Jan-81    62   512 Codefile
RSP.CODE           6  7-Jan-81    69   512 Codefile
BIOS.CODE          8  7-Jan-81    75   512 Codefile
BIOS.C.CODE        8  7-Jan-81    83   512 Codefile
BIOS.CR.CODE       8  7-Jan-81    91   512 Codefile
BIOS.CRP.CODE      9  7-Jan-81    99   512 Codefile
INTER.CODE         4  7-Jan-81   108   512 Codefile
INTER.X.CODE       4  7-Jan-81   112   512 Codefile
INTER.CPM4.CODE    4  7-Jan-81   116   512 Codefile
INTER.CPM1.CODE    4  7-Jan-81   120   512 Codefile
INTER.CPM2.CODE    4  7-Jan-81   124   512 Codefile
< UNUSED >        25              128
14/14 files<listed/in-dir>, 128 blocks used, 25 unused, 25 in largest
```

UTILITIES

```
UTIL1:
BOOTER.CODE         3  4-Dec-80     6   512 Codefile
DISKCHANGE.CODE     8  5-Dec-80     9   512 Codefile
DISKSIZE.CODE       3  3-Dec-80    17   512 Codefile
PINDPARAMS.CODE     9  3-Dec-80    20   512 Codefile
YALOE.CODE         12  2-Dec-80    29   512 Codefile
LIBRARY.CODE       13 23-Jan-81    41   512 Codefile
SAMPLEGOTO.TEXT     4 17-Nov-78    54   512 Textfile
PATCH.CODE         33  3-Dec-80    58   512 Codefile
DECODE.CODE        29  3-Dec-80    91   512 Codefile
COPYDUPDIR.CODE     3  2-Dec-80   120   512 Codefile
MARKDUPDIR.CODE     4  2-Dec-80   123   512 Codefile
< UNUSED >         26             127
11/11 files<listed/in-dir>, 127 blocks used, 26 unused, 26 in largest


UTIL2:
COMPRESS.CODE      10  3-Dec-80     6   512 Codefile
XREF.CODE          29  3-Dec-80    16   512 Codefile
RECOVER.G.CODE      8  5-Dec-80    45   512 Codefile
CPMBOOT.CODE       22  7-Jan-81    53   512 Codefile
KERNEL.CODE        63  2-Feb-81    75   512 Codefile
COMMANDIO.CODE      9  5-Jan-81   138   512 Codefile
< UNUSED >          6             147
6/6 files<listed/in-dir>, 147 blocks used, 6 unused, 6 in largest
```

SYSTEM

```
SYS1:
SYSTEM.SYNTAX     14  4-Dec-80     6   512 Datafile
SETUP.CODE        27  1-Dec-80    20   512 Codefile
SYSTEM.COMPILER   94  7-Feb-81    47   512 Codefile
< UNUSED >        12             141
3/3 files<listed/in-dir>, 141 blocks used, 12 unused, 12 in largest


SYS2:
Z80.ASSMBLER      51  2-Dec-80     6   512 Codefile
Z80.OPCODES        3 20-Dec-78    57    68 Datafile
Z80.ERRORS         8 23-Sep-80    60    70 Datafile
SYSTEM.LINKER     26 27-Jan-81    68   512 Codefile
DEBUGGER.CODE     21  4-Feb-81    94   512 Codefile
< UNUSED >        38             115
5/5 files<listed/in-dir>, 115 blocks used, 38 unused, 38 in largest


SYS3:
8080.ASSMBLER     47  2-Dec-80     6   512 Codefile
8080.OPCODES       3 25-Mar-80    53    44 Datafile
8080.ERRORS        8 23-Sep-80    56    70 Datafile
SYSTEM.EDITOR     49 30-Jan-81    64   512 Codefile
< UNUSED >        40             113
4/4 files<listed/in-dir>, 113 blocks used, 40 unused, 40 in largest
```

CPMDISK (BOOTER)

CP/M Directory

PASBOOT.BAK
PASBOOT.PRN
PASBOOT.ASM
PASBOOT.HEX
PASBOOT.COM
SAMBOOT.ASM

SYSCPM2:
SYSTEM.INTERP 27 7-Jan-81 6 512 Dafcafile
SYSTEM.MISCINFO 1 7-Jan-81 33 194 Datafile
SYSTEM.FILER 32 26-Jan-Sl 34 512 Codefile
< UNUSED > 2 66
SYSTEM.PASCAL 85 4-Feb-81 68 512 Datafile
4/4 files<listed/in-dir>, 151 blocks used, 2 unused, 2 in largest

Vl.C APPENDIX C — Troubleshooting

Refer to this Appendix if you have problems with your p-System. It attempts to point out a number of errors that are commonly encountered. Look up the section that applies to your problem: if you cannot answer a question affirmatively, you may have found the source of your troubles: go back to the body of this Guide, and review the subject in question. If the information you find there does not enable you to solve the problem, contact the supplier of your p-System for support.

System will not Bootstrap

Adaptable Systems

Does track 0 contain a bootstrap?

Are you using SBOOT8 in conjunction with the large contiguous RAM starting before 5000H?

Are you using SBOOTD in conjunction with the large contiguous RAM starting on or after 3000H?

Is the jump table for your SBIOS correct?

Are the bootstrap parameters loaded exactly as described in the text before you jump to the bootstrap?

Did your SBIOS pass all SBIOSTESTER tests?

Was the 'number of drives to test' parameter equal to 5?

PDP/LSI - 11

Is memory management OFF on your LSI-11/23?

Is your VT-l00 in VT52 mode with X-on X-off checking disabled?

Is any ROM located above 64K disabled?

CP/M Adaptable System

Is your sector size 128?

Are you using a standard CP/M 1.4, 2.0, or 2.2 CBIOS?

Is your CDOS version compatible with version 1.07?

Do you have a double density IMS 8000?
If so, you must
  0. Back up your disks.
  1. Boot a double density CP/M 2.2 disk.
  2. Insert the CP/M Booter release disk in drive B and use the
     IMSGEN utility to copy the CP/M bootstrap to this disk.
  3. Insert the CP/M Booter disk in drive A and
     type control-C to warm-boot CP/M.
  4. Run PASBOOT as described in Section IV.3.

Do you have CP/M configured for Dynabyte disk drives?
  This version has been known to be incompatible with
  the CP/M Adaptable System. You must use the full
  Adaptable System.

H-89

If you have only one disk drive you must connect pin 12 to
pin 26 on your cable to the disk drives.

After Bootstrapping, the System Crashes

Do you have a minimum of 48K bytes of memory?

Have you run memory diagnostics recently?

The Printer Doesn't Work

On a PDP/LSI-11

Is your device set to address 177514 or 177510, trap vector 200
When trying to use PRINTER: (device #6)?

Is your device set to address 177520, trap vector 120 when
trying to use REMIN: or REMOUT: (devices #/7 and 8)?

Do you have a DLV-11J?
  The console channel is on channel 0 and should be at the
  standard address 177560. The base address on channel 3
  should be 177500. This will cause channel 1 to be PRINTER:,
  and channel 2 to be REMIN: and REMOUT:. The actual wiring

required is: A9 jumpered x to 1.

On any System

Is there a printer driver linked in your Interpreter?

Does your printer expect any special protocols?
You may need to write an external procedure to handle the protocol.

Does your printer hang/crash when processing NULS?
The System sends NULs as pads in blocks of text.
You may need to write a pre-processor which sends the printer text without the NULs.

You Get an Unimplemented Instruction Error

Are you attempting to use floating point without having linked or renamed a floating point interpreter to be your System Interpreter?

You can't read the disks

Do you have an INTEL MDS?
If you do, you must turn off CRC checking, then make image copies of your disks. You can now enable CRC checking and use the image copies of the release disks.

Screentest Reports Errors

The EDITOR ESCAPE KEY and the KEY TO DELETE CHARACTER are occasionally reported as non-functional, even though they are working properly. If the keys function when you use the Screen Oriented Editor, you may ignore these error messages.

You are C(ompiling or A(ssembling

Syntax Error 'Unexpected end of Input' is encountered...

Did you use the C(ompile or A(ssemble command?
You **cannot** eX(ecute SYSTEM.COMPILER or

SYSTEM.ASSMBLER.

VI.D APPENDIX D -- ASCII

0	000	00	NUL	52	040	20	SP	64	100	40	@	96	140	60	
1	001	01	SOH	53	041	21	!	65	101	41	A	97	141	61	a
2	002	02	STX	54	042	22	"	66	102	42	B	98	142	62	b
5	005	05	ETX	55	045	25	#	67	105	45	C	99	145	65	c
4	004	04\	EOT	56	044	24	$	68	104	44	D	100	144	64	d
5	005	05	ENQ	57	045	25	%	69	105	45	E	101	145	65	e
6	006	06	ACK	58	046	26	&	70	106	46	F	102	146	66	f
7	007	07	BEL	59	047	27	'	71	107	47	G	105	147	67	g
8	010	08	BS	40	050	28	(	72	110	48	H	104	150	68	h
9	011	09	HT	41	051	29	)	75	111	49	I	105	151	69	i
10	012	0A	LF	42	052	2A	*	74	112	4A	J	106	152	6A	j
11	015	0B	VT	45	055	2B	+	75	115	4B	K	107	155	6B	k
12	014	0C	FF	44	054	2C	,	76	114	4C	L	108	154	6C	l
15	015	0D	CR	45	055	2D	-	77	115	4D	M	109	155	6D	m
14	016	0E	SO	46	056	2E	.	78	116	4E	N	110	156	6E	n
15	017	0F	S1	47	057	2F	/	79	117	4F	0	111	157	6F	o
16	020	10	DLE	48	060	50	0	80	120	50	P	112	160	70	p
17	021	11	DC1	49	061	51	1	81	121	51	Q	115	161	71	q
18	022	12	DC2	50	062	52	2	82	122	52	R	114	162	72	r
19	025	15	DC5	51	065	55	5	85	125	55	S	115	165	75	s
20	024	14	DC4	52	064	54	4	84	124	54	T	116	164	74	t
21	025	15	NAK	55	065	55	5	85	125	55	U	117	165	75	u
22	026	16	SYN	54	066	56	6	86	126	56	V	118	166	76	v
25	027	17	ETB	55	067	57	7	87	127	57	W	119	167	77	w
24	050	18	CAN	56	070	58	8	88	150	58	X	120	170	78	x
25	051	19	EM	57	071	59	9	89	151	59	Y	121	171	79	y
26	052	1A	SUB	58	072	5A	:	90	152	5A	Z	122	172	7A	z
27	055	1B	ESC	59	075	5B	;	91	155	5B	[	125	175	7B	{
28	054	1C	FS	60	074	5C	<	92	154	5C	\	124	174	7C	|
29	055	ID	GS	61	075	5D	=	95	155	5D	]	125	175	7D	}
50	056	IE	RS	62	076	5E	>	94	156	5E	^	126	176	7E	~
51	057	IF	US	65	077	5F	?	95	157	5F	_	127	177	7F	DEL