

# System 88 User's Manual

PolyMorphic  
Systems

460 Ward Drive Santa Barbara California 93111 (805) 967-2351



1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100

The operating system described in this manual is the fourth release of the PolyMorphic Systems System 88 operating system software, Executive version 80 and later. The software described herein was created by R. T. Martin, Frank Anderson, Len G. Danczyk, and Roger L. Deran. The manual was written by Robin C. Soto and Dr. Gerald A. Bradley, and revised by Jennifer M. Douglas.

This manual is PolyMorphic Systems part number 810165, Revision D. Copyright 1979, Interactive Products Corporation. All rights reserved.



1000  
1000

1000

1000

1000

1000

1000

1000

1000

1000

1000

## PREFACE

### Summary of Changes in Version 4 of the System Software

This manual is Version IV of the System 88 User's Manual. It describes the 4th release of the System 88 software and its use with both single and double-density systems. Those users who are familiar with the previous version of the System 88 software, will need to note the following changes:

-The storage capacity of the double-sided, double-density diskettes is four times greater than that of the single-sided, single-density diskettes.

-It is imperative that double-density systems not be powered up or down while disks are in the drives and the drive doors are shut. If this does occur, the information on the disk will be harmed.

-Media intended for the Double-Sided 8813 (All Double-Density 8813 Systems are Double-Sided) must not be interchanged with media intended for the Single-Sided 8813 (All Single-Density 8813 Systems are Single-Sided).

-If you have a single-sided and a double-sided 8813 system, you can transfer files from one system to the other through the use of the File Transfer Program which is also included on the System Disk. Appendix-I of this manual explains this process.

-It is now possible to have additional directories on a single diskette. These additional directories are called "sub-directories" because they are off-shoots of the main directory or of each other. Files can now be stored on any of several different levels of the directory tree.

**WARNING:** Disks with sub-directories should not be used with any older version of the system software. If you use any older versions of the system software to PACK a disk with sub-directories, the information in your sub-directories will be permanently erased. The Version 4 software will not allow you to copy sub-directories. If, however, you use an older version of the software and try to copy a sub-directory the system will assume you are trying to copy a file. The result will be that you will have copied a useless directory; none of its dependent files will have been successfully copied.

-Because of this new structure, it is necessary to specify sub-directory names when you are using the sub-directory capability.

-The left bracket may be used in place of the right bracket in

file and path names.

EXAMPLE:

EDIT <2<Filename

is used rather than

EDIT <2>Filename

-The right bracket is acceptable, but using the left bracket allows easy building of longer "Path names."

-In this manual we use the term "Path name" to refer to the specification you must provide in order to indicate what is to be affected by a command.

-Longer path names will be necessary if the sub-directory capability is to be used. To find out more about specifying complete path names, the use of brackets, and the use of the sub-directory, read sections 3, 4 and 5.

-CHECKSUM is a new command which causes a specified directory to be listed with an additional column showing the CHECKSUM for each file listed in that directory. This will help you to determine if a program has been altered. This file can be deleted from the System Disk, but it is not recommended that you delete it since it is helpful in diagnosing problems.

-Mirror and Dup are now included on the System Disk. These two programs allow you to copy a file or image a disk on a single drive system. See Appendix-J for complete instructions on their use.

-The new System Software includes a new printer driver which is described in full in Section 13 of this manual. The new commands associated with this printer driver are:

Printer CUSTOM

which allows the user to provide a program to handle communications between the system and a printer.

Printer SHOW

which allows you to view the default page parameters in effect.

Printer SET

The page parameters which are normally in effect are the ones you specified in the definition of the printer that is being used. However, you may want to change these parameters to accommodate some special circumstance (e.g.

different sized paper or an unusual page layout). Printer SET allows you to establish new parameters that will remain in effect until the system is turned off or loaded. This convenience allows you to specify parameters without having to redefine a printer.

#### Setup

allows you to define a printer for the printer driver.

Read Section 13 for complete instructions on the use of the new printer driver. Appendix H contains information on the CUSTOM function, and provides a sample, user-written program.





## PART I

## OVERVIEW OF SYSTEM 88 OPERATION

## INTRODUCTION TO PART I

The sections in Part I provide you with all of the information that you need to know to operate your System 88: how to physically operate your machine (Section 2) and how to use the system software (Sections 3-6).

Section 2 describes your system's physical components and tells you how to turn on the machine, how to insert a disk (and what a disk is), and how to use the keyboard.

Section 3 discusses the concept of a "file" and tells you what kinds of files may exist in the system.

Section 4 explains the proper form for Path Name specification and the method for getting the information in a disk file into the memory, where you can work on it.

The disk's "table of contents" is the main directory for a disk. The user has the option to create sub-directories, and Section 5 explains how and when you should make use of this sub-directory capability.

The Executive is the section of the system that handles your communications with the system. Section 6 talks about the Executive in detail and discusses the various commands recognized by it. This section also provides information on what goes on inside the disk operating system when your machine is turned on.

### THE PROPER ENVIRONMENT FOR YOUR SYSTEM 88

The proper environment for your System 88 is one which is reasonably free from dirt and smoke. Under normal circumstances some dirt collects in the drives over a period of time, but if the environment is particularly dirty, the drives and heads can be so affected that the media will be damaged as a result. Double sided systems are particularly susceptible to this condition, and so should be operated in a "computer room" environment, free of chalk dust, thick carpets and smoke.

After one year or 6,000 hours of use of your System 88, follow the cover removal instructions in the 8813, 8810 or 88/MS HARDWARE appendix. After you have removed the wood cover, vacuum the system. Do not touch or attempt to clean the heads (if you have a 88/MS refer to the 88/MS User's Manual for head cleaning procedure.)

## Section 1

## GETTING STARTED WITH THE SYSTEM 88

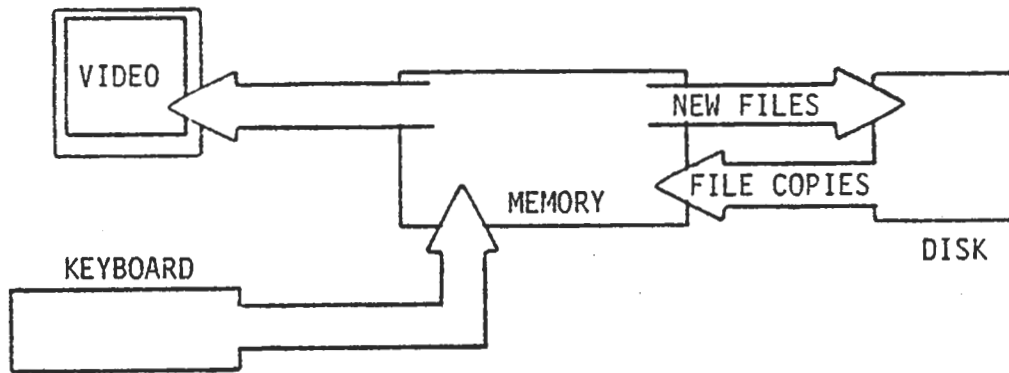
## 1.0 INTRODUCTION

Small computers like the PolyMorphic Systems microcomputer, the System 88, are putting the power of the computer into the hands of many, many people who do not have access to the huge computers of big corporations and scientific establishments. The definition of "computer user" is fast becoming broader and broader, embracing not only the computer professional but also professionals of other kinds, business executives and staff members, students, and hobbyists.

Because "computer user" is now coming to mean any person who wants to increase his or her effectiveness, builders of microcomputers, like PolyMorphic Systems, are trying to correct the impression that computers are hard to understand and hard to use. Wherever computers HAVE been hard to understand and use, builders are striving to improve them. The mini-floppy diskette, for instance, is a breakthrough in ease of use and storage. And the documentation that accompanies these systems is intended to be comprehensible to the inexperienced and experienced computer user. Frequently we tell you when a section will be of interest and benefit to the inexperienced computer user, and when it will be of most value to the more experienced programmer. There are many explanations about how the system operates which should be of interest to those of technical and non-technical background alike.

In that spirit, consider this conceptual description of the System 88. The disk is a storage place, like a filing cabinet. Data of any kind is magnetically recorded on it. All that is really "on" the disk is a large number of small magnetized locations. The system can "read" the patterns of magnetized locations and interpret the patterns as information.

Figure 1. Movement of Information



To work with the information stored on the disk, the system makes a copy of a disk "file"-- as you might make a copy of the contents of a file folder-- and puts that copy into "memory," which is a temporary storage place inside the computer's main unit. Memory is the actual work-space where you put things that you want to examine and change. It corresponds to your desk-top, where you spread out the copy of the file so that you can read it and mark it up. You work on the contents of memory by using a keyboard, and you see the part of memory you are affecting, and how you are affecting it, on the video screen. When you are done working with the contents of memory, you can store it on a disk; you can also delete the previous version of the disk file from the disk.

In other words, you can update your filing cabinet by replacing the old contents of the file folder with new contents. Of course, you can create entirely new files and save them on disks, too.

The System 88, as delivered, includes a pre-recorded operating system, partly on the disks labeled "System Disks" and partly on components inside the main unit. The collection of programs on the System Disk which we refer to as "software" represent the system's intelligence; they are responsible for the system behaving the way it does. Before successful operation of your system, you must insert a disk which contains these essential programs. All applications disks, the System Disk, and the WordMaster disk contain the operating software. This "software" is not shown in the conceptual drawing above.

(To verify which system disk you should have, read section 1.1 below.) All of the above mentioned disks include "Exec" as part

of the software. Exec is the program which takes control automatically when you Load or power up your system with the System Disk in drive one (System Residence drive). Exec allows you to communicate directly with the system, and there are specific commands for you to use for this purpose.

There are other programs on each of these disks which you can ask Exec to run. Besides Exec, the System Disk contains BASIC, the Assembler and the Editor.

When you invoke one of these other programs on the System Disk, BASIC for instance, you are telling the system that you want BASIC to receive your input rather than Exec. BASIC is a simple computer "language" which provides an easy way to do hard problems. When you awaken BASIC you will then be communicating with the BASIC interpreter which translates your input for the system.

The Assembler is another program on the system disk which you can invoke from Exec and which takes over the communication between you and the system. The assembler is used by those who are writing programs in assembly language.

The Editor is a program that you may be using quite frequently as it allows you to create and change text quickly and easily.

The WordMaster disk contains the software necessary to use the System 88 for word processing. Programs on this disk allow you to create consistently attractive documents of all types. If you are using the System 88 to create an instruction manual like this one, you will be using WordMaster and the Editor so you will want to use the WordMaster disk instead of the System Disk.

This manual devotes a section to each of the four programs on the System Disk; read these sections for a sound introduction to their use. For complete instructions on the use of the Editor and WordMaster for word processing, read the System 88 WordMaster manual. For extensive tutorial and reference information on BASIC consult the System 88 BASIC manual. The Macro 88 Assembler manual provides instructions on the use of the assembler. System programmers will also want to read The System 88 System Programmer's Guide

## 1.1 SYSTEM 88 MODELS

The System 88 product line consists of the System 8813, available with up to 3 mini-floppy disk drives; the System 8810 with one mini-floppy drive, and the 88/MS add-on with 2 large floppy drives.

The 8813 and 8810 models are available with optional double-sided, double density, mini-floppy disk drives. The 88/MS, an add-on storage for the 8813, is available with either double or single-sided, double density large floppy disk drives.

**IMPORTANT:** The media (floppy diskettes) are not interchangeable between single and double-sided systems. Therefore you must not use media intended for the single-sided in the double-sided double density 8813. (NOTE: All single density 8813 systems have only single-sided drives, and all double density 8813 systems have only double-sided drives.)

The blank, mini-floppy diskettes which you purchase for use with your double-sided system should be designated "double-sided" on the label. PolyMorphic Systems double-sided applications and System diskettes will contain this designation and a colored dot on their labels. Single-sided diskettes have no special designation on their labels.

### 1.1.1 What Software Should You Have Received?

Check the following chart to be sure that you received the correct software for your model.

8813 and 8810 Systems are shipped with:

- 5" Single Density System Disk Part Number 820190
- 5" Single Density Confidence Disk Part Number 820186
- 5" Single Density WordMaster Disk Part Number 820131

Double Density 8813 and 8810 Systems are shipped with:

- 5" Double Density System Disk Part Number 820191
- 5" Double Density Confidence Disk Part Number 820163
- 5" Double Density WordMaster Disk Part Number 820162

Your First 88/MS Add-on Storage Unit is shipped with:

- 8" System Disk Part Number 820188
- 8" Confidence Disk Part Number 820175
- 8" WordMaster Disk Part Number 820166

You should receive one Confidence Disk for every drive in your system.

## 1.2 MANUAL ORGANIZATION

The purpose of this manual is twofold: to help you to begin to use the system right away, and to provide essential reference material for your present questions and questions that might arise later.

If computers are new to you, you may want to just skim sections dealing with the inner workings of the system and discussing machine language programs. Besides straightforward information on how you operate this system, we include material explaining some of the basic concepts behind microcomputer programming. We have indicated the sections containing information which, while

useful, is not strictly necessary to the operation of the system.

If you are an experienced computer user, you will probably rely most heavily on the reference list of system commands included in this manual, which briefly summarizes each command and gives its correct form. You may find much of the introductory material interesting to read, even though you are already familiar with it. Specific information on the internal structure of the system software is included in the System Programmer's Guide (available separately). The System Programmer's Guide provides information on writing your own system software and connecting your own machine language programs to the disk operating system.

.....

The following section of this manual invites you to sit down at the keyboard and try some simple operations. The remainder of the manual is divided into three major sections. Part I contains basic information concerning the operation of the system. It discusses the physical components of the system and their use. We also introduce the ideas of a file, a disk directory, and a disk operating system we call Executive, concepts essential for successful use of the system. And we list the commands used in the system and discuss their use.

Most of the information in Part I will be of use to the non-programming user of the system as well as to the programmer.

The material in Part II, on the other hand, is devoted to the actual construction and use of disk files, and is aimed at the person planning on actually programming on the system. The information covered in Part II is presented in a way that helps the non-programmer, with the aid of a programming textbook, to quickly produce his or her own files.

Part III contains information on the system Editor and Assembler. You use an Editor to create and edit text files. This manual, for instance, was created and edited as several disk files using the System 88 Editor. Almost everyone who uses the System 88 will be using the Editor extensively. You can make full use of the Editor without knowing anything about computers or computer programming. An Assembler translates assembly language programs into machine language, and the Assembler section is required reading only for assembly language programmers.

As you read through this manual, you will find that the material introduced in earlier sections is expanded upon in later sections. If at any time you feel lost, look up the topic that you are unsure about in the Index or Glossary.

### 1.3 QUICK DEMONSTRATION OF SYSTEM USE

Even before you learn all of the details about the System 88 software, we're going to take you through a quick demonstration of the use of your machine. By the time you leave this section, you will have done many of the kinds of things you can do with the system. If you want more information on each subject we cover, check the Index.

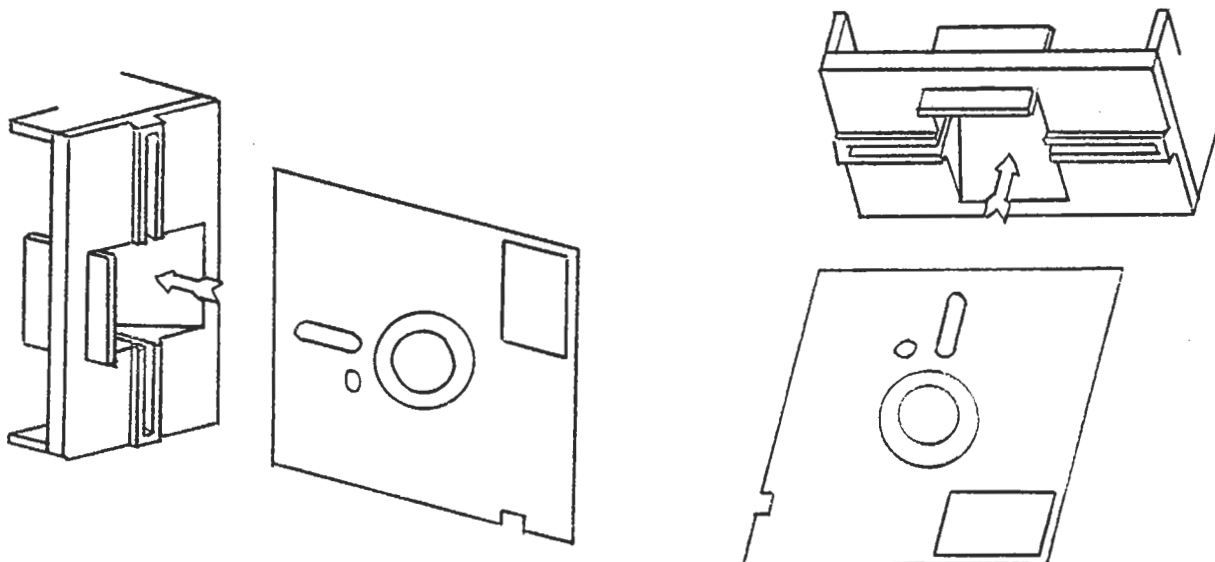
Anyone who can use a typewriter keyboard can sit down and begin using the PolyMorphics System 88. So whatever you are-- novice or expert-- let's start.

If you have an 8813, insert the key in the lock on the main unit, press in on it, and by turning the key clockwise, turn the system ON. If you have an 8810, press the top of the rocker switch to turn the system ON. Turn the video monitor (TV set) ON.

Take the System Disk out of its paper envelope--but DON'T remove the disk from its black sheath. The disk stays inside its black sheath at all times. The disk is very delicate, so handle it carefully. Don't touch the exposed areas of the disk itself.

Open drive 1, the left-hand slot (some systems have only one slot) on the front of the main unit. To insert the disk in a vertical drive, slide the disk into drive 1, notched edge down, with the disk label to the left. To insert it into a horizontal drive, slide the disk into the drive with the notched edge to the left and the disk label up (see Figure 2).

Figure 2. Putting the disk into the drive.





Push the disk all the way home gently. Close the drive. Don't force the drive shut; if it won't close, make sure the disk is all the way in.

**WARNING:** If you have a Double Density 8810 or 8813, or if you have an 88/MS add-on you must not power the system up or down while the disks are in the drives and the doors are closed. If you power the system up or down while disks are in the drives and the doors are closed, you will harm the contents of the disk. If you follow the instructions in this section and turn on the system before inserting the diskettes you will prevent this problem.

When you have properly inserted the disk and closed the drive door, push the button marked "Load". Note that the load button begins to blink, indicating that processing is taking place. After ten to twenty seconds, this message will appear on the monitor screen:

```
(Exec/##-top of RAM is ####.)  
$
```

The cursor (a solid white rectangle) will be just to the right of the dollar sign. The message tells you the version number of your System 88 system software, and the highest memory location that the system can use.

If this message is not displayed, go through the hookup procedure and the above procedure again.

The dollar sign is the "system prompt." A prompt is a symbol indicating that the machine is waiting for an instruction from you. The cursor symbol indicates your screen position; the next character that you type on the keyboard will appear on the screen where the cursor is now.

The keyboard is much like a typewriter keyboard, with numbers, symbols, and upper and lower case letters. Type all instructions below EXACTLY as shown, including spaces. Type all instructions in capital letters. (Note that the CAPS LOCK key affects letter keys only; you must use SHIFT to shift the other keys, even if CAPS LOCK is locked down.) The symbol (CR) means "hit the carriage return key"; all instructions end with a carriage return. Do not space before hitting the carriage return. You can erase symbols on the screen by hitting the DELETE key; letters disappear one at a time from right to left.

Now let's start using the system.

First we'll find out what is already recorded on the System Disk. Disks have a directory, or table of contents, listing the names of any files they contain, just as file cabinet drawers often have front labels listing the files they contain. A disk file contains information, just as the folders in the file cabinet

contain papers and letters.

Remember, (CR) means carriage return. When you see (CR) in this section, hit the RETURN key. All instructions end with a carriage return. Type this:

```
LIST(CR)
```

You now see a list of the contents of the System Disk. Next you will be adding a file to the System Disk, and the name of the file (and its location and length) will automatically be added to the directory.

To add a BASIC file to the disk, begin by typing:

```
BASIC(CR)
```

The machine is now "in BASIC": it will now understand instructions written in the BASIC "language." (It doesn't matter at this point if you don't know BASIC.)

The fact that the system is in BASIC is indicated by the BASIC prompt > that now appears on the screen. The prompt you see now is different from the previous prompt \$. This is because you are talking with BASIC instead of with the Executive.

We will add a BASIC file (in this case a computer instruction written in BASIC) to the System Disk. The file will consist of the words "WELCOME TO THE SYSTEM 88" (or any other message of a few words that you make up). Type:

```
10 PRINT "WELCOME TO THE SYSTEM 88"(CR)
```

```
SAVE;INITIAL(CR) (NOTE: You are naming your file initial)
```

The red light on drive 1 now comes on, to indicate that drive 1 is working. What is happening is that the system is "saving" the message by putting it (or "writing" it) on the disk. Up till now the message has been stored only in the temporary memory inside the main unit. The light goes out when the file has been saved on the System Disk. You will now see the BASIC prompt > ; this means that you can go on.

Type:

```
BYE(CR)
```

You are now out of BASIC. The name of the new file (your message) now appears in the disk directory. To see it, type:

```
LIST(CR)
```

There it is. Now, to send a copy of your file to the screen, type:

## TYPE INITIAL(CR)

You can ask the machine to start up your program that you have saved as a file simply by typing:

```
INITIAL(CR)
```

We chose the name INITIAL for a special reason. Whenever you reset or turn on your machine, if the System Disk contains a file named INITIAL, the system will "run" that file without waiting for any instructions from you. Try this out; hit the Load button. Your system will restart itself and run your program INITIAL.

You will see that you are again in BASIC (you see a BASIC prompt > ). Now you'll enter a simple program in BASIC. (A program is a set of instructions for the machine to perform.) Number program lines so that BASIC will know in what order to perform the program commands. Type, exactly as shown:

```
10 PRINT "HOW DO I STOP THIS THING?"(CR)
20 GOTO 10(CR)
RUN(CR)
```

Program line number 10 tells BASIC to print on the screen the words within the quotation marks. Program line number 20 tells BASIC to return to program line number 10. After this two-line program is entered, you type the RUN command; this tells BASIC to execute the program (that is, to perform the program commands).

The words HOW DO I STOP THIS THING? are now being printed rapidly over and over on the bottom line of the screen. As the words are printed, the previously printed lines scroll up and disappear at the top of the screen. The program is an endless loop; the first of the two instructions says to print the words, and the second says to go back to the first instruction. So the computer repeats the PRINT instruction rapidly and endlessly. Speed and resistance to boredom are a computer's main virtues.

To stop this program, hold down the control key (CTRL) and type a Y. You will notice that the BASIC prompt is now a double prompt symbol >>. This indicates that program execution has been interrupted. You can make the program continue to run by typing CON.

A small modification in the program limits it to a prescribed number of runs. Type this, hitting the carriage return at the end of each line:

```
10 PRINT "I WILL STOP MYSELF"(CR)
20 I=I+1(CR)
30 IF I<10 THEN GOTO 10(CR)
RUN(CR)
```

The program will execute ten times and stop.

Here is a slightly more complicated program. For WORD below, substitute any short word of your choice. Don't forget to hit a carriage return at the end of each line.

```
10 FOR X=1 TO 200(CR)
20 PRINT TAB(25+25*(SIN(X/3))), " WORD "(CR)
30 NEXT(CR)
```

Let's save this program by typing:

```
SAVE;SINWAVE(CR)
```

Now, after you receive a BASIC prompt (indicating that the machine is ready for another command), type:

```
RUN(CR)
```

The system is now printing your word repeatedly at the bottom of the screen, each time at a different tab (indentation). As the word prints, each line above it scrolls up and eventually disappears off the top of the screen. The machine selects the number of spaces to move over before printing the word by performing the computation you gave it in line #20. To be precise, the machine computes the sine of X divided by three, where X=1. This quantity is multiplied by 25, and then 25 is added to it. This gives the first tab value. Since X has not yet reached its terminal value (200), the system goes back to line #20, and performs that line with X=2. This continues on with X increasing by 1 until X=200; at that point, line #20 is performed one last time, and the program is finished.

Each time the word is printed on the screen, BASIC has had to go through several different steps to calculate the tab value. As you can see, the system works fast. (This tab formula, by the way, generates a sinusoidal wave form.)

Now, "leave" BASIC and return to the system level by typing BYE and a carriage return. You will once again see the system prompt \$, indicating that you are out of BASIC and again talking directly to the system.

We are now going to delete a file. (Remember to hit a carriage return at the end of each line that you enter.) Type:

```
DELETE INITIAL(CR)
```

You will receive the following message:

```
<1<INITIAL.BS deleted.
```

If you display the directory again (by typing LIST), you will see that INITIAL is no longer displayed in the disk directory.

Now that you have learned how to save BASIC programs as files, list disk directories, save an INITIAL file, and delete files,

you will learn how to construct and use a very powerful tool of the system: a command file. Command files are files from which the system draws instructions. Such a file contains a series of the kinds of commands that you might enter from the keyboard; whenever you want the computer to perform that set of instructions, instead of having to type them, you have only to run your command file .

One of the easiest ways to construct a command file is to create a text file using the Editor. A text file is a collection of letters and symbols. This may be simple text, such as a book chapter or a letter to your bank. Or a text file may be a program. The purpose of an Editor is to allow you to type in text and make changes easily in that text.

Type:

```
EDIT YOUR-COMMANDS
```

(Don't forget to hit a carriage return.) At the top left hand corner of your screen you will see "Edit/##" (the version number of the Editor). Next will appear the message:

```
Input file: not found
```

```
Output file: opened
```

```
Hit any key to continue.....
```

"Input file: not found" means that the Editor knows that you are creating a new file, not editing one already in existence. After displaying this message, the Editor will wait until you tell it to continue. To do so, type any character from the keyboard. The screen will clear and the cursor will appear at the top left corner of the screen. You are now ready to create a text file that you will use as a command file.

Note: Lines beginning with a semi-colon are comments and are ignored by the system. When you type the following example, you do not have to type the comments. If you do, make sure that a space follows the semi-colon wherever you insert a comment.

```
; This is a command file
; It contains the instructions that will tell the
; system to
; 1) list your System Disk directory,
; 2) run your BASIC program,
; 3) save it under another name,
; 4) exit BASIC,
; 5) list the directory again,
; 6) delete the copy of your program, and
; 7) display your program.
LIST
SINWAVE
SAVE;WAVE-2
BYE
LIST
DELETE WAVE-2
TYPE SINWAVE
```

Now exit from the Editor. Type an Escape (the key marked ESC). Next type a Control-E by holding down the Control key (marked CTRL) and typing an E. You will see the message "Exiting...." When the Editor is done saving your text file, you will see the system prompt \$ once again. To run your command file, simply type:

YOUR-COMMANDS (CR)

The system will now start drawing its instructions from your file because it realizes that your file is not a program, and so must be used as a command file.

As the system performs your command file instructions, you will note an interesting thing about the execution of your BASIC program -- it runs even though you did not ask the system to bring in BASIC. The system brings in BASIC automatically to run a BASIC program file. The system is able to decide if a file that you have asked for is a BASIC file. If it is, BASIC is brought in by the machine and told to run the BASIC program.

You have now done all the procedures you will use most often when using your machine. You may be surprised at how little else there is to learn in order to use your machine effectively.

## Section 2

## SYSTEM OVERVIEW

Before you use your System 88, a few brief explanations are in order. This section tells you how to handle disks, and how to use the disk drives, the keyboard and the Load button, and start up the system. The actual hardware specifications for your System 88 may be found in the User's Hardware Guides (Appendixes F and G).

## 2.1 THE DISK

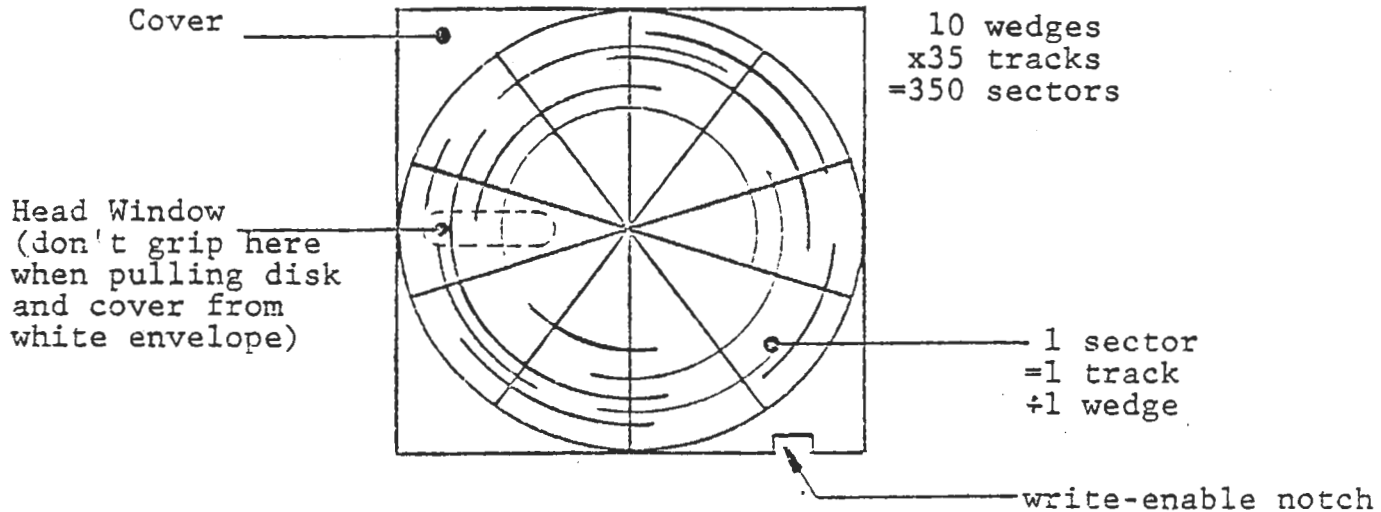
The disk is a device for storing data. Data are recorded on the disk surface much as music is recorded on a cassette tape. The round disk is permanently enclosed within a square protective covering, and rotates within that sheath. A disk drive "reads" data on the disk as the disk rotates, in somewhat the same way that a needle "reads" music on a phonograph record. The drive can also record or "write" data onto the disk.

Your computer uses the type of disk known as a "minifloppy diskette." This means that the disk is flexible and small. The amount of data which can be stored on each "minifloppy" is determined by the type of System 88 you have. Single-Density models can store 89,600 "bytes" (a byte is a small unit of information, usually representing one character) or about forty pages of single-spaced text. Double-Density models store data at twice the density and store it on both sides. This provides 357,400 bytes of storage.

Actually, the amount of storage provided by a diskette is usually measured in sectors, rather than bytes. A sector is 256 bytes. Single-Density diskettes store 350 sectors, while Double-Density diskettes store 1400 sectors. A typical page full of typewritten text requires 10 sectors.

Take a disk out of its paper half-envelope. The paper envelope is NOT the black protective cardboard enclosing the disk itself. NEVER remove a disk from the black holder.

Figure 2. Cutaway drawing of a disk.



When you look at the disk, you will see that only a few areas of the actual disk surface are visible. This is because the surface of the disk is extremely delicate. NEVER touch the disk areas not covered by the protective cardboard. Don't ever write on the protective covering except VERY lightly with a felt-tip pen: it's important that no impression be made upon the disk. A disk should never be folded, bent, kept in temperatures over 125 degrees Fahrenheit or under 50 degrees Fahrenheit, or placed near a magnetic field. Always keep the disk in its white paper envelope (and stored in a safe place) when you are not using it. Four sectors of every diskette are saved for directory space. This leaves 346 sectors free for files or sub-directories.

Usually you can write information onto a disk as well as read data from it. However, you can "write-protect" your disks. The system cannot write data on a write-protected disk. This feature ensures that the data on a disk remains intact. The disk cover has a small notch in one edge called the "write-enable notch" (see Figure 2 above). To write-protect a disk, stick a "write-protect tab" over the write-enable notch. The write-protect tabs are the small stickers provided with your disks. Put the tab on so that it covers the notch on both sides of the disk cover. If you should want to write data on a write-protected disk, remove the write-protect tab from the write-enable notch and use the disk in the normal way.



The disk of most importance to the operation of your machine is the System Disk. This disk contains the files used by the disk operating system as parts of the system itself. This disk is essential to the operation of the system and must always be placed in the System Drive, usually drive 1, the drive farthest to the left).

## 2.2 THE DISK DRIVES

The front of your System 88 contains one or more slots with small doors or latches. These are the openings to the disk drives. Within these drives is the apparatus which writes information to and reads it from the disks.

The drives are numbered from left to right, beginning with number 1. You must always place the System Disk in the System Drive, usually drive 1. If you have an 88/MS add-on you may be using drive 4 as the System Drive.

A small red light is on the front of each drive. When this light is on, the disk inside the drive is being either read from or written to.

CAUTION: Never interrupt the system when one of these lights is on. (Don't turn the system off until all these lights are off.)

## 2.3 THE KEYBOARD

Your system uses a full upper and lower case keyboard. When the system is first turned on or reset, you can use the full upper and lower case character set. This is called "the FULL mode." In this mode the keyboard is almost identical in use to a typewriter keyboard.

Just as with a typewriter, you use the SHIFT key, for upper case letters, and for the symbols on the main keyboard number keys.

Typing with the CAPS LOCK key down also produces upper case letters. The main keyboard number keys, however, are not affected by the CAPS LOCK key with the exception of the following keys: ^, 0, -, [, ], \. If the CAPS LOCK key is depressed, the shift key can not be used to obtain the other symbols on these keys.

Use the DELETE key to delete one character at a time. (A character is any symbol that you can type on your keyboard--numbers, letters, symbols, control characters, etc.) Deletion is from right to left.

A useful feature of the system is its 64-character type-ahead capability. "Type-ahead" means that you can type up to 64 characters during times that the system is not able to pay attention to the characters that you type in from the keyboard (for instance, during the running of a BASIC program). The characters entered during that time will be displayed on the

screen and processed when the system is once again free to deal with information from the keyboard.

### 2.3.1 Numeric Pad

The numeric pad located to the right of the main keyboard area is unaffected by the SHIFT and CAPS LOCK keys on the main keyboard. The Roman Numeral keys are function keys which are not used by Exec. They will only be used by programmers.

### 2.3.2 Control Characters

Certain special functions are performed by the use of characters called control characters. To type a control character, hold down the CTRL key on the keyboard and type the letter indicated.

#### EXAMPLE:

Control-Y -- Hold down the CTRL key and type y.

You may use the following control characters when you are typing instructions to your machine. (These commands are also available for use when you are using the computer language BASIC.)

Control Characters Used for Deletion (deletion is to the left) are:

Control-W: Deletes one word at a time.

Control-X: Deletes entire line

Other Control Characters:

Control-Y Halts whatever action is being performed.

Control-L Form feed-- clears screen.

Control-I TAB-- moves cursor to the next tab position (tabs are every eight spaces), as does the TAB key.

NOTE: When you are displaying a file on the video screen (by using the TYPE command), a Control-K character in your file causes the cursor to move to the upper left corner of the screen; a Control-L character clears the screen. When you are using the Editor, Control-L enters a Greek lambda into your file which, when sent out to a printer, causes it to move on to the next page or "form"; Control-K enters a Greek mu, which some printers also use as a form feed. Typing either Control-L or Control-K while in BASIC or Exec will move the cursor, but will produce a "syntax error."

### 2.3.3 Other Keyboard Case Modes

In FULL mode, the keyboard works essentially the way a typewriter

keyboard does. But sometimes it is inconvenient to use the full character set. For instance, most commands must be typed in upper case letters. Rather than holding down the SHIFT key or keeping the CAPS LOCK key on, it would be convenient to have the machine automatically convert all letters to upper case. This can be done by using the fold command. After a system prompt \$ type:

```
fold
```

(in lower case) and a carriage return. The system prompt \$ is the indication that the system is ready for your next command. From the time that you use the fold command until you type:

```
FULL
```

followed by a carriage return, all letters will appear in upper case. This command affects only the letters (a-z): the other keys are unaffected by the fold command.

Fold mode is useful when you are not using lower case letters at all. You may find yourself using mostly upper case letters, and lower case letters only occasionally. The "flip" command switches upper and lower case. Starting in FULL mode, after a system prompt \$ type:

```
flip
```

(in lower case) and a carriage return. From this time until you change the keyboard mode, you will use the SHIFT key to type lower case letters. Upper case letters will be produced when you type unshifted letters. For example, when in flip mode, an unshifted r will produce a capital R. A shifted r will produce a lower case r. This command affects only the letters (a-z); the other keys are not affected.

## 2.4 OPERATING THE SYSTEM

Now that you have some idea of what the physical components of your system look like and what they can do, you will want to start using the machine. If you have not already looked at Section 1.3, A QUICK DEMONSTRATION OF THE USE OF THE SYSTEM, you might wish to do so after reading the rest of this section. The following paragraphs will tell you how to get your system into operation. Section 1.3 shows you a few things that you can do once you turn on your machine.

### 2.4.1 Turning on the Machine

Insert the key into the ON-OFF lock on the front panel, press in on it, and turn it to the right until it stops. The red light above the lock is now lit, to indicate that the machine has been turned on. (To turn off the machine, slowly turn the key back to OFF).

### 2.4.2 Placing Disks in the Disk Drives

Take the colored paper envelope off of the disk. Hold the disk lightly, being careful not to touch the exposed sections of the disk surface. Hold the disk vertically so that the disk label is on the left side of the disk and in the upper corner. The square notch in the edge of the disk cover will be on the bottom edge of the disk. (Some systems have one horizontal drive; put the disk in with the label up and the notch in the edge to the left.)

Open the disk drive door and carefully slide the disk all the way into the slot. Close the door. The drive cannot write or read information to or from the disk if the doors are open!

The System Disk must always be placed in the System Drive, usually the drive farthest to the left.

Once the System Disk has been inserted in the System Drive, Load the system.

The white button on the front panel of your machine labeled Load is the reset button. Always push it when you want to restart the disk operating system. When you push the Load button, the disk operating system starts up. You will see the following message on the screen:

```
(Exec/83-top of RAM is ####.)  
$
```

The number following the word "Exec" is your system software version number. The next number tells you the highest address of usable memory on your system.

The dollar sign symbol \$ is the system prompt. It tells you that the system is waiting for your command. For more details on what occurs within the system at start-up time, see Section 6.4, SYSTEM START-UP.

### 2.4.3 Hazards During Disk I/O

Information is stored on a disk by writing data from the temporary memory in the main unit onto the disk. The data is recovered by reading it from the disk and placing a copy of it into memory. These read and write operations are called "disk I/O" (Input/Output). The red light on the front of each disk drive is lit when I/O is being performed on the disk inside that drive.

There are three things that you must not do while disk I/O is taking place. DO NOT hit the Load button, open a disk drive door, or turn off the system. Any of these actions will interrupt read and write operations and may scramble the contents of the disk. If the system is interrupted while it is updating a disk directory, all of the contents of the disk may become inaccessible.





## Section 3

## OVERVIEW OF FILES

Disk files are groups of data. The data can be anything you want them to be--words, numbers, results of experiments, client files, games, BASIC programs, statistical calculation programs, etc.

A disk file is a storage device, like a folder in a file cabinet. Your folder may hold any type of information that you desire. You identify each file by its label. A collection of folders is grouped together in a file cabinet drawer just as disk files are grouped together on a disk. You may place a label on the front of a file cabinet drawer listing the names of the files within. In the same way, every disk has an area set aside called the disk directory--a "table of contents" that lists files and sub-directories found on the disk.

Imagine a file cabinet in which you do not actually remove any of the folders from the drawer. Instead you somehow tell the cabinet that you want the information in a particular folder in a particular drawer. The cabinet then makes a copy of the contents of the folder and delivers that copy to your desk, where you can work on it. You can always replace the old contents of a folder with NEW information, but any information that you take out of the folder is only a copy of the folder's contents. This is how a disk system works. You tell the system which file you want on which disk, and a copy of that file is placed into the memory of your machine, where you can work on it. If you want to know which files are in a particular "drawer" (disk), you ask the system for a display of the disk directory (see Section 6.2.2, LIST). At any time you may get rid of old files (see Section 6.2.3, DELETE), create new ones (see Part II, USING AND BUILDING FILES), and get the information out of existing files (see 4.10, INVOKING FILES).

There are five different kinds of files, categorized by the type of information they contain:

Machine language program files:

Although you communicate with the system by using various kinds of statements, even English words (like LIST, FULL, etc.), the machine itself understands only binary data (1s and 0s). The capital letter "R", for example is interpreted by the computer as the following pattern of binary numbers: 01010010. These binary statements are called "machine language."

A series of these machine language instructions, intended to be followed in sequence, is called a

machine language program. A machine language program file consists of nothing but binary numbers that represent machine instructions. The file BASIC.GO on the System Disk is a machine-language program file containing the interpreter for the language BASIC.

It is awkward writing a program in machine language form. Instead, we can write a program in a computer language called "assembly language." This program is then translated into machine language by your system. The entity which translates assembly language programs into the numbers used by your machine is called an "Assembler," and is itself a machine language program. Once "assembled" (translated by the Assembler), an assembly language program is a machine language program.

For information using the System 88 Assembler, see Section 12, THE MACRO-88 ASSEMBLER.

#### System files:

Certain files, called system files, have attributes that set them apart from regular files--you may not use the following commands on them: RENAME, DELETE, COPY, TYPE, PRINT, or EDIT. Any kind of file may be designated a system file (e.g., the file holding the language BASIC is a system file). (For information on designating a file as a system file, see the System Library volume, System Programmer's Guide, available separately.)

A special kind of system file is a file called a system overlay. An overlay is a file used by the system as an actual part of the system itself. You cannot execute an overlay; only the system itself can call overlays into action.

Usually system files will not be listed when a disk directory is displayed. (To see how you may see system files listed, see Section 6.2.1, ENABLE and DISABLE.)

#### BASIC program files:

Programs written in BASIC are stored as files in text form, that is, in the same form in which they were originally typed into the machine. When brought into memory by the system, such a file causes BASIC to be automatically invoked by the system to run the program file. A BASIC program file may also be brought into memory by BASIC. For information on writing BASIC programs, see the System Library volume BASIC: A Manual.

#### Command files:



A command file is a file containing system commands. For information on using and constructing a command file, see Section 9, COMMAND FILES.

#### BASIC data files:

A BASIC .data file is a file built using BASIC file-handling commands. Such a file holds numeric and string data. It might hold a book chapter, for example, or a list of prime numbers. For information on constructing and using data files, see the System 88 BASIC Manual

#### Text files:

A text file is a file containing data in text form. It holds the same sorts of data as a BASIC data file but is created using the Editor. A text file can be a BASIC program, an assembly language program, a command file, an office memo, a travel itinerary--anything in text form. For information on using the Editor to create text files, see Section 11, THE EDITOR.

#### Sub-directories:

Sub-directories are stored on the disk like files. Listed in sub-directories are more sub-directories and files which are not found listed in the main disk directory. This means that in addition to the 4 sectors set aside for the main disk directory, there can be other directories on the disk that are off-shoots of the main directory. Listed in those sub-directories can be even more sub-directories.

You can not COPY, EDIT, TYPE or PRINT any directory.

### 3.1 FILE AND DIRECTORY NAMES

All files and directories have names. The name for a main disk directory is the same as the name of the drive which contains it. Sub-directories and files are given names from 1 to 31 characters in length. Since this 31 character limit includes the 3 character extension, which the system affixes to the end of all sub-directory and file names, your file name must not be any longer than 28 characters. (See 3.2 of this section for explanation of extensions)

File names may contain numbers, special symbols, and upper and lower case letters. They may even contain control characters (e.g., Control-L). But it is, generally speaking, not a good idea. Control characters will not show on the screen when you are typing the file name. They are in the name, however. This makes file names with control characters in them hard to remember, and worse to type. When a file name is listed in a disk directory display, any control characters in it will show up

as Greek letters. (To find the Greek letter representation of a control character, see Appendix B: The ASCII character Set.)

The symbols that a file name may not contain are a comma, a space, a period, a plus sign, a TAB, a carriage return, or an escape.

Any other symbols or combination of symbols are legal, except for the one-character file name \*. This symbol is used in several commands as a special "wild-card" symbol. The system would be very confused by its appearance as a file name.

Some examples of legal and illegal file names:

Legal:

CASHFLOW-3  
A  
Correspondence

Illegal:

ACCOUNTS PAYABLE (space in name)  
\* (illegal 1-character name symbol)  
  
SEVENTEEN-WAYS-TO-INVEST-MY-MONEY  
(more than 31 characters)  
  
MEMOS-IN+MEMOS-OUT  
(plus sign in name)

If you try to save a file under a file name that already exists on that disk, an error message will be displayed: "That file already exists."

### 3.2 FILE NAME EXTENSIONS

We mentioned in Section 1 that the system automatically brings BASIC into memory from the System Disk when a BASIC program file is invoked. (When you "invoke" a file, you are asking the system to bring a copy of the file into memory and then "run" the file. This assumes that the information within the file is a runnable series of instructions that form a program.) How does the system know that the file being invoked is a BASIC program? The answer -- file name extensions.

You will notice when you display a disk directory that all file names end with a two-character extension after a period.

Example:

GREETINGS.BS

These extensions give additional information to the system about file contents. You will notice that when you save a BASIC program, the file containing the program will always appear in the directory with a .BS extension. This extension tells the system that the file is a BASIC program. In the same way, other extensions provide "handles" to the system for managing other kinds of files.

The extensions that the system recognizes and uses are:

.BS	BASIC program file
.GO	Machine language program file
.TX	Text file
.DT	Data file
.OV	System overlay
.DX	Directory
.SY	Symbol Table
.PS	Printer Driver

Whenever you save one of the above types of files, the system automatically affixes the proper extension to its file name.

The system does NOT enforce the use of the above extensions. If you wish to specify any extension as part of a file name you may do so. The above extensions are the file name extensions that the system recognizes, but any two-character extension of letters and/or numbers separated by a period from a file name is legal. However, a warning: the system cannot gain any information from an unknown extension. You may, for example, save a BASIC program under a file name with any extension (e.g., TIMETABLE.F2). The system will not recognize that file as a BASIC program, however, because its name does not carry a .BS extension. If you type that file's name after a system prompt \$, you will find that BASIC is not automatically invoked to run the program because the file is not recognized as a BASIC program. You will receive an error message from the system.

.....

You now know what kind of files are stored on the disk, how to name them, and how the system labels them with extensions. You will need to know how to tell the system which file you want it to affect. Often, merely supplying the file name will not be sufficient; you will need to provide a path name. Section 4 explains how the file names are used as path names or parts of path names.



## Section 4

## PATH SPECIFICATION

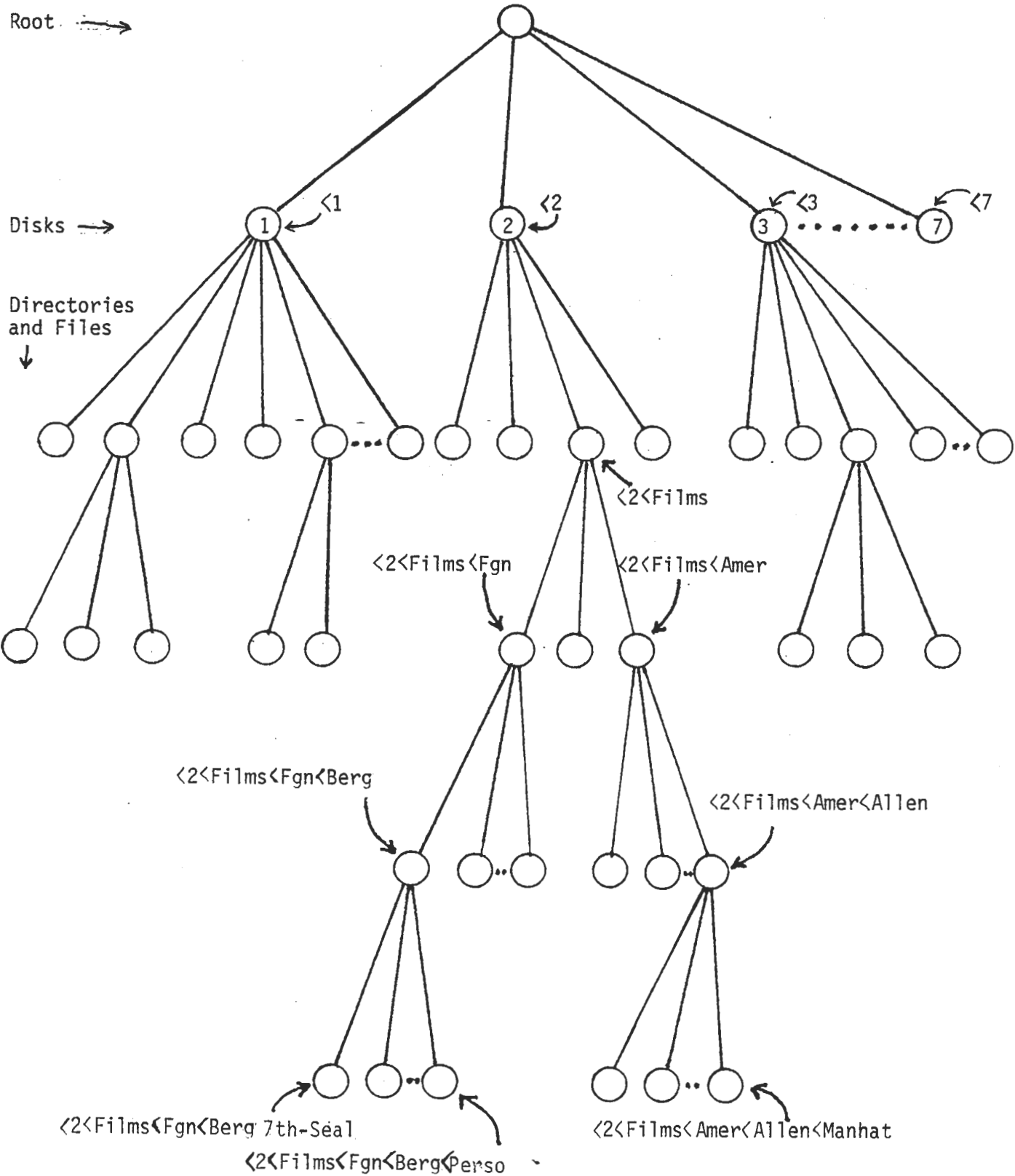
At various points in the operation of any program - such as the System-88 Exec - it is necessary for the operator to specify files, disk drives, and file directories.

A System 88 has disks, directories and files. Each of the commands that the system understands is either meant to affect one of these three entities, or it is meant to affect the whole system. Unless a command is meant to affect the whole system, its entry alone is not enough to tell the system what you want. Since there can be more than one disk, more than one directory and more than one file, there must be a method for providing a name that specifies which drive, which directory, or which file is to be affected by a command.

The entry the user types which specifies a specific drive or file or directory, is called a "path name." A path name is made up of a sequence of disk, directory or file names which we can also call component names. The sequence of component names describes the sequence of selections in directories and sub-directories that will lead to the desired disk, directory, or file.

This structure is sometimes called a "tree" because of the way the directories branch into sub-directories and sub-sub-directories. (See Figure 4.1 for clarification of directory structure.)

# PATH NAMES AND DIRECTORY HIERARCHY



To picture how the system understands the relationship between the path name components, we can imagine an internal directory which is a list of all of the main disk directories. This imaginary, internal directory can be compared to the base or root of a tree. When the component name we supply is 2 (the name of a disk drive) the system is provided with the information it needs to establish a path to a particular main disk directory; if 2 is the only component entered, the system will know to access the directory on drive 2. If our entry also contains a sub-directory, the system will then continue its path to the sub-directory we have named. If we have also supplied the name of a file which is listed in that sub-directory, the system would then know that the component we want it to access is that file and would complete the path from the root to our specified file. The last component in our entry is the name of the directory or file that we want the system to access; if a file name is a part of the path name the user enters, it must be the last component since no directory or file can branch off from a file.

We have explained that the last component name in a path name is the item which the system will attempt to access. But, what will it attempt to do to that item? This depends on what command we supply before our path name.

The following chart shows the four groups of system commands, determined by what part of the system they affect. In this section we are concerned with the first three groups since they must be accompanied by path names. The commands in the fourth column do not require any path name since they affect the entire system.

#### Types of System Commands

FILES	DIRECTORIES	DRIVES	SYSTEM
COPY	UNDELETE	INIT	ZAP
RENAME	LIST or	IMAGE	DUMP
DELETE	list	PACK	ENABLE
TYPE	DIRECTORY	ZPACK	DISABLE
PRINT	DIRECTORY	DNAME	DISPLAY
SAVE			#
ZCOPY			Printer
ZGET or			CONTINUE
GET			FLIP
EDIT			FOLD
			RESET

The path names required for each type of command are different. Commands which affect drives require a one component path name.

Commands which affect directories require a one or more component name (more than one if a sub-directory is involved). Commands which affect files require a two or more (more than 2 if sub-directories are involved) component path name. The following sections explain in detail how to enter path names.

#### 4.1 PATH NAMES SPECIFYING DRIVES

A command which is meant to affect an entire drive would need to be accompanied with a path name which specifies which drive is to be affected. A path name which identifies a drive would be made up of one component name, a single digit, as in the following example:

```
PACK <2
```

NOTE: The form for supplying the complete path name involves the use of left angle brackets (less than signs). A left angle bracket precedes the first component and left angle brackets are placed between all other components. No left angle bracket is placed at the end of the last component. There must be a space between the command and the path name. There must not be a space within the path name as that will cause the system to determine that the path name is being terminated and some other entry is beginning.

#### 4.2 PATH NAMES SPECIFYING DIRECTORIES

A command which is meant to act on directories can require one or more components, depending on whether the directory being specified is a main disk directory or a sub-directory.

For Example:

```
LIST <2
```

The above command and path name would cause the system to display the main directory for the disk in drive 2.

If we wanted to see a list of the contents of one of the sub-directories we would need to supply a path name with more than one component. Whenever there are two or more components in a path name, these components are separated from one another by left angle brackets.

```
LIST <2<Films
```

This entry would cause the system to list the directory called Films which is listed in the main directory of the disk in drive 2.

As does the above sample path, the following points out a particular directory to be affected by the LIST command.



LIST <2<Films<Fgn

However, this directory branches off from the <2<Films directory and so is the ending component of a longer path. Fgn is a sub-directory which is found in the directory called Films which is a sub-directory that can be found in the main directory of the disk in drive 2.

The final component of the path is always the name of the part that is to be affected by the command. The other components, which make up the initial stem of the path, are not affected by the command. They are necessary parts of the path because they help specify which directory is to be listed.

#### 4.3 PATHS SPECIFYING DISK FILES

The following path name specifies a text file. Its initial stem is made up of a disk and two directory specifiers. The command is of the type which affect disk files, therefore the path must include a drive specifying component; in this case, two directory specifying components and a file name.

EDIT <2<Films<Fgn<Berg

This particular entry tells the system that the file named Berg is located in the sub-directory called Fgn. Fgn is in turn listed in another sub-directory called Films, and that sub-directory can be found in the main directory of the disk in drive 2.

All paths which end with file names do not need to be as long as the above example. Only necessary specifying component names are included in a path name. For instance, if we wanted to create a text file on drive 2 called 5-27-79 and if we have no need to create a sub-directory structure we would type the following:

EDIT <2<5-27-79

Section 5 of this manual explains the reasons why you might choose to use the sub-directory capability of the system. If you do not choose to use this capability you will never need to specify directory names because specification of the drive name is equivalent to specifying the main directory of the disk in that drive.

.....

Now you understand that a full path name is the entry you type after a command which tells the system exactly what it is to affect. There are two allowed short cuts in the typing of path names. You don't need to specify the drive name when the drive you wish to access is the System Residence Drive (usually drive 1). You don't need to use the opening left angle bracket before your single component path name when you are entering commands

which affect drives or the LIST command when it is to affect the main disk directory.

#### 4.4 OMISSION OF DRIVE NAME OF SYSRES DRIVE

The first left angle bracket is the indicator that the component following it is a drive name.

Note that in sections 4.2, 4.3, and 4.4 the drive name was always the first component name in the path names. This is because the system can not begin to establish a path unless it is given a drive name (the drive names are the closest components to the root or origin of all paths).

The drive name component can be omitted when the drive being specified is the System Residence drive. The system residence drive, usually drive 1, is the location of the system disk. Since the user needs to access this drive frequently, it has been established as the default drive. If the user provides no drive name the system will assume he wants to specify the default or System Residence drive.

Therefore an entry that does not begin with a left angle bracket is transformed by the system into a complete path name -- one with the drive name appended to the front.

For example, if your System Disk is in drive 1 you could type:

```
EDIT Actors<Women<Keaton
```

This partial path name would be transformed into the following full path name:

```
<1<Actors<Women<Keaton
```

The system would access a file on drive 1 called Keaton which is contained in the directory Women, which is an off-shoot of the directory Actors which is listed in the main disk directory of that drive.

You can see that the system needs full path names in order to establish paths between the root and the specific thing you want to access. However, it knows how to transform some partial path names into complete path names.

If you typed:

```
LIST Actors<Women
```

This partial path name would be transformed into the following full path name:

<1<Actors<Women

A sub-directory on drive 1 called Women would be listed as a result of the transformation of your partial path name into a complete path name. Since the omission of the left angle bracket indicates that the Sysres drive name must be appended to the front of your partial path name, be careful not to omit this bracket by accident. If you typed the following:

EDIT 2<Actors<Women<Keaton

the system would see that you have left off the opening bracket and would assume you wanted the Sysres drive name appended. Your partial path name would become:

EDIT <1<2<Actors<Women<Keaton

Instead of opening the file you thought you had specified on drive 2, the system would look down the directory structure of drive 1. It would find no sub-directory called Actors, so it would create one. Then it would create the directory called Women and open a file called Keaton.

To avoid such a misunderstanding, remember that the system only transforms partial path names in two ways. 1) It appends the Sysres drive if there is no left bracket in your path name, unless 2) it appends only a left bracket because the particular command that you have entered is one that is to affect the entire drive. This second type of transformation is explained in the next section.

#### 4.5 OMISSION OF FIRST LEFT ANGLE BRACKET

We have explained that omission of the first left angle bracket indicates that we want to specify drive 1, but are taking a shortcut in the typing of the path name.

Some commands, those which affect drives, can be followed by incomplete path names as follows:

PACK 2

and then transformed as follows:

PACK <2

It does not become the following:

PACK <1<2

The system did not, as we suggested in 4.4, append the system residence drive to the front of this incomplete path name.

This is because in the instances where commands which affect drives are used, the system checks your path name to see if it is a drive name. If it is a drive name, the left bracket is appended to the front and the path is established. If the path name you enter is not a drive name then of course the command can not be performed.

Another example of this sort of transformation done to incomplete path names is as follows:

List

becomes

List <1

In the above entry we left off both the left angle bracket and the drive specifier, but the system knew that we wanted to list the contents of the main directory of the disk in drive 1.

In the following entry

LIST 2

becomes

LIST <2

not

LIST <1<2

Also, when a directory or file is to be specified, the following transformation might occur:

EDIT 2<File

becomes

EDIT <2<File

#### 4.6 PATH NAME LENGTH

We have already explained that partial path names are transformed into full path names -- that the system adds the initial bracket

or the bracketed system residence drive number when appropriate. When the system transforms partial path names that you enter into complete path names which will allow it to establish a path, it also adds the 3 character extension described in Section 3.

In Section 3 we also explained that there is a 31 character limit to file names. This same limit applies to path names. This means that the number of characters in all of the components of your path (The only exception is that the 3 characters necessary to specify the drive name are not counted as part of the total number of characters.) must be 31 or less.

When determining whether or not you are within the 31 character limit for path name length, you must allow for the extension the system will add to your partial path name to make it a complete path name. Don't forget that to every sub-directory and file, the system adds a 3 character extension, unless you enter it yourself.

This means that if you type the following:

```
EDIT <2<Comedian<Impersonator<Little
```

you may think that you are within the 31 character limit because you have typed only 28 characters (not including the command). However the above entry will be answered with the error message:

```
Gfid says: Name too long
```

This is because you did not allow the system to attach its extension abbreviations. It needs to affix a .DX to both Comedian and Impersonator and it needs to affix .TX to Little. This means we have created a file specifier which is 34 characters long and which cannot be processed.

#### 4.7 Special Disk Specifier Symbols

You may occasionally want to request a file, even though you're not sure what disk drive it's on. Prepared for that eventuality, the System 88 recognizes two special symbols # and ? as disk specifiers.

##### 4.7.1 The Disk Specifier Symbol ?

If you type:

```
<?<Etude
```

you are telling the system, "I don't know what disk it's on, so YOU find it." The system will first check the System Disk in for

the file Etude. If it cannot find the file there, it will go on to check the remaining drives in the system. If, after checking all the drives, the system still cannot find the file, it will give you this error message:

```
I can't find that file.
```

There are several things that you cannot do using this disk specifier symbol. You may only use this symbol as a disk specifier inside angle brackets. For example, you cannot type:

```
LIST ?
```

The system will respond with an error message: Gfid says: Bad disk identifier.

You also cannot use ? as a disk specifier when using the RENAME command or when asking the system to create a file (e.g. EDIT <2<BOOK <?<Chapter). BASIC also does not understand the ? symbol. The cases where you cannot use the ? symbol are:

```
Directory List command:
  LIST ?
  LIST <2?<File-A
File Rename command:
  RENAME <?<File <2<NewFile
SAVE command:
  Filename:<?<National-Debt
BASIC:
  LOAD, <?<BASIC-TUTORIAL
? Does not search subdirectories:
  EDIT <2?<File-A
  TYPE <2?<File-A
  PRINT <2?<File-A
```

File Copy command (when used as part of file specification of file to be created):

```
EDIT<2<APPLICATIONS <?<STUFF
```

If you are interested in more information on the system commands mentioned above, see Section 6.2, THE SYSTEM COMMANDS.

In all of the cases above, the system will refuse to fulfill your request. Usually (depending on the particular case) the system will display the error message:

```
"<?<" is not allowed here
```

#### 4.7.2 The Wild Card Path Specifier Symbol #

The # symbol is useful in specifying drive numbers and directories which we call paths. If you want, you do not have to specify a particular drive and directory when writing command files or programs. Instead, you can use the # sign. Later, when you are running the command file or program you can tell the system what the # sign stands for. The # sign can stand for a drive number, or a drive number and one or more sub-directories. This meaning attached to the # sign is called the "wild card path".

If # is to mean that the file is on drive 2 and is found under the directory "Coins" which is found under the directory "Money", we can type the following while the system is in Exec.

```
$# 2<Money<Coins
```

We have just established the Wild Card Path. Note that there is a space between the # sign and 2 and a carriage return after Coins. Now we can type the following while in Exec:

```
PRINT <#<Quarters
```

and the system will look in the drive 2 directory for a directory named "Money". In the "Money" directory it will locate another directory named "Coins" and in the "Coins" directory it will find the file named "Quarters".

The # sign works a little differently if it is used after you have used the ? as a file specifier. As we explained in the previous section, the system searches every drive for the file named after the ?. Once this procedure has been accomplished, the system remembers what drive it found the file specified with ?, and it then assumes that # means that you want to specify that same drive.

This is convenient as you do not have to force the system to check every drive when you want it to find a file as it does when you use the ? sign.

NOTE: When the # symbol is used after you have used ?, it will only stand for a drive name. If you want to cause it to represent a drive name and a sub-directory you will need to redefine #.

The symbol # can be used as a disk specifier whenever you wish to request a file on that same disk or whenever you wish to reference that disk. The # symbol is a legal disk specifier for any command or process except the INIT and IMAGE commands.

Type:

## \$DISPLAY

while in Exec and you will be shown what the system understands # to mean.

### 4.8 INVOKING FILES

You store your files on disks; you must bring a copy of a file in from the disk and put it into memory in order to do something with it. File invocation is the process of copying a file from a disk into memory and beginning execution of that file, assuming that the file is runnable (i.e., a BASIC or machine language program). (You'll recall that a program is a series of instructions followed in order by a computer.)

To invoke a file, type the path name of the file you want to bring in from a disk. This includes the disk specifier, the directory specifier (if the sub-directory structure is being used) the file name, and optionally, the file name extension (e.g., <2<ACRONYM.BS). If you do not include the extension, the system will look for the first file it can find on the specified disk with the given file name. It will try to bring this file in, regardless of its extension.

This could create a problem. If you have two files, ACRONYM.BS and ACRONYM.TX, make sure either that you include the file name extension when invoking the BASIC program file, or that ACRONYM.BS occurs on the specified disk BEFORE the file ACRONYM.TX. Otherwise, you might wind up trying to invoke the text file (ACRONYM.TX), which is not runnable because it is full of words instead of machine language instructions.

There is one interesting example of a machine language program file invocation that you have run up against before in Section 1. That is the invocation of BASIC itself. To enter the language BASIC, you type (after a system prompt) BASIC and a carriage return. Although it might appear to the casual eye that we are giving the system a command, we are in fact invoking a machine language program file. The file BASIC resides on the System Disk. Because this disk is always in the default drive, no disk specifier has to be included in the BASIC file specification. When we type BASIC, we are really implying <l<BASIC. The system brings the file BASIC.GO into memory from the System Disk, and begins program execution. BASIC itself is merely another machine language program file, although it is also a system file. (Please remember that when we speak of "BASIC," we are talking about the computer language BASIC itself, and not a program WRITTEN in BASIC.)

More comprehensive information on the invocation of files is presented in Sections 7, BASIC PROGRAM FILES and 8, MACHINE LANGUAGE PROGRAM FILES.



## Section 5

## THE DISK DIRECTORY

## 5.0

The first four sectors of every disk are reserved for the main disk directory, which contains a list of files and sub-directories on the disk. The main disk directory is displayed by use of the LIST or DIRECTORY commands accompanied by the drive name of the disk to be listed. The directory also carries within it information not seen in the display of the directory, but necessary to the functioning of the system. After discussing the information maintained in the directory that you DO see, we'll discuss the information that you don't.

When a disk directory is displayed, you might see something like the following:

```
Disk GUTNBERG has 20 files on it.
240 sectors in use, 2 deleted
  Size Name
  10 CHECKERS.BS
 140 Article:lst-Draft.TX
  50 CHECKBOOK.F2
```

## 5.1 THE DIRECTORY HEADER

The first two lines of the disk directory display are the directory header. The header tells you a) the name of the disk whose directory you are displaying; b) the number of files in the directory c) the number of sectors used by those files; and, d) the number of sectors making up deleted files (inactive, inaccessible files).

## 5.2 THE DIRECTORY PROPER

The rest of the directory listing displays the names and sizes of all its active (undeleted) files. The size of a file is the number of disk sectors used by that file. Under certain conditions, this section of the directory can list other information about the files on the disk as well. (See Section 6.2.1, ENABLE and DISABLE for more details.)

## 5.3 "INVISIBLE" FILES

If you add up the number of files listed in the directory (three), you will see that that number is not equal to the number

of files given in the directory header (20). You will also note that the number of sectors in use mentioned in the header does not agree with the number of sectors in use you can see displayed in the directory proper. Obviously there are files on the disk that are not listed in the directory. What are these "invisible" files?

There are three classes of files that are not displayed in the disk directory--deleted files, other directories and their respective files, and system files.

Deleted files are files that you have decided you no longer need. You have deleted them by using the DELETE command (Section 6.2.3, DELETE), or the system deleted them automatically to make way for revised versions when you used the Editor. Deleted files, although no longer listed in the directory, are not physically gone from the disk. The number of sectors that they use is reflected in the directory header both in the total number of sectors in use and the number of sectors in use by deleted files. You may not access deleted files (that is, find the file on the disk and copy information from it). If you attempt to bring a copy of a deleted file into memory, the system will tell you "I can't find that file." To bring a deleted file back into active, accessible status, use the UNDELETE command (Section 6.2.4, UNDELETE). To actually remove deleted files from a disk, use PACK. (Section 6.2.5, PACK).

Another type of file not listed in a directory is the system file. BASIC itself is a system file. Such a file cannot be deleted or renamed. A special type of system file, an overlay, cannot be brought into memory or executed except by the system itself, and cannot have its contents displayed. A system overlay file is always found on the System Disk.

Later in the manual (Section 6.2.1, ENABLE and DISABLE), we will discuss how you may see system files listed in a disk directory display.

The third type of file that can be on the disk but not listed in the main directory is the sub-directory. of course sub-directories can be listed in the main directory, however, those first level sub-directories might contain more sub-directories which are not listed in the main disk directory but in one of its off-shoot directories. No files that are found in a sub-directory would be listed in the main disk directory. The number of sectors occupied by these files and sub-directories which are listed in a sub- directory, are included in the main disk heading as a part of the total sectors in use figure.

#### 5.4 UNDISPLAYED DIRECTORY INFORMATION

Besides the file information talked about above, the directory carries within it certain information about disk files that is not displayed, but IS used by the system. The directory "knows" if a file is a system file.

The directory also carries within it information about the status of a file: 1) is the file deleted or active?, and 2) is the file "new" (is the file an original, or a copy of another file)? Such information provides information to the system it needs to handle files.

## 5.5 SUB-DIRECTORIES

All main disk directories take up 4 sectors on their disk. This means that you can store a maximum of about 60 file names (The total number of file names that can be accommodated by one directory varies depending on the length of each file name.) in that main disk directory. If you have a double density 8813, or an 88/MS add-on you may very likely find that a 4 sector area is not enough room to house the names of all the files you would like to be able to store on a single disk.

This limited directory factor is not a problem because many sub-directories can be stored on your disk as well. This means that your main disk directory may list the names of other directories. These sub-directories are also limited to 4 sectors. However, one sub-directory name on your main disk directory could be the parent directory for as many sub-directories as you can access legally given the 31 character limit (see Section 4 on providing legal path names.)

**WARNING:** If you have a double-density system use the subdirectory capability to avoid overfilling your directory. If the name of a file that you have opened will not fit in your directory, you will lose your text or data!

The sub-directory structure also allows the convenient grouping of files so that those which are obviously a sub-group of a larger category can be grouped together under it. The following are examples of sub-directory use. Note that if no sub-directory exists on the disk, you can create one by providing a path name with all of the desired sub-directory names and the file name that is to be stored.

```
$EDIT Film\Fgn\Berg\7th-Seal
```

```
$EDIT Film\Am\Wood\Manhat
```

```
$EDIT Film\Fgn\Berg\Perso
```

```
$EDIT Film\Fgn\Wood\Annie
```

If we were to store information or reviews on films we might create the the above path names. The major category is FILM, which is listed in the main disk directory. Then under that main category there are two main divisions, FGN (foreign) and AM (American). Within each of those categories are the names of individual film artists, WOOD (Woody Allen) and BERG (Bergman).

All of these headings so far mentioned become the names of sub-directories. The last headings, ALLEN and BERG, each contain files which are named according to film titles. These files contain the actual information which we are attempting to store. We could also create a file under FGN or AM which could contain general information on Foreign or American films. This sub-directory tree structure allows for convenient storage which pertains to the actual way that these types of information relate to each other. This sort of grouping together of information also allows easy transfer or deletion of several related files at once (see section 6 on the use of COPY and DELETE).

Sub-directories are stored on the disk like files but you can tell that a particular item listed in the main directory is a sub-directory because it has a .DX extension.

Section 4 explains how to provide path names for files and sub-directories, and Section 6 shows how COPY, DELETE, RENAME, and UNDELETE work both with files listed in the main disk directory and with sub-directories and their off-shoot files and directories.

## Section 6

## THE EXECUTIVE

Whenever you see the system prompt \$ or \$\$, you are talking with the part of the disk operating system called the Executive (Exec). Exec handles all communication between the system and you--it processes the words that you type and responds to them either with the appropriate action or with an error message. (When you are communicating with BASIC or the Editor, your commands are processed by them and not by Exec, since you are not actually communicating with the system.)

You will probably refer often to sections 6.2, SYSTEM COMMANDS, and 6.3, REFERENCE LIST OF SYSTEM COMMANDS, since they contain information that you will use frequently in operating the system. Sections 6.1 and 6.4 discuss some of the internal workings of Exec (e.g., how Exec processes input, what happens within the system at start-up time, etc.). This information is not strictly necessary for operating the system; you may find, however, that you enjoy understanding HOW the system works and that knowing why the system responds to your instructions the way it does will allow you to make better use of it.

## 6.1 HOW EXEC PROCESSES INPUT

After Exec issues a system prompt \$, it waits for you to give it an order. When it receives an input (the words that you've typed from the keyboard), it follows three steps in deciding what to do. It decides if the input is: 1) a system command; 2) a file invocation; or 3) a command file. The procedure it follows in processing the input is to ask the following questions:

Is this a command?

Exec looks at the first word typed and determines if it begins with a disk specifier (a number enclosed in angle brackets). If a disk specifier is present, Exec knows that a file is being invoked, and Exec then moves on to the next step in processing the input. Otherwise, Exec compares the word to the commands in its list of legal commands. If it finds a match, it takes the appropriate action in response to the command. If no match is found, Exec decides that a file on the System disk has been invoked (which requires no disk specifier), and Exec moves on to the next step in input processing--determining if the file invocation is legal.

Is this a file invocation?

If Exec does not find a file of the given name on the disk specified, it puts an error message on the screen. If such a file DOES exist, Exec does the following:

Is this file a machine language program?

If the file appears to be a runnable file, and not a system overlay, Exec brings a copy of the file into memory and tries to run it. If the file is a system overlay, the message "I can't run that file" appears on the screen. If the file is not a machine language program, Exec moves on to the next step: determining if the file is a BASIC program.

Is this file a BASIC program?

If the file carries the file name extension .BS, and otherwise appears to be a BASIC program, Exec brings the language BASIC into memory and "tells" it to run the program. If the file is NOT a BASIC program, Exec moves on to the next input processing step.

Is it a command file?

If Exec is able to find a file but is not able to run it as a BASIC program or a machine language program, Exec will try to use the file as a command file. A command file is one from which the system draws instructions. More details on constructing a command file, ending the use of a command file, and determining what commands are legal as command file items will be found in Section 9, COMMAND FILES. For now, the important thing to remember is that from the time a command file is invoked until its use is ended, the system takes instructions entirely from that file, without knowing whether those instructions come from a command file or from the keyboard.

If Exec is not successful in using a command file (if, for instance, an unknown command or unrecognizable file name is found in the command file) use of that file will be ended, and an error message will be displayed: "(Cmdf abort)."

From the information above, it is clear that if given an input that it cannot recognize, Exec will assume that you have given it a file invocation for a non-existent file. If, for example, you type "ZZZ" and a carriage return, you will receive the message "I can't find that file."

## 6.2 SYSTEM COMMANDS

Some of the commands that Exec recognizes, we have encountered before in this manual. Those commands will be covered more fully here.

NOTE: Every command must be entered in upper case and after a system prompt. You must type a carriage return at the end of every command line. Only as many letters need be typed as will uniquely identify that command. (In other words, you only have to type as many letters as are required to distinguish the command from another in Exec's command list.) All of the entries below will match the word "IMAGE" in Exec's command list:

```
$$IMAGE
$$IMAG
$$IMA
$$IM
```

### 6.2.1 ENABLE and DISABLE

The computer ordinarily operates in what is called the disabled mode. Another mode of operation, the enabled mode, is intended for use by those who have a considerable degree of familiarity with computers. The experienced user is considered familiar enough with assembly language and system operation to find interesting and to benefit from an expanded disk directory display and access to the front panel display.

#### 6.2.1.1 Disabled Mode

When the system is turned on or restarted, it comes up in disabled mode. This state is indicated by the appearance of a single dollar sign prompt \$. All of the situations and commands that we have discussed up to this point occur when the machine is in disabled mode. When in disabled mode, you are able to write programs, save programs, and otherwise use the system fully.

#### 6.2.1.2 Enabled Mode

When the system prompt is a single dollar sign symbol \$, you know that the machine is operating in the disabled mode. To enter enabled mode, type:

```
ENABLE
```

Exec will respond with a double dollar sign prompt \$\$, indicating that you are now operating in enabled mode.

One option open to you when the system is in enabled mode is entry to the front panel display. Type a Control-Z (hold down the CTRL key and type a Z). A software simulation of a traditional computer front panel will appear on the monitor

screen. The information shown in the front panel display is explained in Appendix E, The 4.0 Monitor and the Front Panel Display. Briefly, however, what you are seeing is the contents, expressed in machine language, of a section of your machine's memory. This can be extremely useful for the machine language programmer, and rather disconcerting for the non-machine language programmer. You can use the directions given in Appendix E to modify the contents of the memory locations shown in the front panel display.

To return to the system from the front panel display, type a G. Entry to the front panel display is prevented, whether in enabled or disabled mode, whenever disk I/O (disk read and write operations) is taking place.

Another advantage to operating the system in enabled mode is the additional information provided when a disk directory is displayed. This information, like the information provided in the front panel display, is likely to be of benefit only if you are an experienced machine language programmer.

A typical disk directory display, when the system is in disabled mode, might look something like this:

```
Disk CATCHWORD has 12 files on it.
155 sectors in use, 0 deleted.
Size  Name
10   SKETCH.BS
5    MAIL.BS
6    Graphics.GO
```

Three files seem to be missing from the display of the directory: The directory header tells us that we have six files on the disk, but only three are listed. To find those "missing" files, we must operate in enabled mode.

When the system is in enabled mode, the display of the partial disk directory looks like this:

```
Disk CATCHWORD has 12 files on it.
155 sectors in use, 0 deleted.
Size Addr   La   Sa  Name
6     4   2000  2000  Exec.OV
2     A   2000  2000  Test.OV
10    C     0     0   SKETCH.BS
5    16     0     0   MAIL.BS
6    1A   3200  3200  Graphics.GO
32   20   3200  3200  BASIC.GO
```

Now we see the three files missing from the first directory display. They are: one system file (BASIC.GO) and two system overlay files (Exec.OV and Test.OV). The file name extension .OV indicates an overlay file. The directory tells us the name and size (in sectors) of the files on the disk. In enabled mode it



also tells us the disk addresses of the files, and their load and start addresses (explained below).

Let's discuss each of these new items of information separately. You will need the following information if you plan on doing machine language programming; if you intend to program primarily in BASIC, skip over the next few paragraphs and return to them at your leisure.

#### Disk address:

We already know that there are sections on disk called sectors. The information that we store on a disk is written into these sectors. The directory has always told us how many sectors were used by each file. Now we can see at exactly what sector on the disk each file begins and ends. Notice that size of the files is given in ordinary decimal notation (base 10). The disk addresses look rather strange, however. They are given in hexadecimal (base 16) rather than ordinary decimal (base 10) numbers. It turns out that it is extremely useful to have some information in hexadecimal. This is because when we talk to the machine in machine language we talk to it in hexadecimal numbers, which it translates into binary numbers (base 2).

Note that the first disk address is 4. Remember that sectors 0-3 (we start counting with a zero) are reserved for the disk directory itself.

#### Load address (La):

The load address is a hexadecimal number which designates a memory location address. This address tells the system where in the main unit's own memory to start writing a copy of the disk file when that file is invoked.

#### Start address (Sa):

The start address is also a hexadecimal number representing a memory location address. The "start" address is the beginning location of the machine language program, and is the address the system goes to when starting execution of that program.

Note that the load address is non-zero only for machine language program files (including system files and system overlay files). Load and start addresses of 2000H are reserved for system overlays. A zero load address indicates BASIC programs and text files.

To reenter disabled mode from enabled mode, type:

DISABLE

Exec will respond with a single dollar sign prompt \$, indicating that the machine is in disabled mode.

### 6.2.2 LIST, DIRECTORY and CHECKSUM

The contents of a disk is discovered by looking at the disk directory--the disk's "table of contents." If the sub-directory capability is being used, then there may be more than one directory that has to be listed in order to view all of the contents of a disk.

The LIST command is used only to display disk directories, and means "LIST the directory of the disk in drive n."

In order to see a disk's main directory, type:

```
$LIST n
```

where n is the disk specifier (drive number) selecting the disk whose directory you wish to display.

EXAMPLE:

```
$LIST 3
```

The directory of the disk in drive 3 will be displayed. As always, no disk specifier need be given for a disk in the default drive, drive 1.

You may also use the special disk specifier symbol #. (See Section 4.7, Special Disk Specifier Symbols, for more information on the use of this symbol.)

EXAMPLE:

```
$LIST #
```

where # has previously been defined as a valid drive number by the use of the ? disk specifier symbol.

Sometimes so many files are listed in the directory that the complete directory display will not fit on the screen at one time. If this occurs, the system will display a dot on the screen at the bottom of the directory display. To continue the display, type any character. Typing a Control-Y will end the directory display at any time.

If you have a printer, you may use the DIRECTORY command to list the disk directory out to your printer. Once you are sure that your printer has been interfaced to the system software, (see Section 13, THE PRINTER DRIVER) type:

```
$DIRECTORY n
```

where n is the disk specifier (drive number) selecting the disk whose directory you wish to see printed. You may also use the special disk specifier symbol # and abbreviate the command.

EXAMPLE:

```
$DIR #
```

If you want to list the contents of a sub-directory, you must supply a path name which contains the drive name and the sub-directory name.

EXAMPLE:

```
$LIST <2<sub-dir-a
```

Note that you can not leave off the opening bracket when listing a sub-directory as you can when providing a main disk directory path name. If the above path name were typed without the opening bracket, the the Sysres drive name would be appended to the front of the path name causing the drive 1 directory tree to be searched for the desired directory.

CHECKSUM is a file on the system disk which can be invoked by typing:

```
$CHECKSUM DIRECTORYNAME
```

while the system is in DISABLEd or ENABLEd mode. This file will cause the specified directory to be displayed. This directory will look like a regular directory listing but there will be an additional column which provides the CHECKSUM for each file. When you first receive your system disk, you should record the CHECKSUM for each file. Later if you think you have a problem with the system software, you can use a CHECKSUM command to view the directory of the file or files in question. If any of the numbers in the CHECKSUM column have changed, then some part of the software has been altered and may be causing problems. You can, ofcourse, use this program on files which you create. Record their CHECKSUM when they are working correctly. Later you can use this command to see if the CHECKSUM has changed.

Since CHECKSUM is a file on your system disk, you can delete it if you want. We do not reccommend that you do, however, since you will then be unable to list the CHECKSUM of your files when you are having problems.

For an explanation of the information to be found in the disk directory display, see Section 5, THE DISK DIRECTORY, and 6.2.1, ENABLE and DISABLE.

### 6.2.3 DELETE

The meaning of the DELETE command is simply "Tag the following

files or sub-directories to be removed from the disk on the next PACK operation, and don't display these files or sub-directories in the disk directory."

You may recover deleted files by using the UNDELETE command.

To delete a file provide the DELETE command and the file's path name. You may include as many path names (separated by commas) within one DELETE command as will fit on one line of the screen.

To delete files, type:

```
DELETE Pathname,Pathname,...
```

EXAMPLE:

```
$DELETE <2<ELECTRONS,<3<CHESS.UG,<3<Clock.TX
```

If you give no extension in a file specification, DELETE will delete the first file of that name that it finds on the given disk, regardless of its extension. If an extension IS given, DELETE will delete only a file of the given name with that particular extension.

Whenever a file is deleted, the directory header is updated to reflect the new number of deleted sectors on the disk. The figure indicating the number of files on the disk stays the same, until the deleted file is actually removed from the disk.

Gone But Not Forgotten: Use of the DELETE command is only the first step toward removing a file from a disk. Once a file is deleted, it is no longer listed in the disk directory display. It can no longer be invoked. However, it is not physically gone from the disk. It can, in fact, be "undeleted" (see Section 6.2.4, UNDELETE). The final process in removing files from a disk involves the use of the PACK command (Section 6.2.5, PACK). Only then is a deleted file truly gone from a disk.

If Exec is unable to delete one of the files mentioned in the DELETE command, it ignores the rest of the file specifications after that point on the line. A system prompt will be displayed, indicating that Exec is waiting for another command.

EXAMPLE:

```
$DELETE <2<ACCOUNTS.DT,<3<Addresses.TX,<9<RECIPE,<2<Tax
```

Exec was not able to delete <9<RECIPE (invalid disk specifier). <2<Tax will not be deleted, even though it may be a deletable file, because it follows a file specification that caused an error.

One of two error messages will be delivered to you on the screen if Exec has not been able to delete a file: I can't find that file, or I can't do that to a system file.

If files CAN be deleted, a message will appear on the screen listing the files deleted.

EXAMPLE:

```
$DELETE 3-D-MAZE,<2<AIR-ROUTES
```

```
<1<3-D-MAZE.BS deleted.
```

```
<2<AIR-ROUTES.TX deleted.
```

### 6.2.3.1 Delete a Sub-Directory or A File in a Sub-Directory

To delete a sub-directory, type its correct path name.

EXAMPLE:

```
DELETE $<2<Film<Fgn<Berg
```

The sub-directory named Berg, an off-shoot of the sub-directory Fgn found in the main directory in drive 2, would be deleted as would all of the files listed in the Berg directory.

EXAMPLE:

```
$<2<Film<Fgn<Berg<7th-Seal
```

The above is the path name of a file in the Berg directory; this file and any others listed in Berg will be deleted when Berg is deleted.

If we typed:

```
DELETE $<2<Film<Fgn
```

the directories called Fgn, Berg, the file called 7th-Seal and any other directories or files branching off of Fgn would be deleted.

You can not DELETE an entire main disk directory. DELETE only affects individual files, and sub-directories and their dependent files. Since DELETE is a command which affects specific files on a disk and not the entire disk, you must supply the opening left angle bracket and drive name if you do not want the Sysres drive name to be appended to the front (See Section 4.4).

```
$DELETE 2<Film<Fgn
```

The above partial path name would cause the system to search the drive 1 directory tree which for a file or sub-directory called Fgn.

### 6.2.3.2 Using DELETE with \*

An additional feature of the DELETE command is its use of a "wild card" symbol, an asterisk (\*). In cards, a wild card is a special card that can be used to match any other card. In the same way, the DELETE wild card symbol matches any file name or extension it is paired with.

#### EXAMPLE:

```
$DELETE <2<*.TX
```

The above command will delete ANY files with the extension .TX from the disk in drive 2 (other than system files), not just the first file found with that extension.

#### EXAMPLE:

```
$DELETE <3<PAYROLL.*
```

The DELETE command above will cause to be deleted any files found on the disk in drive 3 with the name PAYROLL, regardless of their extensions.

One can see how the wild card symbol can be very useful. Suppose that you have an entire family of programs that handle company payroll. The programs are all named PAYROLL, but have different extensions, depending upon their functions (.TX for text files giving explanations on using the programs; .DT for payroll data files; .BS for the actual payroll calculating programs; etc.). All of the PAYROLL programs are on one disk. You may decide that you want to eliminate all of the PAYROLL data files in order to replace them with new ones. Use \*.DT in a DELETE command. (Of course, this will delete ALL files on the disk with the extension .DT.) You may decide that you want to delete all PAYROLL programs, regardless of their extensions. Use PAYROLL.\* in a DELETE command.

One wild card combination prohibited is \*.\*. This file specification in a DELETE command would result in all deletable files on the given disk being deleted--a dangerous capability. To completely erase a disk, use the INIT command (see Section 6.2.15, INIT).

You may also use the special disk specifier symbols ? and # as part of the file specifications given in the DELETE command.

## EXAMPLE:

```
$DEL <?<Diet, <#<Calendar
```

or, if # has been defined as <2<Film<Fgn, typing the following:

```
$DEL <#<Berg
```

will be the same as typing:

```
$DEL <2<Film<Fgn<Berg
```

and will cause Berg and all of its off-shoot files and sub-directories to be deleted

(See Section 4.7 for information on the ? and # disk specifier symbols.)

## 6.2.4 UNDELETE

The UNDELETE command simply says "I've changed my mind. I want the file or files specified by my path name to be undeleted.

The procedure for returning deleted files to active status involves the use of the UNDELETE command with a directory path name. The UNDELETE command operates on ALL deleted files listed in the directory specified by the directory path name.

If the files you want to UNDELETE are listed in a main disk directory, then you only need to specify the drive name in your path name.

## EXAMPLE

```
$UNDELETE 2
```

Notice that we did not provide the opening left angle bracket. The system will not append the Sysres drive name to our partial path name because the files we want to UNDELETE are listed in a main disk directory. Whenever a command is being used which affects an entire main disk directory, the system refrains from appending the Sysres drive name if it sees an unbracketed drive name. Instead, it appends the opening left bracket to the above partial path name. Since the above command causes all files listed in drive 2 to be undeleted, its entry would also cause sub-directories listed in drive 2 to be undeleted. Remember that when a sub-directory is undeleted, all of its off-shoot directories and files are also undeleted.

If we wanted to UNDELETE a sub-directory which is not listed in a main disk directory, our path name would have to specify the drive name and the component names leading to the sub-directory we wish to UNDELETE.

## EXAMPLE:

```
$UNDELETE <2<Film<Fgn<Berg
```

Of course, any off-shoot files and directories of Berg would also be undeleted.

Remember that even though you can DELETE a specific file, by typing its path name after the DELETE command. You must UNDELETE it by typing the path name of the directory which contained it.

## EXAMPLE:

```
$DELETE <2<Film<Fgn<Berg<Persona
```

would cause the file named Persona in the Berg directory to be deleted. To UNDELET this same file we must type the following:

```
$UNDELETE <2<Film<Fgn<Berg
```

then Persona and any other DELETED files under Berg would be undeleted.

You may also use the special disk specifier symbol # in the UNDELETE command. (See Section 4.7.2 Wild Card Path Specifier)

## EXAMPLE:

```
$UNDELETE #
```

would UNDELETE all files listed in the directory indicated by the path name by which # is defined.

As files are undeleted, a message appears on the screen for each file, indicating that it has been undeleted, and giving its full file name.

## EXAMPLE:

```
$UNDELETE 2
```

```
Biplane.TX undeleted.
```

```
Botany-Notes.TX undeleted.
```

```
TEXT-FORMATTER.GO undeleted.
```

Warning: You already know that you cannot create a disk file with the same name as another active file on that disk (that is, the same file name AND the same extension). But you CAN create a file with the same name as a deleted file. Let's say that you have a BASIC program named FACTORIAL. You have deleted that



program file and saved a new version of the program on the same disk as FACTORIAL. You decide that you want to bring back into active status several previously deleted files. You use the UNDELETE command, and find that you suddenly have TWO files with the same file specification on the same disk. To get out of this predicament, use either the RENAME command to rename the first duplicate file on the disk (remember that RENAME will rename the FIRST file on the disk it finds with the specified name) or the DELETE command to delete the first duplicate file on the disk (DELETE also will delete the first file of the given name that it finds on a disk).

#### 6.2.5 PACK

To physically erase deleted files from a disk, use the PACK command. PACK erases all deleted files, and moves the rest of the files up ("packs" them together and pushes them toward the beginning of the disk) so that no empty areas are left between files. This reclaims the space used by deleted files and causes the directory heading to be updated to reflect the new number of files existing on the disk and to show that there are no deleted sectors.

Type:

```
$PACK 2
```

Since PACK is a command which affects the entire disk in the specified drive, there will be no Sysres drive name appended to a partial path name which does not have an opening left bracket.

The Sysres drive name will be appended if we type the following:

```
$PACK
```

You may also use the special disk specifier symbol # in the PACK command if the # has been defined as a drive name only.

EXAMPLE:

```
$PACK #
```

If # is defined as <2, the disk in drive 2 will be PACKed.

Once a disk has been PACKed, all the deleted files are irretrievably gone. Take care when you use PACK; especially, double-check to make sure you are PACKing the right disk.

When you use PACK, an automatic ZAP is also performed. That is PACK it re-zeroes memory before it has finished its last data transfer. Therefore make sure before you use PACK that you do not have any data in memory that you need to retain.

### 6.2.6 TYPE and PRINT

It is often desirable to display the contents of a particular file. The TYPE command puts a copy of the asked-for file on the monitor screen. To display a file, type:

```
$TYPE Pathname
```

EXAMPLE:

```
$TYPE <3<DESIGN-SPECS.BS
```

The BASIC program file DESIGN-SPECS will be brought into memory from the disk in drive 3 and displayed on the screen.

The path name you give TYPE may contain the disk specifier symbols # and ?. (See Sections 4.7.1 and 4.7.2)

EXAMPLE:

```
$TYPE <?<PHOENIX
```

TYPE "pages" the file display by stopping at every 14th carriage return symbol that it encounters within the file. That is, fourteen lines of information (one screen) are presented each time you hit a key. A dot is then shown, and TYPE waits for any character except an X before going on to the next "page" of the file. If you type an X, the file display is ended, and you are returned to Exec. A Control-Y command also terminates the TYPE display. To print a copy of a file on a printer, use the PRINT command. Making sure that your printer has been interfaced to the system software (See Section 13, THE PRINTER DRIVER), type:

```
$PRINT Pathname
```

You can interrupt the printing of your file with Control-Y if you decide that you do not want a full listing of the file. A word of warning. You can use TYPE or PRINT on any kind of file. What happens if you display a file that isn't in text form (data translating to keyboard-type letters and symbols)? A machine language program file, for example, is stored in machine language, a kind of data that isn't in text form. The system will take the file and try to convert what it finds in it to standard keyboard symbols. You will see an interesting but illegible display of what TYPE or PRINT thinks is text data, but isn't. Therefore, use TYPE or PRINT only on files which are stored in text form (BASIC data files, text files, BASIC program files, etc.).

You may not use PRINT or TYPE on a system file. If you try to do so, the system will display the error message: I can't do that to a system file.

### 6.2.7 PAGE

If you have a printer "hooked up," the PAGE command causes it to feed one form-- move from the present page in the printer to the top of the next.

### 6.2.8 COPY

To copy a file from one disk to another, or simply to make a copy of a file on the same disk as the original file, use the COPY command. Type:

```
$COPY (original) Pathname (new) Pathname
```

EXAMPLE:

```
$COPY <2<REVIEW <3<REVIEW
```

The above command will copy the first file on the disk in drive 2 with the name REVIEW. The copy will go on the disk in drive 3, under the name REVIEW.

If you are COPYING a file which is listed in a sub-directory to another disk where that sub-directory does not exist, a sub-directory structure will be created on that new drive which will accommodate that file.

EXAMPLE:

```
$COPY <2<Film<Fgn<Berg<7th-Seal <3<Film<Fgn<Berg<7th-Seal
```

The above entry will result in the opening of the sub-directories called Film, Fgn, and Berg on drive 3. And, of course 7th-Seal will then exist on drive 3 as a file in the Berg directory.

You may use the disk specifier symbols ? and # when using the COPY command, as long as you do not use the ? symbol in the path name of the new file to be created.

EXAMPLE:

```
Legal: $COPY <?<COMMON-COLD <#<Rare-Flu
```

```
Illegal: $COPY <2<Rational-Numbers <?<Sane-Numbers
```

A file may not be created with the same name as another active file on the same disk. If you had typed:

```
$COPY <2<REVIEW <2<REVIEW
```

an error message would have been displayed: "That file already exists." You may, however, make a copy of REVIEW on its own disk if you use another name.

## EXAMPLE:

```
$COPY <2<REVIEW <2<REVIEW-2
```

To create a new file with the same name as the original file, you may use the wild card symbol \*.

## EXAMPLE:

```
COPY <3<FORMATTING <2<*
```

You may not use COPY on a system file. If you try to do so, Exec will display the message: I can't do that to a system file.

To substantially speed up COPY, see Section 6.2.18.1, Using ZAP to Speed Up COPY.

## 6.2.9 RENAME

It is possible to rename files without making additional copies of them. Type:

```
RENAME original Pathname.extension new Pathname.extension
```

## EXAMPLES:

```
$RENAME <2<BILLS.TX <2<BILLS.DT
```

```
$RENAME <3<TIME-SCHEDULE.BS <3<DEADLINES
```

If you do not supply an extension for the new file name, the original file's extension will be used for the new file name. If you do not supply the extension of the original file, the system will rename the first file of the given name found on the disk.

The RENAME command only involves the changing of the original path name. One component of THE PATH NAME THAT YOU CAN NOT CHANGE IS THE DRIVE NAME. If you attempt to do this, you will be told that "you can not RENAME across directories" since there is no file to RENAME on the new drive. If you want a file on drive 3 that is just like the one on drive 2 but with a different path name, you will have to use the COPY command.

When a file is renamed, a message will appear on the screen telling you so, and giving the full old and new file path names.

## EXAMPLE:

```
$RENAME <2<Results-of-Experiment.DT
<2<DataResults-of-Experiment.DT renamed to Data.DT
```

The RENAME command uses the wild card symbol \* in the same way that the DELETE command does. The symbol \* may represent one of

the components in a path name or an extension. Thus you can rename several files with one RENAME command.

EXAMPLE:

```
$RENAME <2<RESULTS.* <2<DATA.*
```

```
RESULTS.DT renamed to DATA.DT
```

```
RESULTS.TX renamed to DATA.TX
```

```
RESULTS.WW renamed to DATA.WW
```

```
$RENAME <3<*.DT <3<*.F1
```

```
GRAPH.DT renamed to GRAPH.F1
```

```
INVENTORY.DT renamed to INVENTORY.F1
```

You may not, however, use a wild card symbol for the path name of one of the files mentioned in the RENAME command if you use a wild card symbol for the extension of the other file name given.

EXAMPLE:

```
$RENAME <2<MOUSE.* <2<*.TX
```

The above command is confusing to the system; if it obeyed the command, you could wind up with several files with the same name AND with the same extension. The system will not obey the command above.

If you are using the sub-directory structure be sure that you follow the following rules about renaming sub-directories.

You can provide a RENAME command to change one sub-directory name at a time.

EXAMPLE:

```
$RENAME <2<DOCTORS <2<Physicians
```

For the above command to work DOCTORS must be a sub-directory that is listed in the main disk directory on drive 2. If you want to change the name of a sub-directory that is an off-shoot of DOCTORS type the following:

```
$RENAME <2<DOCTORS<SURGEONS<HEART <2<DOCTORS<SURGEONS<BRAIN
```

Notice that when you are renaming sub-directories, the only sub-directory that can be renamed is the last one in your path

name. You can not ask the system to do the following:

```
RENAME <2<DOCTORS<SURGEONS<HEART <2<DOCTORS<SPECIALISTS<BRAIN
```

If you were to attempt such a renaming request, the following error message would be presented:

You can not RENAME across directories.

You also can rename files that are part of a sub-directory structure by providing their path name and the new path name that you want.

EXAMPLE:

```
RENAME <2<FILM<FGN<BERG<PERSONA <2<FILM<FGN<BERG<FACES
```

You will then see the following message:

```
PERSONA.TX renamed to FACES.TX
```

Notice that we changed only the last component name in the new path name.

Do not use the wild card symbol \* in the renaming of sub-directories or of files which are listed in sub-directories.

System files and system file overlays cannot be renamed.

You cannot use the disk specifier symbols ? and # when renaming a file.

#### 6.2.10 SAVE

The SAVE command is used to make a disk file that is a copy of an area of the machine's memory. The command is almost always used to save on a disk a copy of a machine language program that is in memory. Type:

```
SAVE
```

followed by a carriage return. SAVE will then ask you several questions. (Note: your answer to each question is ended when you type a carriage return.) The questions are:

- 1) From address:
- 2) Load address:
- 3) Start address:
- 4) Number of sectors (1-7FH):

### 5) Filename:

These questions presuppose a certain amount of knowledge on your part about machine language and memory. You must know WHAT area of memory you want to save (the beginning memory location and the number of bytes you want to save). You also must know where in memory your program should be copied into when you bring this file back into memory.

We will discuss each of the SAVE questions separately.

#### From address:

This address is the beginning memory address that SAVE is to begin copying from. You answer this question with the hexadecimal (base 16) number which is that memory location's address. Note: You know that a number is in hexadecimal form when it is followed by an H (e.g., 3200H).

If the first byte of your program is at location 40A0H, you would answer the above question 40A0 followed by a carriage return.

#### Load address:

The load address is the first address of the area of memory that you want this program to be copied into whenever it is later brought in from the disk. The answer to this question is the hexadecimal number that is that memory location's address.

The memory contents you want to save are in a certain area of memory, beginning with the memory address you gave as an answer to the "from address" question. It may be that, later on, when you invoke this file that you are saving, you do not want the program copied back into the same memory area that it was originally read from. By giving different load and from addresses, you can save a program that starts at memory location 9000H (for example). Then whenever you bring that file back into memory, you can have the program placed in memory beginning at location 3200H.

#### Start address:

The start address is the memory address that will contain the beginning instruction of your machine language program when it is loaded. A hexadecimal number is expected as the answer to this question, representing the address of that memory location.

Later on, when you want to use this file that you are saving, and it is loaded into memory at the load address you have supplied, the system will try to begin program execution. You must specify a start address in response to the above question in order for the system to execute your program starting with the first instruction.

Number of sectors (1-7FH):

You must give a hexadecimal number representing the number of disk sectors you want to reserve for the file you are creating. This number must be between 1 and 7F (7F=127 in decimal). With this number, you are in effect telling SAVE how many data bytes you wish saved. There are 256 bytes per disk sector.

Filename:

You must state the name you want to give to this file you are saving. A disk specifier must be included, so that SAVE will know on what disk to place this file. (You may not use the ? disk specifier symbol (See Section 4.7.1) An optional extension may be included. If you do not specify an extension, the system will affix a default extension of .GO.

After you answer the last question (and have typed a carriage return), the contents of the specified area of memory will be written onto a disk as a file. If for some reason you decide that you really don't want to go through with the SAVE command, type a Control-Y command to abort the SAVE command and return to Exec.

## 6.2.11 GET

It is often convenient to bring a machine language program file into memory without beginning program execution. GET simply says, "Get a copy of a file and put it into memory." Program execution does NOT begin. In order to place a program into memory without executing it, type:

GET Pathname

## EXAMPLE:

\$GET <3<PERMUTATIONS.GO

The file PERMUTATIONS is now in memory beginning at its load address.

You may use the disk specifier symbols ? and #. (See Sections 4.7.1 and 4.7.2)

## EXAMPLE:

\$GET <?<Bridge/16

GET may be used to bring any machine language program into memory. You can, for example, bring BASIC itself into memory.



## EXAMPLE:

```
$GET BASIC
```

Although BASIC is now in memory, you will have to use the START command to start the execution of BASIC.

An error message will be generated if you try to bring in a file that is not a machine language file.

You might wonder about the use of a command which only brings a machine language program into memory without executing it. It is often very useful to combine more than one machine language program within one disk file. The only way to do that is to bring in copies of ALL of the programs into memory (by using the GET command), and then save their combined memory locations as one file by using the SAVE command. This presupposes, of course, that none of the memory locations of the programs overlap, and that the programs are designed to execute together.

Another reason for bringing a machine language program into memory without executing it is to make it accessible for "debugging." "Debugging" is the term programmers use for the process of tracking down problems in a program. You can bring your program into memory by using the GET command, then debug it using the front panel display (see Appendix D, The 4.0 Monitor and the Front Panel Display).

## 6.2.12 START

When a machine language program is placed into memory by way of the GET command, the START command must be used to begin execution of that program. Type:

```
$START
```

The program in memory will start running.

## Caution:

Most machine language programs are saved with load and start addresses of 3200H. This is simply because it is a convenient place to start loading programs, since it is the start of "user memory" (that is, the area of memory set aside for your use).

The system begins program execution by jumping to a "start" address. This means that the system starts program execution with the instruction at that address. When you invoke a machine language program by giving its file specification to Exec, the system "knows" the file's start address, which may be any memory location

address. When you use START to begin program execution, a start address of 3200H is assumed. This means that if you use START on a program, the program's start address must be 3200H, because that's where the system is going to jump to in order to start executing your program. If your program really starts at some other location, the system will still try to execute whatever random numbers are at location 3200H.

So if you plan to use GET to place a program into memory, and you want to execute that program, make sure that the program's start address is 3200H.

### 6.2.13 REENTER

The REENTER command is used for programs that have a "warm-start" address. Many programs contain a section of code which is executed only when you want some program variables cleared or set to their initial values. This is generally done the first time through the program. After that, you usually do not want to execute that area of code again. The first time you execute the program, you want the system to jump to the start address. Thereafter, you want an address further along in the program (past the initializing code) to be used as the start address. This is the "warm-start" address.

The purpose of the REENTER command is to give you the capability of beginning program execution at the program's warm-start, rather than start, address.

There is a distinction between using REENTER and START in the ENABLED and DISABLED modes. When the system is DISABLED, both commands check to see if there appears to be anything in memory. If there does not appear to be anything in memory both commands send the following message:

Nothing to Run!

When the system is ENABLED, the START command runs whatever program is located at 3200H and the REENTER command runs whatever is located at 3203H. To use REENTER, type:

```
$$REENTER
```

and a carriage return.

There's a catch.... As in the above START command, some assumptions are made concerning the start and warm-start addresses of machine language programs brought into memory. The REENTER command assumes a warm-start address of 3203H. REENTER is going to jump to address 3203H, because it assumes that your warm-start instructions are going to be there. If your program is at 8000H, the system is going to try to execute whatever is at memory location 3203H. The results could be very strange indeed, and certainly not what you are expecting.

#### 6.2.14 CONTINUE

You use the CONTINUE command when you want to resume execution of a machine language program that has been interrupted. For instance, if you have interrupted the running of your machine language program with a Control-Y, you can resume that program's execution by typing:

```
CONTINUE
```

after a system prompt \$. You will probably use this command most often to return to BASIC from the system level. (Remember, BASIC is itself a machine language program.) For example, you may exit from BASIC by using the EXEC command (see Section 7, BASIC PROGRAM FILES). To return to BASIC, use the CONTINUE command. IF you had a program in memory when you exited from BASIC, you will find that program still intact when you return to BASIC after using the CONTINUE command.

#### 6.2.15 IMAGE

NOTE: In order to use this command, the machine must be operating in enabled mode.

You can duplicate the contents of one disk on another by using the IMAGE command.

Type:

```
$$IMAGE
```

You will then be asked two questions:

From which drive?

IMAGE expects the number of the drive holding the disk you want copied.

To which drive?

You must tell IMAGE which drive is holding the disk upon which you want the copy to be placed.

EXAMPLE: \$\$IMAGE

```
From which drive? 3
To which drive? 2
```

A complete verbatim copy of the disk in drive 3 will be made on the disk in drive 2. Even the disk name will be the same.

You cannot use the disk specifier symbols ? and # when imaging a disk. (See Sections 4.7.1 and 4.7.2) You may not make a copy of disk's contents on that disk itself; an attempt to do so will cause a confused Exec to give you the error message: What?

Because imaging a disk onto the disk in the System Drive would destroy the contents of your System Disk, Exec will not allow you to do that, and will issue the following error message:

I can't do that to the System Drive!

Because every write operation is verified, imaging a disk will perform a simple disk surface test of the disk that is being written to: if the disk contains a "bad" location, an error message will be displayed, "Verify error."

Note: Before you use IMAGE, make sure that you do not need to keep any of the data that is in memory. This includes programs, text, data, etc. IMAGE uses all of memory from the start of user's memory up to the end of memory to hold the disk contents that is transferring to the new disk. This makes IMAGE very fast but destroys any data that you might have had in memory before you used IMAGE.

#### IMAGING A DATA DISK ON A TWO DRIVE SYSTEM:

Type:

\$\$IMAGE

When you see:

From which drive?

Do not type the number of the drive from which you wish to image. Instead, remove the system disk and insert the data disk to be IMAGED. Make sure that a blank, inited disk (to be IMAGED onto) is inserted in drive 2.

Now type 1 after "From which drive?" and 2 after "To which drive?".

WARNING: Do not answer the "From which drive?" question before removing the system disk and inserting the data disk that is to be IMAGED.

After the IMAGED function is complete, you will be asked to insert a system disk and type a RETURN.

#### 6.2.16 INIT

NOTE: The machine must be operating in ENABLED mode when you use this command. INIT cannot be used from a command file.

INIT initializes a disk. This means that INIT erases all information on the disk. You also give a name to the disk at this time. Before using a disk for the very first time, or to

completely erase a disk already in use, you must initialize it.

Type:

```
$$INIT
```

followed by a carriage return.

INIT will ask:

```
Which drive?
```

Enter the number of the drive containing the disk you wish to initialize. You may not use the disk specifier symbols ? and #. (See Sections 4.7.1 and 4.7.2) Nor can you initialize your System Disk (the disk in the System Drive). Exec will display the error message

```
I can't do that to the System Drive!
```

if you try to do so.

INIT then will display the following message:

```
(Cleaning disk)
```

At this point INIT is writing zeros everywhere on the disk, completely erasing everything on it. This process performs a rough disk surface test, since an error message ("Verify error") will be generated if INIT is not able to write a zero in a particular place.

INIT will then ask:

```
Disk name (up to 8 characters)?
```

You must state a disk name no more than eight characters long. This name will appear in all subsequent listings of the disk directory.

Any disk may be initialized at any time. Any information on that disk will, however, be gone irretrievably.

#### 6.2.17 DNAME

You may rename any disk by using the DNAME command. Type

```
DNAME
```

followed by a carriage return. You will then be asked two questions:

```
Which drive?
```

DNAME expects the number of the drive containing the disk you

want to rename. If DNAME accepts your disk specifier as valid, it will then ask for the new name of the disk:

New diskname:

Give it a name of eight characters or less. That will now become the new name of the disk. A disk name may include any of the keyboard characters including upper and lower case letters.

#### 6.2.19 ZAP

Use the ZAP command to write zeroes in every memory location from the start of user memory (3200H) to end of memory (the last memory location available on your machine). Type:

\$ZAP

followed by a carriage return.

Why would you want to write zeroes throughout memory? When you turn on or restart your machine, even though you may not have put anything into memory, there is still non-zero data there. A given section of memory may contain what looks like random numbers. Often you want to make sure that there is NOTHING in memory except your program. For instance, if you are "debugging" a machine language program that generates data and stores it in memory, you would like to be able to see where your data ends and empty memory begins. If you "zap" memory before you invoke your program, you know that your data ends where the zeroes begin.

##### 6.2.19.1 Using ZAP to Speed Up COPY

The COPY command will be speeded up if it is preceded by the ZAP command. The reason for this is that COPY uses memory to transfer data, and the more memory available for it to use, the faster it will run. If memory has been ZAPPED, all of user memory may be used, thus COPY will be able to run at optimum speed. In fact you may find that COPY can work over twice as fast after the ZAP command has been used. This increased speed is dependent on the amount of memory you have in your machine and the size of the file to be copied.

If you have used ZAP before using COPY, it will re-zero memory after it has finished its last data transfer. Hence you should make sure before you use ZAP, either alone or in combination with COPY, that you do not have any data in memory that you need to retain.

### 6.3 REFERENCE LIST OF SYSTEM COMMANDS

Commands available at the system level are listed below. For information on general BASIC commands and BASIC file commands, see the System Library volumes BASIC: A Manual and The System Programmer's Guide.

NOTE: Remember that all commands must be typed in upper case, and a carriage return must be typed at the end of a command line.

The word Pathname, used in the following command definitions, indicates that an entry specifying which file or disk or directory is to be affected by the command. Sometimes pathnames will need to contain more than one component name. (See Section 4) The term "command syntax" is another way of saying, "This is the proper form in which to type the command."

#### THE COMMANDS

##### CHECKSUM (page 49)

Lists the specified disk directory with CHECKSUM for each file. Command Syntax: CHECKSUM Pathname (of directory)

##### CONTINUE (page 65)

Resume execution of a machine language program. Command syntax: CONTINUE

##### COPY (page 57)

Copy a file. Command syntax: COPY original Pathname copy new Pathname

##### DELETE (page 51)

Delete files and sub-directories from disk. Using wild card symbol is legal. Command syntax: DELETE Pathname,Pathname,....

##### DIRECTORY (page 49)

Print a disk directory on a printer. Command syntax: DIRECTORY (Pathname of directory to be printed)

##### DISABLE (page 45)

Put machine into disabled mode. System prompt for this mode is \$. Command syntax: DISABLE

DISPLAY (page 39) While in Exec you will be shown what the system understands # to mean. Command syntax: DISPLAY

DNAME (page 67)

Rename disk. Command syntax: DNAME

ENABLE (page 45)

Put machine into enabled mode. System prompt for this mode is \$\$\$. Command syntax: ENABLE

GET (page 62)

Place copy of machine language program file into memory but do not execute program. Command syntax: GET Pathname

IMAGE (page 65)

Duplicate a disk by copying entire disk. May be used only when system is in enabled mode. Command syntax: IMAGE

INIT (page 66)

Initialize a disk. May be used only when system is in enabled mode. Erases a disk by writing zeroes in every sector. Sets disk name. Command syntax: INIT

LIST (see page 49)

Display a directory. Additional information is given when the system is in enabled mode. Command syntax: LIST Pathname

PACK (page 55)

Pack disk, reclaiming sectors held by deleted files. This is the only way to actually remove deleted files from a disk. When the system PACKs a disk, it also does a ZAP. Be sure that you do not need the information in memory. Command syntax: PACK Pathname

PAGE (page 57)

Send form feed to printer. If text is present after PAGE, it is printed starting at the top of the next page. Command syntax: PAGE

PRINT (page 56)

Print the contents of a file on a printer. Command syntax: PRINT Pathname

REENTER (page 64)

Warm-start machine language program currently in memory. A warm-start address of 3203H (hexadecimal) is assumed. Command syntax: REENTER



RENAME (page 58)

Rename a disk file or sub-directory. Using wild card symbol is legal. Command syntax:

RENAME (original) Pathname,(new) Pathname

SAVE (page 60)

Save a machine language program as a disk file. You must provide hexadecimal "from," "load," and "start" addresses, and the number of sectors to be used. Command syntax: SAVE

START (page 63)

Start execution of a machine language program currently in memory. Start address of 3000 (hexadecimal) is assumed. Command syntax: START

TYPE (page 56)

Display the contents of a file. Command syntax: TYPE Pathname

UNDELETE (page 53)

Undelete all deleted files on a specific disk. Command syntax: UNDELETE Pathname

ZAP (page 68)

Puts zeroes in all memory locations from 3200H to top of memory. Command syntax: ZAP

## 6.4 SYSTEM START-UP

Every time you push the Load button you restart the disk operating system. This section discusses what happens at the time of system start-up. Although not necessary for operation of your system, this information will give you some understanding of the internal processes within the system.

### 6.4.1 Start-Up of the Disk Operating System

We have mentioned quite a few times in this manual that you must place the System Disk in the System Drive, usually drive 1. This is because Exec has been told that drive 1 is the System Drive. If you attempt to do anything that might destroy the System Disk in the System Drive, Exec will refuse to perform that action and will give you an error message: I can't do that to the System Drive! For example, Exec will not let you image another disk onto the disk in the System Drive or initialize the System Disk.

The System 88 central processor card, the heart of the system's main unit, contains three ROMs (Read-Only-Memories). A ROM is a section of memory that cannot be erased or written into by the user. The first ROM contains the 4.0 Monitor (a section of code which performs some basic system functions and which is used by the disk operating system). (See Appendix D, 4.0 Monitor and the Front Panel Display, for information on the Monitor.) The second and third ROMs contain the core of the disk operating system. The use of ROMs maintains the validity of the disk operating system and system monitor, since ROMs cannot be erased or altered. At the time of system start-up, the monitor performs its basic system functions and begins execution of the disk operating system.

The operating system then brings into memory and executes the copy of Exec on the System Disk. Before Exec prints anything on the screen, it does two things: 1) it notes the highest usable memory location on the system (known as the "top of memory"), and 2) it searches for a file named INITIAL on the System Disk (see Section 6.4.2, INITIAL Program).

Exec has noted and displayed the address of the last "good" memory location. Supposing for now that no program named INITIAL has been found, Exec will print the following message on the screen:

```
(Exec/##-top of RAM is ####)
$
```

The number after the word Exec is the version number of your system software.

The four-digit number in the message refers to the address of the

last usable memory location Exec found. If the system is behaving normally, and memory is good, this hexadecimal number will end in a string of Fs (e.g., 5FFF, BFFF, or 9FFF).

A system prompt \$ is also displayed on the screen, indicating that the system is now ready for your commands. Note that the prompt is a single dollar sign symbol, indicating that the system is operating in disabled mode.

#### 6.4.2 INITIAL Program

If Exec had found a program named INITIAL after determining the top of memory, you would not see a system prompt or a screen message. Instead, the system would automatically execute the file named INITIAL. An INITIAL file is invoked in the same way as any other file: 1) if INITIAL is a runnable machine language program, it is brought into memory and executed; 2) if it is a BASIC program, BASIC is automatically invoked, and BASIC then runs the program; 3) if it is a command file, the system draws its instructions from it until the use of the command file ends.

The reason for the existence of a program called INITIAL is to let you decide what the user of your machine will first see when that machine is turned on or restarted. For example, let's say that you have the following BASIC program saved on the System Disk under the name INITIAL:

```
10 PRINT "This system currently contains bookkeeping",
20 PRINT " programs." \PRINT
30 PRINT TAB(10), "To see what programs you can run,"
40 PRINT TAB(10), "put the disk labeled CONTENTS in"
50 PRINT TAB(10), "drive #2. Now type 'L 2' and a"
60 PRINT TAB(10), "carriage return."
70 REM The next program statement types BYE command
80 REM and carriage return ("CHR$(13)") for you, so
90 REM that you exit BASIC after this program executes.
100 OUT 0, "BYE"+CHR$(13)
```

Whenever someone starts up or resets your machine, the first thing that he will see will be:

This system currently contains bookkeeping programs.

To see what programs you can run, put the disk labeled CONTENTS in drive #2. Now type 'L 2' and a carriage return.

>BYE

\$

You can save any type of program you desire under the name of INITIAL.



## PART II

## BUILDING AND USING FILES

## INTRODUCTION TO PART II

You now know how to operate the system. For the fullest use of the system, you should know how to create your own disk files.

The purpose of this section is to provide you with the knowledge you need to build and use files of several different sorts: BASIC programs, machine language programs, command files, and BASIC data files. Some of the information in Sections 7 and 8 you have already seen. It is provided again here so that each section of the manual is largely complete in itself.

Although these sections tell you how to save and invoke BASIC and machine language files, they do not tell you how to write the programs themselves. For information on BASIC, see the System Library volume, BASIC: A Manual. For information on using the system Assembler, see Section 12, THE ASSEMBLER.



## Section 7

## BASIC PROGRAM FILES

BASIC is perhaps the most widely used of all computer languages in the field of micro-computers. Many applications programs and games are written in BASIC, and beginning programmers generally find programming in BASIC much easier than programming in assembly language.

If you want information on programming in BASIC, there are many good books designed to teach it. For a full explanation of the BASIC that is part of your system, see the System Library volume BASIC: A Manual.

This section discusses what to do with your BASIC program after you've written it. You will learn how to save your program as a disk file and how to bring that file into memory from the disk.

## 7.1 BRINGING A BASIC PROGRAM FILE INTO MEMORY

You may bring a BASIC program file into memory in either one of two ways: invoking it at the system level or, while in BASIC, using the BASIC command LOAD.

## 7.1.1 Invoking BASIC Program Files at the System Level

To invoke a BASIC program at the system level: after a system prompt \$ or \$\$, type the pathname of the BASIC program file you wish to invoke (disk specifier, file name, and, optionally, the file name extension .BS).

## EXAMPLE:

```
$<3<MATRIX.BS
```

You may use the ? and # disk specifier symbols when invoking from the system level.

## EXAMPLE:

```
$<?<Hertzsprung-Russell
```

When we speak of being "at the system level," we mean that you are communicating with Exec (indicated by a system prompt \$ or \$\$), and not BASIC, the Editor, or the Assembler.

Exec will process your file invocation in the following way: Exec decides that your input is not a system command. It then finds your file on the disk specified. Exec then must decide if your file is truly a BASIC program or some other type of file (such as

a text file). Exec looks at the extension of your file's name. If the extension is .BS, Exec knows that a BASIC program is being invoked. A copy of the language BASIC is brought into memory automatically. After BASIC is brought in, Exec "tells" it that it has a BASIC program to run. When BASIC is invoked in the normal way (that is, when you type BASIC and a carriage return after a system prompt), a screen message appears telling you which version of BASIC you have invoked and how much room is left in memory. When the system invokes BASIC because you have asked the system to run a BASIC program, the screen message does not appear. Instead, BASIC immediately loads your program into memory and begins execution of that program. Execution of that program will begin whether or not the program was saved in "auto-execute" (self-starting) mode or regular execution mode. (See Section 7.2, SAVING BASIC PROGRAM FILES, for information on saving BASIC programs in auto-execute and regular modes.)

After your program is finished running, you are still in BASIC if you see the BASIC user prompt > or >>. To exit from BASIC, use the commands BYE or EXEC.

It is clear from the information above that Exec uses the extension of your file's name to identify that file as a BASIC program file. You may save a BASIC program as a file with any extension you might want to invent. However, you will not be able to invoke a BASIC program file from the system level unless that file carries the extension .BS. (That is, the system will not invoke BASIC and instruct BASIC to run your program if the file name of your program does not include the extension .BS.)

You can also run your BASIC program by typing

```
$BASIC pathname
```

to the Exec. BASIC will then run the program you named.

#### 7.1.2 Bringing BASIC Program Files into Memory While in BASIC

You may also bring a copy of a BASIC program file into memory while you are communicating with BASIC. After a BASIC prompt > or >>, type:

```
LOAD,pathname
```

followed by a carriage return. You may not use the ? and # disk specifier symbols when you are communicating with BASIC at the > or >> levels. If you try to do so, BASIC will tell you: "<?<" not allowed here. (See Section 4.7, Special Disk Specifier Symbols.)

EXAMPLE:

```
>LOAD,<3<MAILING-LIST
```

BASIC will respond with a prompt if it has found the file and has



put a copy of it into memory. Then type RUN and a carriage return. If BASIC is unable to find the file, it will display an error message: "I can't find that file." If BASIC has found the file, and if the program in it has been saved in auto-execute mode, the program will begin running automatically without the use of the RUN command.

We have seen in Section 7.1.1 on invoking BASIC program files from the system level that Exec cannot identify a BASIC program file as such unless it carries an extension of .BS. When you are in BASIC, BASIC itself does not require that a program file carry the extension .BS. However, if a BASIC program file does NOT have an extension of .BS, you MUST specify the correct extension in the file specification given in the LOAD command. Otherwise, the BASIC program file will not be found.

EXAMPLE:

You have a BASIC program file named BANKING.RT on the disk in drive 2. You may not invoke this file from the system level. If you type <2<BANKING.RT or <2<BANKING after a system prompt, you will get the error message "(Cmdf abort)I can't find that file." The system displays this error message because it cannot identify the program as a BASIC program (no .BS extension). Exec realizes that the program is not a machine language program. It then tries to use the file as a command file. The first line of the file is a BASIC program line, however, and not a legal system command or file invocation. The system ends the use of the file as a command file-- hence the message "(Cmdf abort)"-- because it cannot understand the first line of that file. It then concludes "I can't find that file." You can use BASIC BANKING.RT to run your file. Or if you are in BASIC, you can bring a copy of this file into memory by way of the BASIC LOAD command. After a BASIC prompt, type LOAD,<2<BANKING.RT. You MUST specify the extension, since it is not the expected extension of .BS. If you type LOAD,<2<BANKING you will get the error message: "I can't find that file."

## 7.2 SAVING BASIC PROGRAM FILES

BASIC programs may only be saved as disk files from within BASIC. In other words, you must be working in BASIC and use the BASIC SAVE command to save a BASIC program file.

To save a BASIC program that is now in memory (having just been written or brought into memory from a disk), type SAVE followed by a comma or semi-colon (for auto-execute mode) and the file specification you wish to use for the new file.

EXAMPLE:

```
>SAVE,<3<DATA
```

This program will now be saved as a disk file on the disk in drive #3, with the name DATA.BS. (If you do not specify a file name extension when saving a BASIC program, the system will affix the default extension of .BS to the name of your file.)

We have mentioned "auto-execute" mode before. Remember that a program saved in auto-execute mode begins execution as soon as it is loaded into memory by BASIC--no RUN command is needed. BASIC knows that if a semi-colon follows the word SAVE, the program is to be saved in this mode.

EXAMPLE:

```
>SAVE;<2<Finance
```

If you display a BASIC program saved in auto-execute mode by using the TYPE command, you will see the simple method BASIC uses to set up an auto-execute program file--the letters RUN appear as the last characters in the file.

EXAMPLE:

```
TYPE <3<COUNTER.BS
```

```
10 I=0
20 I=I+1
30 PRINT I
40 IF I<100 GOTO 20 ELSE PRINT "END"
RUN
```

The presence of the RUN command at the end of the file ensures that BASIC will execute the program file.

### 7.3 EXITING FROM BASIC (THE USE OF EXEC, BYE, AND CONTINUE)

To exit from BASIC (that is, to return to communicating with the Exec), you use one of two commands: BYE or EXEC after a BASIC prompt > or >>.

The purpose of the EXEC command is to let you leave BASIC, perform a few simple Exec functions (with the exception of ZAP and PACK which do an automatic ZAP) and then return to BASIC.

EXAMPLE:

```
>10 PRINT "THIS IS A BASIC PROGRAM"
>EXEC
(Exec/##)
$DELETE <2<ORBIT.BS
<2<ORBIT.BS deleted.
$CON
>LIST
>10 PRINT "THIS IS A BASIC PROGRAM"
```

The Exec commands that you can use in this manner are: PRINT,

TYPE, DELETE, PACK, COPY, RENAME, LIST, DIR, UNDELETE, ENABLE, DISABLE, PAGE, and SAVE.

When you do not need to return to BASIC, use the BYE command. When you use this command, all data files are closed. Use of the CONTINUE command will not return you to BASIC. You can invoke BASIC in the normal way (by typing BASIC and a carriage return), but you will find that any BASIC program that you had in memory is no longer there.

#### 7.4 BASIC PROGRAMS AS SYSTEM FILES

A BASIC program may be marked a system file. BASIC programs so marked are protected in the same way that other system files are: the commands COPY, DELETE, RENAME, TYPE, and PRINT will not work on them. Also, within BASIC, the only legal commands are RUN (without a line number), SCR (scratch), and BYE, to exit BASIC. New lines may not be entered into the program, and the program cannot be SAVED from BASIC. If you want to mark BASIC program files as system files, see the System Library volume The System Programmer's Guide.



## Section 8

## MACHINE LANGUAGE PROGRAM FILES

## 8.0

We have talked about machine language programs before, and we have already seen a large example of a machine language program file--the file containing BASIC itself. This section discusses in more detail: what a machine language program is, how to bring into memory a machine language program file, how to execute and "warm-start" such a file, and how to save a machine language program as a disk file. It also gives information on a special machine language program, the system file.

The next few paragraphs contain introductory material. If you are an experienced programmer, you may want to skip over them. If you are new to machine language programming, you will need to know the following information.

When you communicate with your machine, everything that you type is translated into ones and zeroes (binary data). Even letters have a binary representation. The code that your machine uses to represent keyboard symbols is called ASCII. A capital A, for example, in the ASCII code is the binary number 01000001 (65 in decimal or 41 in hexadecimal).

Long strings of 1s and 0s are very cumbersome. We can reduce this problem somewhat by using hexadecimal numbers (base 16). Hexadecimal numbers turn out to a convenient representation of binary data for humans to use: Data is stored in your machine in groups of eight "bits" (1s and 0s), called "bytes." Because of this grouping, two hexadecimal digits represent one byte of data. When it is necessary to indicate that a number is in hexadecimal rather than decimal form, the hexadecimal number is followed by a capital H. The number 41H, then, is the number 41 in base 16. (41H is 65 in decimal.)

Among other things, these numbers stored in the memory of your machine may represent characters (in ASCII code). A BASIC program, for example, is stored in text form--that is, it is stored in memory as ASCII code values representing the keyboard characters used to make up the lines of the program. When data in memory does not represent a character, it may be a machine language instruction or a value used by a program. (It is important at this point to acknowledge the ambiguity in our use of the word "data." For many people, the word "data" has come to mean specifically information that is generated by a computer program or is used by that program. According to that definition, data is a different kind of thing from the program

that uses it. This manual usually uses the word in its more general sense-- that is, as a synonym for information. Given this usage, ALL of the numbers stored within memory and on disks as files are data, whether they represent characters, BASIC programs, machine language programs, information used by programs, etc.)

Since the machine only understands numbers, it is not clear how so complicated a thing as a BASIC program is ever acted upon by the machine. At the machine's most primitive level, it understands only some 78 instructions, each represented by a different number. The most complicated thing that these instructions can do is to move numbers from one location in memory to another and perform simple arithmetic operations on them. The instructions are called machine language instructions. A series of these instructions forms a machine language program. BASIC itself is a machine language program. Its function is to take your BASIC programs, interpret them, and respond to them with the appropriate action. A machine language program interpreting a program in text form (e.g., BASIC interpreting a BASIC program) naturally works much slower than a machine language program working alone.

Constructing a machine language program would be a slow and frustrating process (number after number after number) if it were not for "assembly language" and "assemblers." Assembly language is the language that programmers use to write machine language programs. It allows us to use words to represent the numerical machine language instructions that we want to use. It is obviously much easier to remember the instruction JMP (jump) than the hexadecimal number C3H. The assembler is the program (itself a machine language program) that translates our assembly language program into machine language code. Within our actual machine language program file, therefore, there are no "words" like JMP, only numbers like C3H.

The advantage of machine language programs over higher level language programs (such as BASIC programs) is that they take up quite a bit less room within memory and execute or run much faster. On the other hand, many people consider writing in assembly language more difficult than writing in a high-level language such as BASIC because the former requires an understanding of the logical structure of the machine.

For information on the use of the System 88 assembler, see the System Library volume The MACRO-88 Assembler.

### 8.1 BRINGING A MACHINE LANGUAGE PROGRAM FILE INTO MEMORY

There are two methods of bringing a machine language program file into memory from a disk: invocation and use of the GET command. The first method brings in a copy of the file and attempts to begin program execution. The second brings a copy of the file into memory but does not run it.

### 8.1.1 Invoking Machine Language Program Files

You are now familiar with examples of file invocation from earlier sections of the manual. As you will recall, to invoke a file, type the full file specification after a system prompt (disk specifier, file name, and optional file name extension).

Whenever a machine language program is saved as a disk file, its "load" and "start" addresses are stored within the disk directory (see Section 6.2.10, SAVE). When you give Exec a file invocation, such as:

```
$<3<FLOW-CHART
```

it looks at the directory of the disk in drive 3. It makes sure that the filename has the ".GO" extension and hence is a runnable machine language program. It then copies the disk file into memory beginning at the file's load address, and executes it. The start address tells the system the memory location containing the program's starting instruction. (Files with .OV extensions are machine language, but are not runnable in this sense; they are used internally by the system.)

If Exec has determined that the file is not a machine language program file, the file will be invoked as a BASIC program or a command file. If none of these attempts to run the file is successful, an error message will be displayed.

### 8.1.2 Bringing Machine Language Program Files Into Memory:

GET, START, and REENTER

You may sometimes want to bring machine language files into memory without executing them. This can be done by using the GET command. Type:

```
GET filename
```

followed by a carriage return. You may use the ? and # disk specifier symbols when you use the GET command. (See Section 4.7, Special Disk Specifier Symbols.)

EXAMPLE:

```
$GET <2<CATALOG
```

Exec will look for the file CATALOG on the disk in drive 2. If the file is a runnable machine language file, and NOT a system overlay file, a copy of the file will be put into memory at the file's load address. Program execution will not begin, however, until START or REENTER is used.

Machine language program files (except system file overlays) are usually saved with load and start addresses of 3200H. This means that when these files are brought into memory, they are copied

into memory beginning at location 3200H. Program execution is begun by jumping to the instruction at location 3200H. Memory location 3200H is simply a convenient place to put programs, because it is the start of the user area of memory. When you use the START command to begin execution of a program, a start address of 3200H is assumed. The REENTER command is used for programs with a warm-start address. The warm-start address is an alternative start address used when you wish to skip over the section of your program that assigns initial values to program variables. REENTER assumes a warm-start address of 3203H.

## 8.2 SAVING MACHINE LANGUAGE PROGRAM FILES

The use of the SAVE command is discussed in great detail in Section 6.2.10, SAVE. However, it is important to note that ONLY machine language program files may be saved by way of this command. When determining the load and start addresses that you wish to use when saving a program, remember that START and REENTER assume start and warm-start addresses of 3200H and 3203H respectively. If you plan to use GET, then START or REENTER, make sure that your load and start addresses correspond to these locations.

## 8.3 SPECIAL MACHINE LANGUAGE FILES (SYSTEM FILES)

We have mentioned system files and system overlay files often in earlier sections of the manual. To recapitulate:

System files are files recognized by the system as special files. You may not use the following commands on a system file: COPY, DELETE, RENAME, TYPE, PRINT, or EDIT.

BASIC is an example of a system file, but any type of file may be made a system file. The System Programmer's Guide provides a way for you to designate your own files as system files. If you try to rename or delete a system file, the system will display the error message: "I can't do that to a system file."

A system overlay file is a type of system file. However, it is a very special kind of file, since it constitutes part of the actual system itself. Only the system itself can execute an overlay. If you try to invoke an overlay, you will get the error message: "I can't run that file."



## SECTION 9

## COMMAND FILES

A command file is simply a text file from which the system draws commands and instructions. The text entries in a command file may be any legitimate inputs: system commands (LIST, TYPE, etc.) or file invocations (including invocations of BASIC program files, machine language program files, other command files, etc.).

Exec ignores any line in a command file beginning with a semi-colon followed by a space, TAB or a CR. This allows you to place comments in your command file. BASIC, on the other hand, will not accept any input beginning with a semi-colon. Therefore, make sure your command file comments do not occur among statements which will communicate with BASIC.

## EXAMPLE:

## A Typical Command File

```
$TYPE <2<COMMAND-FILE.TX
; Make a copy of a BASIC program
COPY <2<CHECKBOOK.BS<3<CHECKBOOK1.BS
; Run that program
<3<CHECKBOOK1.BS
BYE
; Delete the copied file
DEL <2<CHECKBOOK.BS
```

When you invoke a command file, the system starts reading the command file entries. Neither BASIC nor Exec have any way of knowing if instructions are coming from a command file or directly from the keyboard. When you use a command file, then, the effect is just as if you were typing in all of the commands yourself from the keyboard. After performing the action asked for by the first entry, the system returns to the command file and reads the next statement in the file. This continues on until: a) the system reaches the end of the command file; b) an error occurs; c) you type a Control-Y (interrupting the system); or d) the system comes upon the command INIT within the command file.

When the use of a command file is ended by the occurrence of an error, the error message "(Cmdf abort)" is displayed.

The kinds of things that can cause an error to occur when a command file is in use are: a) the system doesn't recognize a

command given in the file (e.g., DALETE <2<PLOTTER.GO, instead of DELETE <2<PLOTTER.GO); b) the system can't find a file mentioned in a file invocation; c) the next entry processed is a BASIC command, but you are presently at the system level, communicating with Exec; or conversely d) you are in BASIC, and the next command file entry processed is a system command.

Command files are very useful when you want a sequence of events to occur, but typing in the necessary commands and file invocations would be tedious and time-consuming. For instance, you could set up a command file that would bring in every file on one disk one by one and copy each one to another disk under a new name. Or you might have a program or a set of programs that performs a lengthy series of statistical calculations. Rather than wait for a long time to simply enter the name of another statistics program, you can easily set up a command file that will invoke a series of programs to run one after the other.

You may even have a command file invoke itself. For example, given the command file <3<DEMO-FILE.TX:

EXAMPLE:

```
TYPE <3<DEMO-FILE.TX
```

```
; This command file will run a series of BASIC
; programs. The user of this file must type a
; Control-Y
; to end the demonstration.
```

```
;
BASIC
LOAD,<2<STANDARD-DEVIATION.BS
LOAD,<2<ANALYSIS-OF-VARIANCE.BS
LOAD,<2<STUDENTS-T-TEST.BS
BYE
```

```
; Note: The above BASIC programs do not ask for
; user input. If they had, the system would look in
; the command file for the required input and would
; have found either the BASIC commands BYE or LOAD.
; The use of the file would have been aborted, since an
; input error would have been caused within BASIC.
```

```
;
; The file now invokes itself, to keep the above
; process running until you type a Control-Y.
```

```
;
<3<DEMO-FILE.TX
```

Constructing command files consists of nothing more than placing text into a file. This may be done either by way of the system Editor or from within BASIC. To use the Editor, see Section 11, THE EDITOR. To construct text data files in BASIC, see the volume BASIC: A Manual on building basic data files.

One important point to remember when constructing command files is to remember whom you're speaking to, Exec or BASIC. If one

line in the command file is "BASIC," make sure that from that point until you exit from BASIC with a BYE statement, all command file entries are legitimate BASIC commands. By the same token, do not use BASIC statements when the command file is communicating with Exec.

#### Examples of Incorrect Command Files:

```
; use of this command file will be ended
; when an error is generated--
BASIC
;
; you are now talking to BASIC, but BASIC doesn't
; accept a line beginning with a semi-colon. An
; error message will be displayed:
; "(Cmdf abort)Syntax error."

; Use of this command file will be ended because a
; BASIC statement appears when the file is talking
; with Exec.
;
<2<Data-Gatherer.BS
BYE
TYPE <2<Data-Gatherer
IF X>2 THEN PRINT "End"
; The above BASIC statement causes an error message to
; be displayed: "(Cmdf abort)I can't find that file."
; (Exec was looking for a command or file name
; consisting of the letters IF)
```

You will find one feature of Exec especially useful when you construct command files. That is Exec's ability to recognize the special disk specifier symbols ? and #. (See Section 4.7, Special Disk Specifier Symbols.) You may use the same command file many, many times. But you may find it inconvenient to reedit the command file every time you use it. You may find yourself doing that, however, if the drive numbers mentioned in the file are not correct for this particular use of the file.

#### EXAMPLE

```
<3<Data-Creator.GO
; Data-Creator creates a data file, DATA1 on drive 3
COPY <3<DATA1 <3<Data-Copy
LIST 3
PACK 3
```

The above command file is fine as long as Data-Creator is indeed on the disk in drive 3. But what if you are not sure on which drive it will be convenient to run Data-Creator? Every time you

use the command file, you will have to re-edit it to make sure it selects the proper drive numbers.

By using the special disk specifier symbols, you can avoid this dilemma.

EXAMPLE:

```
<?<Data-Creator
; Data-Creator creates a data file on its own disk
COPY <#<DATA1 <#<Data-Copy
LIST #
PACK #
```

If Data-Creator is on a disk in a valid drive, Exec will find it. From that point on, the # symbols in the command file will select the drive on which Exec found Data-Creator.

## Section 10

## APPLICATIONS PROGRAMMING

Your System 88 software contains several features of special interest to the applications programmer. This section briefly discusses what an applications program is and points out a few of the system features especially useful for applications programs.

An applications program is one that has been designed to provide a specific service. A program that helps you to figure stress factors or perform a statistical test is an example of an applications program.

A good applications program is "human engineered": fully documented and tailored for use by people not familiar with computers. In fact, the user of a good applications program should not need to know any more about computers than how to insert a disk into a drive and push a Load button. The designer of an applications program generally assumes that the future user of the program will have minimal knowledge of the System 88 and no knowledge of how the program works. This means that an applications program NEVER asks for input by displaying:

?

All requests for data are fully explained by the program (e.g. "How many employees work for you?"). The program should validate all data input by the user and respond appropriately.

## EXAMPLE:

How many employees work for you? -20.5

That's not possible. Try again.  
How many employees work for you?

System 88 provides several tools useful for the development of applications programs and packages. (A software "package" is a set of programs that provide the same type of services, such as a set of programs that perform accounting and bookkeeping functions.) The features you will probably find most useful are: 1) the INITIAL program; 2) command files; 3) designation of a file as a system file; 3) automatic invocation of BASIC; 4) BASIC program auto-execute mode; 5) use of the BASIC file-handling command DEF to connect your own hardware devices to the system software. For more information on each of these points, see the appropriate sections of this manual, the System Programmer's Guide, and BASIC: A Manual.

By using these features in combination, you can limit the amount of computer experience the user of your programs needs. By naming a program INITIAL, you make it the first thing that the user of your system sees when the machine is turned on or reset. INITIAL might be a BASIC program telling the user of the system what kinds of programs the system has in its disk library, or it might be a program asking for log-on information (making sure that the user of the system is a legal user). INITIAL might even be a command file, causing a series of programs to be executed.

A big advantage in the use of command files is the fact that they can contain system commands. This allows you, the programmer, to arrange for certain system functions to be performed (such as deleting data files, invoking BASIC programs, etc.) without the user of your system ever having to enter a system or BASIC command. Your program user, then, can be completely inexperienced in the use of programs or computers. Because BASIC is automatically invoked by the system when you ask Exec to invoke a BASIC program, the user does not even have to be aware he is communicating with a different entity (BASIC vs. Exec) when the BASIC program begins execution. Even when a BASIC program is brought into memory by way of a BASIC LOAD command when you are already in BASIC, automatic execution of that program can be arranged by saving the program in "auto-execute mode"; execution will begin without the use of the BASIC RUN command.

Experienced machine language programmers have several powerful options available to them for applications programming. The System Programmer's Guide provides methods for designating any file as a system file. An applications program so marked cannot be altered, deleted, or renamed. Also, BASIC programs marked as system files cannot be listed or modified from BASIC. This is an obvious advantage in applications programming, where experimenting by an inexperienced user could destroy your program. In addition, the Guide discusses a way to disable the Control-Y command in BASIC. This prevents the user of the program from interrupting its execution. The BASIC DEF statement may be used to define your own machine language file-management routines.

## PART III

## THE EDITOR AND THE ASSEMBLER

## INTRODUCTION TO PART III

The System 88, as delivered, includes several special software items prerecorded on the System Disk in addition to the disk operating system. Two of them, the Editor and the Assembler, let you create text and BASIC and assembly language programs.

The ability to create and change text is valuable to all users of the System 88. The Assembler is useful to those who want to write assembly-language programs. The Editor is discussed in Section 11. You may find it the most useful part of the system. Section 12 briefly reviews the use of The Assembler. It is more fully discussed in the System Library volume The MACRO-88 Assembler.





## SECTION 11

## THE EDITOR

An "Editor" is a system program that has a special use. To put it briefly, the Editor allows you to create and change text. Specifically, the Editor displays on the screen the contents of a text file (a file that YOU can read, as opposed, say, to a machine language file) and lets you move back and forth in the text, adding and deleting as you please. When you write text (including text-form programs, such as BASIC programs or assembly language programs) you use the Editor, working directly on the screen. You can jump from place to place in the text at will, taking things out, inserting things.

## 11.1 INVOKING THE EDITOR

The Editor is a machine language program on the System Disk. To invoke it (that is, to bring it into memory and execute it), type (after a system prompt \$ or \$\$):

```
EDIT in-pathname out-pathname
```

followed by a carriage return. "in-pathname" is the file specification of the text file you want to edit, and "out-pathname" is the file specification you want to give to your new, edited file.

## EXAMPLE:

```
EDIT <2<MYSTERY-STORY <3<WHODUNIT
```

If you do not specify an extension for the input file, the Editor will bring in the first file of the given name found on the specified disk, regardless of its extension. If you do not specify an extension for the output file, the Editor will use the default extension .TX (text file).

If you do not include a disk specifier for either the input or output file, the Editor will use the system drive.

You may use the disk specifier symbols ? and #, as long as you do not use the ? symbol in the file specification of the file to be created. (See Section 4.7, Special Disk Specifier Symbols).

## EXAMPLE:

Legal:           EDIT <?<Orbit <#<Orbit-2  
Illegal:          EDIT <2<Math <?<Tutorial

If you want your new, edited file to have the same file specification as your original text file, do not specify an output file in the EDIT command.

## EXAMPLE:

EDIT <3<END-OF-QUARTER-REPORT.TX

The Editor will delete the original input file on the disk in drive 3, and create a new output file on the disk in drive 3 named END-OF-QUARTER-REPORT.TX.

You use the Editor to create a new text file, as well as to edit an existing file. To create a new text file, choose a file specification, using a file of name that does not already exist on that disk. Type:

EDIT in-pathname

The Editor will look for the file specified in the command, and not finding it, will create a new text file of that name.

To create a BASIC program file, specify the extension .BS.

After you have invoked the Editor, you will see a message in the upper left hand corner of the screen: "Edit/##," which tells you the version number of your Editor. Next you will see one of three messages, depending on the form of your Editor invocation:

If you type:

EDIT in-pathname out-pathname

where the input file is an existing file, the Editor will display the message:

Input file: opened  
Output file: opened

Hit any key to continue.....

If you type:

EDIT in-pathname

where the input file does not exist, the Editor will display the message:

Input file: not found

Creating output file: opened

Hit any key to continue.....

If you type:

EDIT in-pathname

where the input file is an existing file that you want to edit, the Editor will display the message:

Input file: opened  
(Old input file deleted)

Output file: opened

Hit any key to continue.....

(The editor will load text, if it exists, BEFORE stopping and awaiting a strike of any key.)

Note that when you edit an existing file, the old file (input file) will not be deleted if the new file (output file) is given a different name or a different disk specifier.

After one of the above screen messages is displayed, the Editor will wait. You may tell it to continue by typing any key on the keyboard. Then one of the two things will happen: 1) If the input file is an existing file, the cursor (the white rectangle at the top left of the screen) will begin to blink. The Editor is now copying text into memory from the disk input file; don't interrupt it. Or, 2) If the input file does not exist (that is, you want to create a text file), the screen will clear and the Editor will display an unblinking cursor on the empty screen, as an indication that you may begin to type in your text. You are now ready to start creating the file.

If you are editing an existing input file, the cursor will continue to blink until enough text has been copied into memory from your input file to fill half of the available memory space. At that time, the cursor will stop blinking, and the first fifteen lines of your text file will appear on the screen. You can now start to edit your file.

## 11.2 USING THE EDITOR

You are now ready to begin editing or entering text. The cursor is a kind of place marker; it tells you the precise point in the text that will be affected by your commands. You move the cursor to the point you want to affect, then type in what you want to add or delete what you want to remove. The cursor does not "cover up" a character; it fits into a space of its own.

### 11.2.1 Moving the Cursor

When you are typing in text, the cursor moves in the usual way, one space at a time from left to right. When it reaches the end of the line on the screen, it jumps down to the next line. To change existing text, it is important to be able to move the cursor WITHOUT inserting text. You can move the cursor by using the four keys on your keyboard marked with arrows. These keys are on the lower left of the keyboard.

To move the cursor forward (to the right), use the right-pointing arrow key. When the cursor reaches the end of the line, it will jump down to the beginning of the next line below its present position.

To move the cursor backward (to the left), use the left-pointing arrow key. When the cursor reaches the end of the line, it will jump up to the end of the line just above its present position.

To move the cursor up, use the up-pointing arrow.

To move the cursor down, use the down-pointing arrow.

When you move the cursor up and down by use of the up and down arrow keys, you will notice that the cursor attempts to move to the position exactly above or below its present position on the line. This direct up or down movement may be impossible because the cursor is to the right of the end of the line it's moving to. In that case, it will move to the end of the line. That is its new position, and if you move the cursor again up or down, it will attempt to move to the position directly above or below its new position.

To move to the BEGINNING of the line above or below, type Escape, then up arrow or down arrow.

As you continue to move the cursor up or down, eventually you reach the top or bottom line. If you continue to move the cursor up or down, the lines displayed on the screen change. The fifteen-line display on the screen is a "window" into the text now in memory. As you move the cursor up, you advance toward the beginning of your text; as you move the cursor down, you proceed toward the end of the text in memory.

### 11.2.2 Inserting and Deleting Text

You can insert text at any point in the text display. Simply begin typing, and your insertion will appear on the screen at the cursor position. This is also the way you create a new text file; just start typing.

CAUTION: What appears to be one line of text on the screen may not necessarily be a true line. When you

type more than 64 characters, the cursor will "wrap around"; that is, the rest of your line will continue on the next line of the screen. However, these two lines on the screen, apparently separate, are in reality ONE line, since no carriage return was typed to separate them. If you print your output file on a printer, the two lines will be printed as one, with no break between them.

Therefore, unless you know exactly how many characters you want per line in your output file, it is wise to remember to type a carriage return at the end of every screen line so that one line of text in your file will be equal to one screen line.

When you insert text, remember to check to see if you need to change the locations of the carriage returns in the text following your insertion.

To delete text, move the cursor to the RIGHT of the character or characters you want to erase. To delete one character, hit the DELETE key; one character to the left of the cursor will disappear and the cursor will move to the left one space each time you hit the DELETE key. To delete one word, type a Control-W (hold down the CTRL key and type a W). The word to the left of the cursor will disappear, and the cursor will move to the left. To delete one line of text (the characters between two carriage returns), type a Control-X (hold down the CTRL key and type an X). Remember that the Control-X will delete ALL of the characters back to the next previous carriage return, disregarding apparent breaks between lines caused by wrap-around. Be careful, then, that what appears to be a line is in fact one separate line of text.

To change characters, words, or lines: delete the appropriate text, then simply type in your changes before moving the cursor. You can undelete characters you have just deleted with the Control-U command: hold down CTRL and hit U. Deleted characters will be restored one at a time. This works only if the cursor has not been moved since the characters were deleted.

### 11.2.3 Moving Within the Text in Memory

You now know how to move the cursor around in the text display. There are ways to make larger movements within the copy of text that is in memory. This movement is caused by control characters: Control-B, Control-E, Control-N, and Control-P. To type a control character, hold down the CTRL key and type the appropriate letter (e.g., Control-E: hold down the CTRL key and type an E).

You can return to the beginning of the text in memory by using the Control-B (Beginning) command. If you have not outputted any text (see Section 11.2.4, Inputting and Outputting Text),

the beginning of your text will be the first fifteen lines of your input file. In any case, Control-B takes you to the beginning of the text currently in memory.

To get to the end of the text now in memory, use a Control-E (End) command. If you have a long file--too long to be entirely contained in available memory--the end of the text in memory may not be the end of your input file (see Section 11.2.4, Inputting and Outputting Text).

You can "turn the pages" of the displayed text with two other control commands. To see the next fifteen lines of the text currently in memory, use the Control-N (Next) command. To see the previous fifteen lines, use the Control-P (Previous) command.

#### 11.2.4 Inputting and Outputting Text

When the Editor is first invoked, it copies as much text from the input file as will fill half of available memory. It leaves half of memory free so that you can add text to the text in memory.

If you have a long text file, all of it may not fit into half of memory. In that case, if you want to make changes in the text not yet copied in from the input file, you will want to force the Editor to copy in more text from the input file. Use the Control-A (Append text) command. Enough text will be copied in from the input file to fill half of the remaining space in memory. The next Control-A command will fill half of THAT remaining space, and so on, until either all of the text is in memory, or you have run out of room.

You will know when you have run out of space in memory because the Editor will not copy in more text and because when you try to insert text, the cursor will not move and no characters will appear on the screen. In either case, you must make room by outputting text into your output file. Use the Control-O (Out-put text) command. Half of the text now in memory will be output to your output file.

**BEWARE:** Once text has been output to your output file, it is no longer available for editing; it is gone from memory. You can only reach that text by leaving the Editor, then returning and re-editing the file, using the output file as your new input file.

Once you have outputted text, you can again copy in text from the input file using the Control-A command until memory is again full.

**Important note:** Copy in only as much text as you actually need to edit. When you exit from the Editor, the Editor will automatically output any remainder of your input file into your

out-put file. You do not have to use Control-A and Control-O to input and output the text past the part that you want to change.

#### 11.2.5 Searching for a Text String

A special Editor command lets you find any word or other combination of characters (a "string"), wherever it occurs in the text currently in memory. Suppose you suddenly realize that you have been mis-spelling a word, or you want to replace a word with another, everywhere that it occurs. Or you used a certain word or phrase in a particular place, and you want to find that place fast. You can do this by using the Control-F (Find) command. To use Find, type Control-F. A second cursor will now appear next to the first one. Now type the string that you want to find. It appears between the two cursors. You can include control characters (such as a TAB or a carriage return); they will appear on the screen as Greek letters. (By the way, you can delete mistakes in a "search string" just as when you create any text.) When the string is complete, hit the escape key, ESC, once. You can use the Control-F command at any place within the text, but make sure that your position is before the string you want to find; the Editor begins its search at your current cursor position.

After you hit the escape key, the second cursor will disappear. If the original cursor does not blink, the Editor has not found any occurrences of the search string. If the cursor does blink, the Editor has found an occurrence of the search string; the cursor will move to the space directly to the right of the first occurrence of the search string within the text. You can now change the found string.

To find the next occurrence of the search string, use the Control-C (Continue search) command. The Editor remembers the search string and advances in its search to the next occurrence. You can use Control-C as many times as you want: if there are no more occurrences of the search string, the cursor simply will not move from its current position.

To be sure to find all occurrences of the search string, use the Control-B command to get to the beginning of the text in memory. Make sure that your search string is exactly the same as the item you are searching for, mis-spellings and all. If the item you are looking for contains a space, TAB, carriage return, etc., you must include that within the search string.

#### 11.2.6 "Cut and Paste": Moving, Copying, and Deleting a Block

The System 88 Editor has a very useful set of special commands that let you "cut and paste." Using these commands, you can move a block of text from one place to another, copy it in any number of places without re-typing it, or delete the block.

These commands all use the left and right arrows to mark the beginning and end of a block of text.

#### 11.2.6.1 Moving a Block of Text

To move a block of text from one location to another, first mark the beginning and end of the block with arrows. Just before the first character in the block, insert a right-pointing arrow by hitting the ESC key, then the right arrow key. Then go to the end of the block of text and insert a left-pointing arrow by hitting the ESC key, then the left arrow key. The block of text is now marked; moving, copying, or deleting a block of text always begins with marking the block with arrows (and always uses the FIRST marked block if there are several).

Now that the block of text to be moved is marked, move the cursor to the place in the text where you want to insert the block. When the cursor is correctly located, hit the Escape key (ESC), then CTRL-C (hold down the Control key and type C--for Copy). The block of text will appear in the current cursor location.

The last step is to delete the block from its original location. This you can do without even returning to the original location of the block. Just hit the ESC key, then hold down the SHIFT key and hit the DELETE key. The block of text will vanish from its original location. You have now moved the block of text from its original location to a new one.

#### 11.2.6.2 Copying a Block of Text

As you can see, the same commands let you copy a block of text several times over without re-typing it. Mark the block of text as above, then move the cursor to where you want the block to appear again. Type ESC, then Control-C, and the block will appear at the current cursor location. You can move the cursor again and copy the block in other locations if you want to.

All that remains to be done after you've copied the block as often as you want is to remove the arrows from the original block. Once again, this you can do without even returning to the location of the arrows in the text. Just hit the ESC key, then the DELETE key, and the arrows will vanish from the marked block, leaving that block intact.

#### 11.2.6.3 Deleting a Block of Text

To delete a block of text, first mark it with arrows as above. Then hit the ESC key, then hold down the SHIFT key and hit the DELETE key. The block will vanish, arrows and all. Take care not to mark the wrong part of the text, because once it's gone, it's gone.



### 11.2.7 Printing a Marked Block

It's a great convenience at times to be able to print on a printer only a particular part of a file. You can do this by marking the desired block with arrows as above, then typing ESC-Control-P (hit the ESC key, then hold down the CTRL key and type P). Only the marked block will be printed. (If your "printer" is the screen, you will get a distressing and not very helpful display.)

### 11.2.8 Reversing Capitals and Lower Case

One Editor command reverses capitals and lower case--it changes all the capital letters to lower case letters and vice versa. The command affects only the characters to the RIGHT of the cursor, and only the characters in a single line. Symbols other than letters are not affected.

To reverse capitals and lower case letters, type Control-V (hold down the CTRL key and hit V). A line like this:

```
ENTER THE STOCK PURCHASE PRICE
```

becomes this:

```
enter the stock purchase price
```

Now you just have to change the initial e to a capital to have a normal line. In fact, if you begin with the cursor located just to the right of the E, the CTRL-V command will reverse all the letters in the line EXCEPT the initial E.

## 11.3 EXITING FROM THE EDITOR

You must exit properly from the Editor for your editing efforts to be actually recorded from memory onto the disk.

When you are done editing, type an Escape. Then type Control-E. As you leave the Editor, you will see the message:

```
Exiting...
```

The cursor blinks, indicating that the text currently in memory and the rest of your input file are being outputted to your out-put file. Make sure that the disk containing your output file is not write-protected; if it is, you will receive an error message and a system prompt \$, indicating that you are no longer communicating with the Editor, but with Exec.

If your output file will not fit on the disk specified in your output file file specification, the Editor will tell you so:

```
Output disk is full.  
New file name:
```

You must then give the Editor a file specification for a file to hold the rest of the output text. The disk specifier for this new output file must refer to a disk that has room enough for the left-over text. The Editor will now display the message:

```
Output file: opened
```

You now have two output files, each containing part of your edited text. To avoid this complication, make sure that the output file specified in your invocation of the Editor is on a disk that has enough room for your file. (To learn how to re-combine your two files, see 11.4.1, Combining Files.)

If you should accidentally type a Control-Y, returning you from the Editor to Exec, you can resume editing at your point of departure by typing CONTINUE and a carriage return. If for some other reason you find that your editing has been interrupted and you are back at the system level (indicated by a \$ or \$\$ prompt), you will probably be able to resume editing your file without any difficulty. First you must be in enabled mode. If you aren't (if you see the single Exec prompt \$ instead of the double prompt \$\$), BEFORE you do anything else, type:

```
ENABLE
```

followed by a carriage return. When you are enabled, type:

```
REENTER
```

followed by a carriage return.

You should be back editing again.

## 11.4 COMBINING AND DIVIDING FILES

### 11.4.1 Combining Files

The Editor allows you to put two or more files together to form a single file. First write down the names of the files to be combined, in the order you want them combined. Make sure the names are exactly correct.

Begin by invoking the Editor and the first file. Append using Control-A until you have reached the end of this file (you can output using Control-O to make room in memory as you go).

When you have reached the end of the file, type ESC-Control-I (hit ESC, then hold down CTRL and type i). This screen message will appear:

```
New input file name:
```

Type in the name of the file you want to be next, then hit return. The screen will say:

New Input file: opened

Hit any key to continue....

Hit any key. The second file has now been brought in. To append it, type Control-A. To add the next file, repeat the procedure: keep appending until you reach the end of the second file, then type ESC-Control-I and respond to the screen messages. Keep repeating the procedure till you have combined all your files.

#### 11.4.2 Dividing Files

To divide one file into several files, invoke the Editor and the file you want to divide. Use a name for your new output file that you want to assign to one of the new files you are creating.

EXAMPLE:

```
EDIT <2>USA-DATA <2>OHIO-DATA
```

Now, using the arrows as above, mark the part of the input file that you want to make into a new file. Then type Control-D (hold down CTRL and type d--for Dump). The marked block will be copied out to the output file. If the new file is to consist of several parts of the old file, repeat this procedure on the other desired blocks.

You now have a new file consisting of part of what was in the old file. To make another new file out of another part of the old file, type ESC-Control-O. This screen message will appear:

New output file name:

Type in a new output file name, then a carriage return. You will see this message:

Output file: closed  
New Output file: opened

Hit any key to continue....

Hit any key. The next time you mark a block and type ESC-Control-D, the marked block will go out to the newly named output file.

NOTE: If you are dividing up a long file, you may find that the part you want to be your new file is not in memory when you begin--there isn't enough room. If this is the case, remember that this time you CAN'T use Control-O to make more room in memory.

If you do, unwanted material from the old file will go into the new file. Instead, delete the contents of memory as required, then append using Control-A. Repeat till you have the part you want in memory.

### 11.5 REFERENCE LIST OF THE EDITOR COMMANDS

Typing EDIT plus the name of the text file brings in all Editor features and the desired input file. You may also name the output file at this time.

Arrow keys: The arrow keys are used to move the cursor. If your keyboard does not have arrow keys, the indicated control characters will serve the same function.

UP Up-pointing arrow (Control-Q) Move cursor to point directly above present cursor position; if there are no characters directly above the cursor, it moves to the right of the nearest character above.

DOWN Down-pointing arrow (Control-R) Move cursor to point directly below present cursor position; if there are no characters directly below the cursor, it moves to the right of the nearest character below.

RIGHT Right-pointing arrow (Control-S) Move cursor one space to the right; if the cursor is at the right end of the line, it moves to the beginning of the next line down.

LEFT Left-pointing arrow (Control-T) Move cursor one space to the left; if the cursor is at the left end of the line, it moves to the right end of the next line up.

Move the cursor to the BEGINNING of the line above by typing Escape-up arrow. Move it to the BEGINNING of the line below by typing Escape-down arrow.

#### The Commands

##### Control-A

Add to text in memory from input file.

##### Control-B

Display beginning of text in memory.

##### Control-C

Continue search for search string.

##### Control-D

Same as DELETE key.

Control-E

Display end of text in memory.

Control-F

Find a string. Type Control-F, then the string. Then hit ESC (escape key) once.

Control-H

Same as BACK SPACE key: insert a backspace into the text (which appears on the screen as the Greek lower-case iota).

Control-I

Same as TAB key.

Control-L

Insert a form feed character into the text (which appears on the screen as a lambda).

Control-N

Display next fifteen lines from present position.

Control-O

Output text in memory to output file.

Control-P

Display fifteen lines before present position.

Control-U

Undelete deleted characters.

Control-V

Reverse upper and lower case.

Control-W

Delete one word.

Control-X

Delete one line.

ESC-right arrow, ESC-left arrow

To mark a block of text for moving, copying, or deletion, insert a right-pointing arrow at its beginning by hitting the ESC key, then the right arrow key, and insert a left-pointing arrow at its end by hitting ESC and the left arrow.

NOTE: The commands below always affect the FIRST marked block in memory.

ESC-Control-C

Copy a marked block to the present cursor location.

ESC-Control-D

Output a marked block to the output file.

ESC-SHIFT-DELETE

Delete a marked block.

ESC-DELETE

Delete block markers only.

## Section 12

## THE MACRO-88 ASSEMBLER

The System 88, like any computer, actually "understands" only binary data. It obeys instructions that are expressed as binary numbers or code, called machine language. You can write a program in machine language, but the binary-code instructions, being numbers, are rather hard to remember, so the programmer who wants to write in machine language actually uses "assembly language," a translation of machine language into word-like mnemonics. Jump instructions, for instance, all use the mnemonic JMP in place of the corresponding machine language.

Having written a program in assembly language, the programmer submits it to a special program called an assembler. An assembler simply re-translates the assembly language into machine language so that the computer can understand it. The assembly-language program is called the source code, and the resultant machine-language program is called the object code.

Note that an assembler is a translator program, which creates a new version of the program in machine language which can then be run in a computer. BASIC, on the other hand, is an interpreter program, which is automatically called into play whenever you run a BASIC program. When you run a program written in BASIC, the file called BASIC runs simultaneously and intervenes between the BASIC program and the processor, interpreting the BASIC statements.

It might seem that all assemblers would be more or less the same, or at least that all assemblers written to work with the 8080 processor (the processor used in the System 88 and many other small computers) would be much the same, but they are not. The assembler translates your assembly language program into machine language, and the ability of your translator to understand what you are saying limits what you can say. All assemblers written for the 8080 processor allow you to use the assembly language instructions corresponding to the 78 different kinds of machine instructions that are built into the 8080. But all assemblers also have other commands that cause other events than generation of a machine instruction. Some assemblers are very simple, and put the programmer to some trouble to make the assembly-language program correspondingly straightforward. Others, like MACRO-88, can understand and translate some rather sophisticated statements. For instance, MACRO-88 lets you use macro-instructions (which are a bit like sub-routines condensed into a single instruction) in your assembly language program.

This section gives the minimum information necessary to use the MACRO-88 assembler, with no explanation of how to program in

MACRO-88 or of what the assembler does internally or why. To learn more about the assembly process, the philosophy of MACRO-88, its implementation restrictions, its library file system, or its advanced macro features, read the MACRO-88 Assembler Manual.

## 12.1 INVOKING MACRO-88

We'll assume you are "in Edit" and have just finished writing a draft of an assembly language program. You now want to go from the Editor to the Assembler, assemble your program, and check it for errors. Exit from Edit to Exec in the usual way. At this point you can choose to have your program listed on a printer other than your default printer (see Section 13, The Printer Driver). Select another listing device by typing the Printer command and the name of the device you want to list on:

```
$Printer (user's printer driver)
```

(Note: throughout this discussion, variable items are enclosed in "square brackets" [ ] .)

If you do not want to list on a printer other than your default printer, skip the step above.

Now type the assembler invocation command, `Asmb`, and the full name of your source code file (starting with the disk specifier if your source file is not on the default drive), plus another name for the object code output file if desired:

```
$Asmb [source file] [optional object code output file]
```

If the optional object code output file name is omitted, no object code will be generated. You may choose this option when test-assembling a program that you expect will have many assembly errors.

## 12.2 SELECTING OPTIONS

When you invoke the MACRO-88 Assembler, it responds by putting a series of questions on the screen, as follows:

```
MACRO-88 Version (version of this copy): (date of this version)
Hardcopy? (default is video display) (Y or N):
```

Type Y for yes if you want a hard-copy listing (listing on any device other than the screen). Type N if you wish to list on the video screen only---your probable choice if this is the first attempt at assembly for this program. Then hit the carriage return (all responses end with a carriage return).

```
Full listing? (else errors only) (Y or N):
```

The first time you try to assemble a program, you will probably choose to answer this question with an N for no. An N response



means that the assembler will list only errors that it finds in your program, if any. You can then return to Edit and correct errors, then re-assemble.

Symbol table printout? (Y or N):

The assembler will list a table of all the symbols used in the program if you answer yes.

### 12.3 THE ASSEMBLY PROCESS

Assembly begins automatically as soon as you answer the last question above.

The assembler makes two passes through your program, first creating a table of labels, then doing the actual assembly. As the process unfolds, the assembler displays on the screen:

Pass one. Pass two.

Having gone through your source file twice, when MACRO-88 encounters an END statement, it displays:

Error total = [total number of assembly errors]

It then returns you to Exec.

If the assembler comes to the end of your source file on its first pass without finding an END statement, it assumes that the program resides in more than one file and asks you to name the file that contains the rest of the program. This continues until an END statement is actually found. Then in pass two, the assembler asks for the names of all the files again, even the source file specified in the "Asmb [source file]" line. Giving only a carriage return as a response to any of the questions says that there is no continuation file: "I just forgot to put in the END statement." Assembly then continues as if there actually were an END statement.



## Section 13

## SYSTEM 88 PRINTER DRIVER

The System 88 Operating software includes a Printer Driver. It is the job of this Printer Driver to act as the connection and frequently as the interpreter, between the system and the printer. The System 88 Printer Driver is designed so that it can send characters to many different makes of printers, enabling your System 88 to run them. This flexibility is possible because the Printer Driver provides a means for you to teach it about the particular characteristics of your printer. This flexibility can even include parallel printers since the Printer Driver is capable of utilizing a user-written program which will handle the communication between the system and the parallel printer.

All printing-related functions are controlled by the Printer Driver, including the establishing of page parameters (top and side margins, lines per page and, width) and of a DEFAULT printer which the Printer Driver will automatically prepare to run each time the System 88 is turned on or the Load button is pressed. Each time the Printer Driver learns about a new printer it adds that printer name to its list of "known printers" and stores the printer's characteristics until it needs to run that printer.

Whether or not there is a printer connected to the System 88, the Printer Driver is automatically loaded into memory when the system disk is Loaded and is always ready to establish communications with the printer. The Screen printer can be connected if you do not have a printer available but would like to test a program which involves printing. In this case, the Printer Driver will output characters to your video screen rather than to a printer.

. . . . .

Now that you have an idea of what the Printer Driver can do, the next step is to learn how to define a printer and establish a DEFAULT printer. Then we will explain the daily use of your Printer Driver to print from Exec, or BASIC and to connect different printers or establish different page parameters.

### 13.1 USING THE PRINTER EDITOR

How does the Printer Driver learn how to run a particular printer? You teach it by supplying the information it needs about each printer type. You supply this information while in the Printer Editor which you invoke by typing:

```
$Setup
```

Below is a list of things you can do while the printer editor is invoked:

Supply answers to an appropriate series of questions, allowing the Printer Driver to learn about a new printer.

Display the list of characteristics of any known (previously defined) printer.

Establish any known printer as the DEFAULT printer.

DELETE all information about any previously defined printer. In which case, that printer would then be unknown to the system's Printer Driver.

Return the system to EXEC.

When you type Setup you will see the following:

```
Known printers: Null Screen Diablo Diablo-1200 1660
Commands: NEW CUSTOM VIEW DEFAULT DELETE EXIT
##
```

## is the prompt for the printer editor, which is waiting for instructions from you.

The "Known printers" are those which the printer already knows about. The ones shown above are the ones that are pre-defined and included on the system disk shipped by PolyMorphic Systems. As shipped, the DEFAULT printer is the known printer "Null." This means every time you Load your system or turn it on, the printer will prepare to drive the Null printer. While the Null printer is connected, no data will be output to a printer.

If you are setting up a System 88, or if you have just attached a different sort of printer, you will need to establish your printer as the DEFAULT printer. You may find that the printer you wish to connect is not a known printer; you will then need to define your printer to the Printer Driver before you can establish it as your DEFAULT printer. (See 13.1.1)

If the printer you want to use is a known printer, then you should type

```
##VIEW Printername
```

Now you can see what page parameters were established for your printer. If the page parameters are different than those you would set, you can redefine the printer using another name.

For example, you can redefine the Diablo-1200, using that same name only after you DELETE the existing definition by typing

```
##DELETE Diablo-1200
```

NOTE: If you type DELETE Diablo-1200 you will delete the definition of that printer and remove it from the list, freeing the name Diablo-1200 for your own definition. Before you do this, you should jot down the technical characteristics so that you can reenter them in a minute, along with your page parameters.

Once you have determined that one of the known printers is suitable, that its technical characteristics match those of your printer and its page parameters are what you want, you can make it your DEFAULT printer. Your DEFAULT printer is the one which the Printer Driver will prepare to run every time the system software is loaded. You specify your DEFAULT printer while still in the printer editor by typing

```
##DEFAULT Printername
```

After EXITing, the system will load the new DEFAULT printer each time the load button is pressed or the system is turned on. The new DEFAULT printer is, however, not connected after the EXIT command. Either the system must be powered up or loaded, or the printer must be connected with an explicit Printer Printername command.

The use of the NEW command to define a printer is explained in 13.1.1.

#### 13.1.1 Defining NEW Printers

If you would like to teach the Printer Driver about a particular type of printer, first type

```
##NEW Printername
```

You can select any name you want but a single printer name cannot be used twice. There may be no spaces within the printer name (e.g. Diablo 1200 is Diablo-1200).

The information you need to understand the printer definition procedure will be presented in two parts. First we will list the various questions which you may be asked by the Printer Editor. Then we will define the terms used in these questions.

First the Printer Setup questions:

Printer setup:

- Similar to a Diablo?
- Understand Form Feeds?
- Understand TAB characters?
- Speed of printer (in BAUD)?
- Blocking type device?

Your answer to this last question will be followed by different questions if it is affirmative than if it is negative.

If your answer to this last question is Yes, you will see the following questions:

- Device buffer size (0-255)?
- Send a start character?

Again, an affirmative answer to this question will be followed by a different set of questions than will a negative answer.

If you indicate that your printer does send a start character, you will be asked the following:

- ASCII code for START character?
- ASCII code for END character?
- ASCII code for ACKNOWLEDGE character?

If you indicate that your printer does not send a start character, you will be asked the following:

- ASCII code for END character?
- ASCII code for Acknowledge character?

If your answer to the Blocking type device question was No, then the following questions will follow:

- ASCII code for PAD character
- Number of pads after CR?
- Number of pads after LF?
- Number of pads after TAB?

You will only be asked the following at this point if you indicated that your printer does not understand form feeds.

- Number of pads after BS?

Now you will be presented the DEFAULT page parameter questions:

Lines per page (form size)?  
Characters per line (page width)?  
Lines for TOP margin?  
Lines for BOTTOM margin?  
Offset for left EDGE?

Listed below are definitions of Printer Setup terms; consult these while you are defining your printer.

Similar to a Diablo?

(Answer Yes or No)

Your printer is similar to a Diablo if it has the following:

settable left margins (code <esc> 9).  
Settable tabs (code <esc> 1)  
Absolute horizontal positioning (code <esc> HT pos)  
Clear tab stops (code <esc> 2)

Understand Form Feeds? (Answer Yes or No)

Some printers automatically respond to form feed characters by moving on to the top of the next page. If yours does not, the Printer Driver will translate form feed characters into enough blank lines to move the printer to the next page.

Understand TAB characters? (Answer Yes or No)

Some printers automatically respond to tab characters by tabbing over to pre-set tab stops. If yours doesn't, the Printer Driver will simulate tab characters by supplying the correct number of spaces to the printer.

Speed of printer (in Baud)? (Answer with a decimal number)

Printers communicate at a set rate of speed called the Baud rate. Consult the specification for your printer to find this figure.

Blocking type device? (Answer Yes or No)

Some printers have an internal buffer and thus can receive a block of characters at a time, initiated by an optional "start character," delimited by an "end character" and once received the printer returns an "acknowledge character."

Device buffer size?

(Answer with a decimal number) Respond by typing the internal size of the printer's buffer.

Send a Start character? (Answer Yes or No)

Some printers, Diablos for instance, don't need start characters.

ASCII code for START character? (Answer with a decimal number)

The start character varies from printer to printer. For most printers it is <stx> ( ASCII code 2 ).

ASCII code for END character? (Answer with a decimal number)

The end character varies from printer to printer. For most printers it is <etx> ( ASCII code 3 ).

ASCII code for ACKNOWLEDGE character? (Answer with a decimal number)

The acknowledge character varies from printer to printer. For most printers it is <ack> ( ASCII code 6 ).

Lines per page (form size)? (Answer with a decimal number)

The number of lines that your printer can print on one page varies. Most printers print six lines per inch, and most pages are eleven inches long, so 66 is the usual response to this question. If your printer or form differs, do some calculation to determine how many lines your printer will print on your form.

Characters per line (Answer with a decimal number)

Most printers print 80 characters per line if they are using standard size paper. However, you may be using narrower or wider paper and consequently may wish to establish a different width.

Lines for TOP margin? (Answer with a decimal number)

Define how much of a margin you want left blank at the top of each page by stating a number of lines.

Lines for BOTTOM margin? (Answer with a decimal number)

Define how much of a margin you want left blank at the



bottom of each page by stating a number of lines.

Offset for left Edge (Answer with a decimal number)

This establishes the point to the right of the left edge that will be considered the left edge by your printer. A left bias between 2 and 10 is usually suitable.

When you have answered all questions, you will see the following message:

```
Printer defined ##
```

You can then establish your DEFAULT printer by typing

```
##DEFAULT Printername
```

Type EXIT to return to Exec. Once you have returned to Exec you can either push Load or type the following:

```
$Printer Default Printername
```

to cause the DEFAULT printer to be connected.

### 13.1.2 CUSTOM

If you want to drive a parallel printer with the System 88 Printer Driver, you can do so after you have supplied a user written program. The CUSTOM function of the Printer Editor allows you to do this. Type

```
##CUSTOM Printername
```

Questions will then appear on the screen one by one. The first two questions you will have to answer are:

```
Driver name?
```

In answer to this question, type in the name of the user written program which is stored on the system disk and is capable of handling the communication between the Printer Driver and the parallel printer. This name must be three characters or less; the extension on the file must be .PS.

```
Use Standard dialog?
```

An answer of no to this question will result in a short dialog (series of questions) followed by the standard page parameters questions.

If you answer yes, the program will run through the entire usual dialog (the questions shown in 13.1.1).

See Appendix-H for an example of a user-written driver.

## 13.2 PRINTING FROM EXEC.

Printer, PRINT, Directory, LOG, NOLOG and PAGE are the commands which are connected with the printing process initiated in Exec. As you have just learned, when you first install your System 88, and again when you change printers, you will follow the procedures outlined for defining printers to the Printer Driver. These defined printers constitute the list of "known printers." You will probably also establish a DEFAULT printer at that time.

\$Printer Printer-name

while the system is in Exec.

If your physical printer is not turned on and hooked up to your System 88 when you perform the above procedure, nothing will happen when you give a PRINT, LOG or FORMAT command. You will not even be given an error message because the system is waiting for a signal from your physical printer. When this happens, turn on your printer, push Load and type the Printer Printername command again.

If you wish to use a printer other than the DEFAULT printer you can type the command Printer followed by the name of the printer you wish to use.

Remember that you must have already defined that printer to your Printer Driver. To make sure that you have you can type:

\$Printer

You will then see a list of "known printers." Only one of these can be the DEFAULT printer at a time, but any one of them can be connected in moments by following the above procedure.

If you type Printer followed by a printer name other than your DEFAULT printer, your system will connect that alternate printer and continue to be capable of driving it until you use the Printer command to attach another printer, or until you push the Load button or turn off the system.

If you want your printer to print out the contents of a particular file, type

\$PRINT <2>filename

If you want your printer to advance to the top of the next page, type

\$PAGE

If you want the printer to print the list of files in your directory, type

\$DIR <2

If you want your printer to begin typing everything which subsequently appears on the screen, type:

```
$Printer LOG
```

If you want your printer to stop typing what appears on the screen, type

```
$Printer NOLOG
```

LOG and NOLOG provide a means for you to keep a record of all of your keyboard entries. Since your entries eventually scroll off of the screen, you can not refer to some prior sequence of your input and the system's response unless you use the LOG command to cause the printer to keep a record.

### 13.3 PRINTING FROM BASIC

To use the Printer Driver from BASIC, you must use two types statements in BASIC.

First you must attach your printer while in BASIC. This is done with the following statement:

```
10 FILE:C, LIST
```

where C is the channel to which the printer is attached.

Next, the user must provide BASIC PRINT statements in the following format:

```
20 PRINT:C,'prin list'
```

EXAMPLE:

```
10 FILE:2,LIST
20 FOR I=1 TO 10
30 PRINT:2,TAB(I),"Test the Printer Driver"
40 NEXT
```

You can also use

```
LIST:2
XREF:2
DUMP:2
```

For more information, see the System 88 BASIC manual.

### 13.4 USING YOUR SCREEN FOR A PRINTER

You may want to test a BASIC program or FORMAT a text file on a system without a printer. You can do this by typing

```
$Printer Screen
```

This means you have connected Screen, one of the printers your software already knows about to your Printer Driver. Once you have connected Screen as your printer, all output to the printer will appear on the screen.

### 13.5 Printer SET

Once you have defined your Printer Driver and its parameters, and have established it as your DEFAULT printer, you will not have to enter the Printer Editor again unless you need to define a new printer, or establish some other printer as your DEFAULT printer.

You already know that you can temporarily over-ride the DEFAULT printer by typing

```
$Printer printername
```

while still in Exec. You also know that if you are not happy with the DEFAULT page parameters, you can change them by redefining your printer in the Printer Editor. However, sometimes you may want to temporarily over-ride the DEFAULT parameters for your printer. You can do this without entering the Printer Editor, and without answer the technical questions about your type of printer. All you have to do is type

```
$Printer SET
```

You will then be asked the following:

```
Lines per page (form size)?  
Characters per line(page width)?  
Lines for top margin?  
Lines for bottom margin?  
Offset for left EDGE?
```

When you have finished entering your temporary parameters, the system will return to Exec. and you can then operate your printer with the new parameters. These parameters will remain in effect until the system is turned off or until you push Load. Then your DEFAULT parameters will again be in effect.

### 13.6 Printer SHOW

What if you forget what your DEFAULT parameters are? Or, you may forget whether or not you have temporarily changed the page parameters. You can view the page parameters that are in effect by typing:

```
$Printer SHOW
```

## Appendix A

## SYSTEM 88 ERROR MESSAGES

Sometimes Exec cannot respond to a command or file invocation. This may be because the input was incorrect (e.g., INAGE instead of IMAGE), illegal (e.g., DELETE Exec.OV), or impossible to perform (e.g., <2<FILE, where <2<FILE does not exist). At these times, the system displays error messages that tell you the problem and give you some idea of what to do.

All the error messages you may receive from the system are listed below, along with their possible causes. For error messages generated by BASIC, see the System Library volume BASIC: A Manual.

## ERROR MESSAGES GENERATED BY THE SYSTEM

The error codes associated with the error messages are given for the benefit of the machine language programmer who may want to modify the section of the system software that generates error messages. See the System 88 Library volume System Programmer's Guide for information on interfacing your programs with the system software.

ERROR CODE	MESSAGE
0101	DIO says: Bad parameters!
0102	DIO says: Hard error! Preamble bad!
0103	DIO says: Checksum error!
0104	DIO says: Verify error!
0105	DIO says: Write protected!
0106	DIO says: No disk or door open!
0107	No Controller for that device
0108	Data transfer error
0110	System PROMS must be version 74 or later!
0111	I can't do that to the System Drive!
0112	I can't, too much data for object disk
0201	I can't run that file
0202	Nothing to run!

0204           What?

0205           I don't know what to do with that file

0206           I don't have enough memory to do that!

0207           I can only pack entire disks

0300           I can't find that file

0301           I can't access that device!

0302           Disk directory unreadable!

0303           Disk directory unreadable!

0306           I can't read the directory-no disk or  
              door open!

0307           No controller (Controller circuitry may be  
              malfunctioning so severely that CPU ROMs  
              can not tell if it exists.

0309           Bad track 0 switch on the selected drive  
              (Circuitry to detect this may be at fault)

030B           Seek Error (Head positioning mechanism on  
              the selected drive did not get to where it  
              was directed by the controller (Controller  
              circuitry to perform this may be at fault).

030C           Controller Self-check error

0380           IC33 is bad

0381           Self Test Failure in SDLC

0382           Memory Error

0383           Memory Error on 88/MS Controller

0384           Memory Error on 88/MS Controller

0385           Memory Error on 88/MS Controller

0386           Memory Error on 88/MS Controller

0387           Memory Error on 88/MS Controller

0388           Memory Error on 88/MS Controller

0389           Memory Error on 88/MS Controller

038A           Phase Lock Loop tested bad

03FF Disk directory destroyed!

0500 Gfid says: Bad disk identifier

0501 Gfid says: Name too long

0502 Gfid says: .Illegal extension

0503 Gfid says: Name null or weird!

0504 I can't: the directory is full

0505 I can't: the disk is full

0506 I can't RENAME across drives: use COPY

0507 No new extension given

0508 I can't do that to a system file

0509 "<?<" is not allowed here

050C I can't copy directories

0600 That file already exists

0601 That file does not exist

0701 Output file not specified

0702 Output file already exists

0703 Input file not specified

0704 I can't edit that file!

0705 Input file does not exist

0706 I can't have two files open OUT on the same

0901 You haven't defined a set of parameters for that printer.

0902 You have already defined a set of parameters

0903 Please give me a printer name

0904 I can't do that to that printer name!  
(Cmdf abort)

## EXPLANATION OF ERROR MESSAGES

## DIO ERROR MESSAGES

When you see an error message beginning "DIO says:...", that message is coming from a particular area of the system. DIO (DISK I/O) is the part of the operating system that performs disk read and write operations. It reports any errors resulting from problems in writing and reading information to and from a disk.

Error Code 0101 DIO says: Bad parameters!

This usually indicates an internal system error, caused by the system giving bad arguments to DIO.

Error Code 0102 DIO says: Hard error! Bad preamble!

DIO thinks that your disk is bad. It wasn't able to read information off of it, and thinks that the fault lies with the disk, rather than with the system. Try again, perhaps with the disk in another drive. If you keep getting this message, you had better check the status of your disk, perhaps by erasing the disk using the INIT command. This will perform a simple surface test of the disk, by writing a zero in every location in every sector of the disk. (Of course, this costs you all the information that was on that disk.) If INIT can't write a zero in a particular location, you will get the message "Verify error," and you will know that your disk is bad.

Error Code 0103 DIO says: Checksum error!

The data that has been read off of your disk does not look valid to DIO. Try the operation again-- chances are, however, that your data is no longer accessible.

Error Code 0104 DIO says: Verify error!

The system has tried to verify a disk write operation. The data written on to the disk does not match the original data still in memory. This may be due to a faulty write operation or a change in the data in memory. Try again. If you receive this error again, suspect that your disk is bad.

Error Code 0105 DIO says: Write protected!

You are trying to write data on a disk that is "write-protected" (the disk has a write-protect tab fixed over its "write-enable" notch). A write operation cannot be performed on such a disk. To write-protect a disk, place a write-protect tab over the disk's write-enable notch (see Figure 1, Cutaway Drawing of a Disk). To make a write-protected disk available once again for write operations, simply remove the write-protect tab from the disk's



write-enable notch.

Error Code 0106 DIO says: No disk or door open!

You have attempted to access a disk, but the drive you have selected is empty, or the drive door is open. No read or write operation will be performed. If you have specified a legal disk drive number, but your system does not contain that many drives, DIO will respond with this message.

EXAMPLE: \$L 3 (no disk drive with that number)

Error Code 0110 System PROMS must be version 74 or later!

You have tried to run current software on a machine containing an old version of the System 88 PROMS (Programmable Read-Only-Memories). Can't be done.

Error Code 0111 I can't do that to the System Disk!

Exec assumes that the disk in the System Drive (usually drive 1) is your System Disk. It thinks you are trying to do something that will destroy the System Disk, such as imaging over it or initializing it. If you really want to do one of those things to the disk, put it in another drive and use another System Disk in the System Drive.

#### GFID ERROR MESSAGES

The area of the system that deals with getting and identifying disk files is GFID (Get-File-Identifier). If the system has trouble getting file names or identifying a file, GFID will generate one of the error messages below.

Error Code 0500 Gfid says: Bad disk identifier

GFID does not understand the disk specifier you have given to the system.

EXAMPLE: \$LIST @

Error Code 0501 Gfid says: Name too long

The file name you have entered is more than 31 characters long.

Error Code 0502 Gfid says: Illegal Extension

You have tried to save a file with a file name extension longer or shorter than the mandatory two characters in length or in some other way illegal.

EXAMPLE: <2<Frammis.A

Error Code 0503 Gfid says: Name null or weird

You have given a bad file name to the system. This message will also be generated if you enter NO file name to the SAVE command. A "bad" file name is any name not acceptable to the system.

EXAMPLE: \$RENAME <2<PHONE.DT <2<+.DT

(The second file name, +.DT, is illegal.)

#### PRINTER DRIVER ERROR MESSAGES

The system gives these messages for the kinds of errors that can occur when you are using the printer driver.

Error Code 0901: You haven't defined parameters for that printer.

The printer driver doesn't have a set of parameters for the printer you requested.

EXAMPLE: \$Printer Terminet

but you have not yet told Printer what a Terminet is.

Error Code 0902: You have already defined parameters for that printer.

You have tried to define parameters for a printer that Printer already knows about. Use the DEL and NEW commands. (You might also have tried to rename a device with its own name.)

Error Code 0903: Please give me a printer name!

You have tried to use a printer driver command, but have not specified which printer the printer driver is to rename, define, delete, etc. Use the command again, but on the same line give a valid printer name.

Error code 0904: I can't do that to that printer name!

You tried to delete, rename, or view the Null or Screen "printer." Can't be done.

#### OTHER ERROR MESSAGES

Error Code 0201 I can't run that file

You have asked the system to run a system overlay file. Only the system itself may invoke an overlay.

Error Code 0202 Nothing to run!

You have used the START or REENTER command to begin execution of a machine language program. The system, however, believes

that there is nothing in memory to execute.

#### Error Code 0203 What?

A general purpose error message indicating that the system does not understand what you are saying.

EXAMPLES: \$ (a line of spaces) \$<%<FILE

Error Code 0205 I don't know what to do with that file

You have typed a file name after a system prompt, but the system cannot run that file or use it as a command file. You will get this error message if you try to invoke any file with the extension .DT, .SY, or .OV.

Error Code 0206 I don't have enough memory to do that!

You will get this message if you try to load into memory a file that checks for the top of memory and finds that it's too big to fit into the available memory space. For instance, if you try to load the MACRO-88 assembler (which always checks for the top of memory when it is first loaded) into 16K or less of memory, you will get this message.

Error Code 0300 I can't find that file

Whenever the system fails to identify an input as a command or file invocation, it issues this error message. If the system is confused by an input, it usually assumes that you have asked for a file that it is not able to find. Make sure that you have spelled your input correctly.

Error Code 0301 I can't access that device!

Exec has failed to understand the disk specifier you used when you asked it to locate or create a file.

EXAMPLE: \$EDIT <2<File <?<Oops

Exec cannot tell where you want to put the output file.

Error Code 0302 Error Code 0303 Disk directory unreadable!

The system believes that the disk directory has been destroyed. No file on a disk can be accessed if the disk directory is invalid. Try again in another drive. You have probably lost all access to the data on the disk, however. Possible cause-- interrupting disk I/O while the disk directory was being updated.

Error Code 0306 I can't read the directory--no disk or door open

No write or read operation will take place to or from a disk

while the door is open on the drive containing that disk.

(Error 0306)

This is the same error as "I can't read the directory-no disk or door open." However, in this case, the system cannot find the System Disk. Make sure that the System Disk is in the System Drive and that it is in the drive correctly.

Error Code 0308 Data Transfer Error

Checksum error -- memory error on controller board

Error Code 03FF Disk directory destroyed!

The system thinks that your disk directory is no good. Try again in another drive. No files on a disk may be accessed if the disk's directory is bad. Possible cause--interrupting disk I/O while the directory was being updated.

Error Code 0504 I can't: the directory is full

You have tried to save a file on a disk whose directory is full. The directory is of a fixed size and has a finite amount of room for file names. Try saving the file on another disk. Or you may delete files from the full disk, pack the disk, and try again.

Error Code 0505 I can't: the disk is full

You have tried to save a file on a full disk. Try to save the file on another disk. Or you may delete files from the full disk, pack the disk, and try again to save the file on the disk. Error Code 0506 I can't rename across drives: use COPY

You have tried to use the RENAME command on files on different disks.

```
EXAMPLE $RENAME <2<Printer.GO <3<TELETYPE.GO
```

You must use the COPY command when renaming files across disk drives.

Error Code 0507 No new extension given

You have tried to rename or copy a file, but it is not clear what the extension of the new file name will be.

Error Code 0508 I can't do that to a system file

You have tried to use one of the following commands on a system file: RENAME, COPY, PRINT, TYPE, DELETE, or EDIT.

Error Code 0509 "<?<" is not allowed here

You have used the disk specifier ? when it is meaningless, e. g. as part of the file specification of an output file.

EXAMPLE: \$RENAME <?<SysGen <?<Start-Up

Error Code 0600 That file already exists

You have tried to save a file under a name that already exists on the specified disk.

EXAMPLE: (You have a file on the disk in drive 2 named BOOK.) \$SAVE,<2<BOOK

The system will tell you "That file already exists."

Error Code 0601 That file does not exist

Some areas of the system will issue this message if you try to access a non-existent file. Make sure that the file you have requested does exist and is on the disk in the drive that you have indicated.

Error Code 0701 Output file not specified

Some system software requires that you specify an output file. If you don't give an output file, the system will not know where to put the data you are working with.

Error Code 0702 Output file already exists

You have given as an output file a file name that already exists on the disk specified.

Error Code 0703 Input file not specified

Some system software requires you to specify an input file. If you don't specify an input file, the system will not know where to get the data you want to work with.

Error Code 0704 I can't edit that file!

You have tried to edit a file that cannot be edited! You may not edit any file with a load or start address (a system file, a system overlay, or any machine language file).

Error Code 0705 Input file does not exist

You have asked for an input file that does not exist.

Error Code 0706 I can't have two files open OUT on the same device!

You have asked the Assembler to create an output file on a disk that already has an output file open! They must be put on

disks in separate drives.

(Cmdf abort)

The system has tried to use a file as a command file but has been unable to do so. This may be due to an illegal command, a bad file invocation, or an unrecognizable entry in the command file. If you try to invoke an unrunable file from the system level (such as a text file or a BASIC program whose file name does not contain the .BS extension), the system will try to use the file as a command file, but will not be able to do so, and will then give the error message "(Cmdf abort)."

Use of a valid command file is terminated (and this error message displayed) when you type a Control-Y or the command PACK or INIT appears in the command file.

## Appendix B

## ASCII CHARACTER SET

Since the only data that your machine understands is binary data (1s and 0s), all information within the computer must be stored in that form. All of the characters that can be typed from your keyboard (letters, numbers, symbols, control characters, etc.) must have a binary value assigned to them in order for your machine to store them. The code that is most commonly used in assigning values to these characters is called the ASCII code (American Standard Code for Information Interchange). Every symbol you type from your keyboard has an ASCII value and is stored in that form within the machine.

Take a look at the ASCII Character Set table that follows these paragraphs. All of the symbols that you can enter from your keyboard are listed, along with their ASCII values (in hexadecimal).

Besides the letters, numbers, and symbols that appear on your monitor screen, special characters, called control characters, can be entered from the keyboard. You type these characters by holding down the Control key on the keyboard (labeled CTRL) and typing the letter in any column in the table below which is in the row corresponding to the control character. A tab control character, for instance, can be entered either by hitting the TAB key on the keyboard or by typing a Control-I, that is, by holding down the CTRL key and typing an i (or an I).

The control characters perform various functions in the system. Some of the control characters used by the disk operating system are:

Control-I

Same as TAB key (tabs cursor to the right eight spaces)

Control-L

Form Feed (clears screen and places cursor in upper right hand corner of screen)

Control-M

Carriage Return (advances the cursor one line on the screen)

Control-Q

Same as up arrow key

Control-R

Same as down arrow key

Control-S

Same as right arrow key

Control-T

Same as left arrow key

Control-W

Deletes one word

Control-X

Deletes one line (same as CANCEL key on some keyboards)

Control-Y

Interrupts current process

Control-Z

Enters front panel display mode

NOTE: Control characters are not visible when you use them within file names. They will be displayed as Greek characters, however, when that file name is shown in a disk directory display. Each control character will be displayed as the Greek letter that appears in the same row as the control character in the table below. EXAMPLE:

The control character "SOH" (Control-A) will appear as the Greek letter Beta ( $\beta$ ).



## ASCII CHARACTER SET

The following table gives the ASCII code for your machine's character set. The ASCII values are given in hexadecimal form.

Control Characters	Greek Letters	Other Characters			
00 NUL	80 α	20 SP	40 @	60 ` grave	
01 SOH	81 β	21 !	41 A	61 a	
02 STX	82 γ	22 "	42 B	62 b	
03 ETX	83 δ	23 #	43 C	63 c	
04 EOT	84 ε	24 \$	44 D	64 d	
05 ENQ	85 ζ	25 %	45 E	65 e	
06 ACK	86 η	26 &	46 F	66 f	
07 BEL	87 θ	27 ' apos	47 G	67 g	
08 BS	88 ι	28 (	48 H	68 h	
09 HT	89 κ	29 )	49 I	69 i	
0A LF	8A λ	2A *	4A J	6A j	
0B VT	8B μ	2B +	4B K	6B k	
0C FF	8C ν	2C ,	4C L	6C l	
0D CR	8D ξ	2D -	4D M	6D m	
0E SO	8E ο	2E .	4E N	6E n	
0F SI	8F π	2F /	4F O	6F o	
10 DLE	90 ρ	30 0	50 P	70 p	
11 DC1	91 σ	31 1	51 Q	71 q	
12 DC2	92 τ	32 2	52 R	72 r	
13 DC3	93 υ	33 3	53 S	73 s	
14 DC4	94 φ	34 4	54 T	74 t	
15 NAK	95 χ	35 5	55 U	75 u	
16 SYN	96 ψ	36 6	56 V	76 v	
17 ETB	97 ω	37 7	57 W	77 w	
18 CAN	98 Ω	38 8	58 X	78 x	
19 EM	99 √	39 9	59 Y	79 y	
1A SUB	9A →	3A :	5A Z	7A z	
1B ESC	9B ←	3B ;	5B [	7B {	
1C FS	9C ↑	3C <	5C \	7C	
1D GS	9D ÷	3D =	5D ]	7D }	
1E RS	9E Σ	3E >	5E ^carat	7E ~tilde	
1F US	9F ≈	3F ?	5F _u'line	7F DEL	

The characters 21 through 7E will print on a printer. However, some printer font character sets will vary slightly. The characters in positions 27, 5C, 5E, 5F, 60, 7B, 7C, 7D, and 7E may be different. An upper-case-only printer will print only characters 21-5F (characters 60-7E will be printed as 40-5E in most cases).



## Appendix C

## THE SYSTEM 88 GRAPHICS CHARACTER SET

We have already mentioned the ASCII character set (see Appendix B, ASCII CHARACTER SET), which consists of all of the symbols you can type on your keyboard. This set includes upper and lower case letters, numbers, and punctuation--all the characters that appear on the keyboard--plus control characters and some other symbols which can also be typed in from the keyboard. There is another type of character you can display on the monitor screen--graphics characters.

Graphics characters are symbols that you cannot type from the keyboard, but that BASIC or assembly language programs can cause to be displayed on the screen. You use graphics characters to display data that does not lend itself to expression in letters and numbers. For instance, BASIC uses graphics characters when it plots data on the screen with the PLOT function. You too can use graphics characters to graph data, or you can use them to draw lines, geometrical shapes, business-form lines, bar charts, etc.

This appendix discusses how the system forms these characters on the screen by "turning on" tiny sections of character matrices, and how ASCII and graphics characters differ in the way they are represented as binary code in machine memory. It tells you what the various graphics characters are and how you can use them in BASIC or assembly language programs. The following discussion assumes some familiarity with machine language.

## 1.1 REPRESENTING GRAPHICS AND ASCII CHARACTERS

Although graphics and ASCII characters are formed in the same way on the screen, they are represented in different ways within the machine. You already know that every ASCII symbol is associated with a number, its ASCII code. This binary code is one byte long (that is to say, eight bits of 1s and 0s). When the machine sees this number, it knows which character to display on the screen. Every graphics character also has a code associated with it. The machine knows whether a character is ASCII or graphics by whether there is a 1 or a 0 in the "top" bit, the bit farthest to the left in the byte. ASCII characters have a 1 in the top bit; graphics characters have a zero in that position. The next bit to the right may be either 1 or zero without affecting whether the character is graphics or ASCII.

## EXAMPLES:

The ASCII code for the question mark symbol, ?, is the

hexadecimal number 23 (35 in decimal). In binary:

```
00100011
```

The top bit must be set in order for the machine to recognize this as an ASCII character and therefore to display it as a ? on the screen. So the above number becomes:

```
10100011
```

or the hexadecimal number A3 (163 in decimal). To tell the machine that a character is an ASCII character, therefore, you make the top bit a 1 in the binary representation of its ASCII code. This is the same as adding 10000000 to its binary code--which is the same as adding 80H to its hexadecimal ASCII code, or adding 128 decimal to its decimal ASCII code. Because the second bit from the left may be either 1 or zero without affecting whether the character is recognized as ASCII or graphics, the ASCII character above can also be represented by the binary number:

```
11100011
```

The graphics character code for the graphics symbol is the hexadecimal number 23 (35 in decimal). The machine knows that this is not an ASCII character because the top bit is a zero:

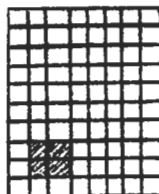
```
00100011
```

If the top bit were 1, the machine would think that the ASCII question mark symbol was being asked for. Again, since the second bit from the left does not affect whether an ASCII or a graphics character is displayed, the same graphics character would be selected by the binary number:

```
01100011
```

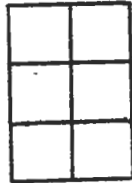
## 1.2 HOW THE SYSTEM FORMS CHARACTERS ON THE SCREEN

Every letter, number, and symbol that you see on the monitor screen is actually composed of many small white dots on a dark background. You can think of the screen as a grid consisting of a large number of small rectangles all the same size. Each rectangle is a screen location where a symbol can appear. Each of these rectangles is itself a grid of small squares or dots. The symbol that you see on the screen is the pattern of dots that have been "turned on," against the background of the dark dots of the rectangle that are "off." For example, the period that you see at the end of a sentence is actually a rectangle of about 80 dots, all of which are dark (or "off") except for four dots near the left bottom corner of the rectangle, which are bright (or "on"). This pattern creates a small white square which you see as a period:

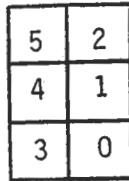


### 1.2.1 How the System Forms Graphics Characters on the Screen

When the system displays graphics characters on the screen, it divides each of the rectangles mentioned above into a grid of six "cells" (2 across and three down):




Each of these cells can be turned "on" or "off" in any combination to form the 64 graphics characters. For example, if all six cells of a rectangle are bright ("on"), you will see a white rectangle on the screen. If all of the cells are dark ("off"), you will not see anything. The cells are numbered in this way:



We've mentioned that each graphics character has a number associated with it. This number tells the system which cells are on and which are off. It does this by presenting the system with a binary number in which bit position corresponds to cell number: bit #0 corresponds to cell #0, bit #1 corresponds to cell #1, etc. A zero says "turn this cell ON"; a 1 says "turn this cell OFF."

#### EXAMPLE:

To select the graphics character , use the hexadecimal number 34 or 74:

00110100

or

01110100

Remember, the first bit on the left must be zero to indicate a graphics character, and the second bit from the left makes no

difference in selecting an ASCII or graphics character.)

This tells the machine: turn on cells 0, 1, and 3; turn off cells 2, 4, and 5.

off	off
off	ON
ON	ON

One last note: ASCII characters are always surrounded by a narrow band of dots that are "off." This allows letters and numbers, etc. to be slightly separated from one another, just as they are when printed by a typewriter or printer. Graphics characters, on the other hand, are contiguous-- their cell parts are lighted all the way to the edge-- so that they can form continuous graphic elements, such as lines.

See Figure C-1 for a complete listing of the graphics characters along with the hexadecimal and decimal representations of their codes.

### 1.3 USING GRAPHICS CHARACTERS IN BASIC AND ASSEMBLY LANGUAGE PROGRAMS

Now that you know how to select the graphics characters you want, you will want to know how to use them in your programs. Remember that we divided up the screen into an invisible grid of rectangles. Each one of those rectangles is a potential location for a graphics or ASCII character. Each rectangle has a memory address associated with it. The address which selects the first location on the first line of the screen is 1800H. Since there are 64 character positions on one line of the screen, the beginning address of the screen plus 64 will give you the first location on the SECOND line of the screen, and so on. The entire screen contains 1024 character positions, so the address selecting the last location at the end of the screen is 1BFFH (1800H + 1023 decimal).

Whether you are programming in BASIC or assembly language, the basic technique for getting graphics characters on the screen is the same: you must determine the memory address selecting the spot on the screen where you want your graphics character to appear. Then move the character to that address. The details of doing this depend on whether you are programming in BASIC or in assembly language.

### 1.3.1 Using Graphics Characters in BASIC

BASIC understands only decimal numbers. Therefore you must convert all hexadecimal numbers to decimal. The first address of the screen in decimal is 6144 (1800H). The last screen address (bottom right corner of the screen) is 7167. Use the BASIC POKE function to put graphics characters where you want them on the screen. (Remember to convert the graphics character codes to decimal too.) The form of the POKE function is:

POKE address,expression

"Give POKE the address (in decimal) of the spot where you want your graphics character to appear, then give it a number or expression which selects the graphics character you want.

EXAMPLE:

```
10 POKE 6213,0 \REM place a white rectangle on fifth
20 REM position of second line of screen (beginning
30 REM address of screen + 64 [2nd line] + 5 [5th
40 REM position])
```

Be careful not to give POKE an address less than 6144 or greater than 7167. Such an address is not a video screen address; poking odd numbers into an undetermined memory location is almost sure to get you into trouble.

### 1.3.2 Using Graphics Characters in Assembly Language Programs

Put graphics characters on the screen via assembly language in the same way that you would insert data into any memory location. Use the hexadecimal representations of the graphics character codes and the screen location addresses.

## EXAMPLE:

```

LXI    H,Screen      ; Point to screen address
LDA    GChar         ; Graphics character
; Display on screen by putting into memory location
; pointed to by HL registers.
MOV    M,A

```

## 1.4 SAMPLE BASIC AND ASSEMBLY LANGUAGE PROGRAMS

Both of these programs display the System 88 graphics characters.

## 1.4.1 BASIC Program

```

10 REM Display the 64 graphics characters on the screen
20 A=6144\REM First screen address
30 PRINT CHR$(12)\ REM Clear the screen
40 FOR C=0 TO 63\ REM C=graphics character
50 POKE A,C
60 A=A+17 \REM Increment screen address for well-spaced
70 REM character display
80 NEXT
90 PLOT 45,23,0\PRINT "SYSTEM 88 Graphics Characters!"
100 PLOT 0,0,0\REM Move cursor to less obtrusive spot

```

## 1.4.2 Assembly Language Program

```

;
; Sample program displaying the 64 graphics characters
; on the video screen
;
User    EQU    3200H      ; Beginning of user memory
        IDNT   User,User
        ORG    User
;
        LXI    H,1800H   ; First screen address
        XRA    A
Start   STA    GChar     ; First graphics char = 0
        LDA    GChar     ; Check character to make
        CPI    40H      ; sure isn't > maximum
        JZ     EndIt
        MOV    M,A       ; Put on screen
        LXI    B,11H    ; Increment screen address by
        DAD   B         ; 17 for well-spaced character
        INR   A         ; display
        STA    GChar
        JMP   Start
EndIt   RET             ; Return to system Exec
;
GChar   DB    0
        END

```



Upper number decimal, Lower hexadecimal.

White is bright, black dark.

0 00H	8 08H	16 10H	24 18H	32 20H	40 28H	48 30H	56 38H
1 01H	9 09H	17 11H	25 19H	33 21H	41 29H	49 31H	57 39H
2 02H	10 0AH	18 12H	26 1AH	34 22H	42 2AH	50 32H	58 2AH
3 03H	11 0BH	19 13H	27 1BH	35 23H	43 2BH	51 33H	59 3BH
4 04H	12 0CH	20 14H	28 1CH	36 24H	44 2CH	52 34H	60 3CH
5 05H	13 0DH	21 15H	29 1DH	37 25H	45 2DH	53 35H	61 3DH
6 06H	14 0EH	22 16H	30 1EH	38 26H	46 2EH	54 36H	62 3EH
7 07H	15 0FH	23 17H	31 1FH	39 27H	47 2FH	55 37H	63 3FH

Fig. C-1 Graphic Character Set

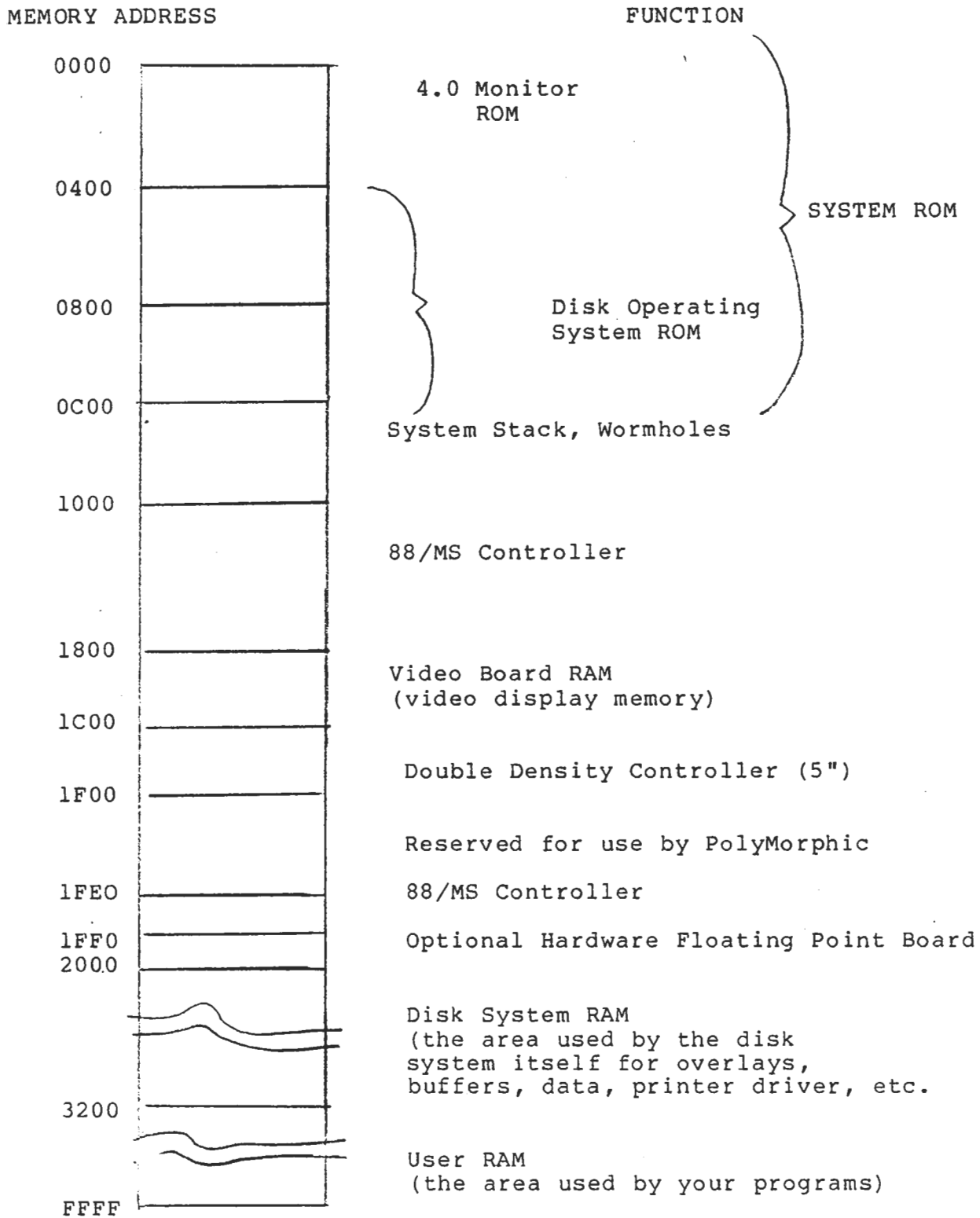


## Appendix D

## SYSTEM 88 MEMORY MAP

The following is a graphical representation of the allocation of the System 88 memory. Memory location addresses (in hexadecimal) appear on the left of the figure; the functions of the various areas of memory appear on the right. The term "ROM" refers to Read-Only-Memory: memory areas which cannot be written into. "RAM" is the term for Random-Access-Memory: memory areas which may be both written into and read from.

SYSTEM 88 MEMORY MAP



## Appendix E

## THE MONITOR; FRONT PANEL DISPLAY

## 1. Operating System ROMS

A computer works by manipulating or processing data by means of programmed instructions. The data that the computer processes is transient material: you put it into the system, and the system processes it back out. Most of the programs you use are also transient: they may be conveniently stored on disk, but you decide when you want to put them in and use them.

Some programs, however, are not transient. They are "part of the system": available at all times and usually used automatically by the system itself in doing its work. We have already talked about the "system software" pre-recorded on the System Disk. The System Disk always resides in the System Drive, so the system software recorded on it is always available. The system itself decides what system software it needs to use in order to obey your commands; its use of system software may not even be particularly apparent to you. Collectively, these programs used automatically by the system in obeying your commands are called the operating system.

Now, there is another group of operating-system programs besides the System Disk, programs that are even less transient than those on the System Disk. This is the group of programs permanently recorded on three memory chips inside the main unit. These three chips differ from all the other memory devices inside the main unit in that they are read-only memories or ROMs. The system can read the information recorded on them, but it cannot erase that information or change it in any way.

It is very important to have a part of memory that cannot be erased or changed, for without it, the system would suffer total amnesia every time it was turned off. The memory contents of the random-access memory units or RAMs are lost when the power goes off. If the system had no permanent memory, when you turned it back on it would wake up with no recollection of how to get to work. You would need to have some means of teaching it how to accept information before you could even begin to use it. For instance, it would not have routines for bringing system software in off the System Disk. Some small computers actually have "front panel switches"-- literally switches in the front panel of the main unit, that are set by hand, one by one, to force the individual microscopic electronic devices inside the main unit into the desired states. This hand-feeding is a tedious

business.

"Firmware" routines like those on the System 88 ROMs allow the system to wake up remembering several crucial things. (The term "firmware" is a compromise between "hardware," the mechanical, permanent components of the machine, and "software," the programs and other relatively transient data.) Thanks to the ROMs, it wakes up knowing how to bring in data from disks. It knows how to accept and correctly interpret input from the keyboard. It knows how to display a character on the video screen. And the ROMs make available several other routines-- "utility programs"-- that the system makes use of without your even necessarily being aware of it.

In addition to these programs, system firmware includes routines collectively called the Monitor. The Monitor incorporates some firmware programs that the user can use directly. These independent functions are the front panel mode and the tape loader. Of special interest is the front panel mode.

## 2. Front Panel Mode

As we said, firmware makes it unnecessary to start out by using front panel switches to force the microscopically small binary devices into desired states. But it is useful at times to be able to go right down to the lowest machine level and directly force individual bytes to take on desired values. The Monitor makes this job easy, by providing a front panel display and letting you reach out and change the items shown in the display. We should note that this ability to reach down into the machine and forcibly change single bytes can be dangerous.

The front panel display is given its name because it is a video monitor display that conveys the kind of information a conventional front panel might give, but in a more useful form. The display shows all of the CPU registers, the "workspace" of the CPU, and a memory "window." The "workspace" is the areas of memory pointed to or addressed by the register pairs, including the program counter and stack pointer. The workspace display shows, therefore, the program area, stack area, and data areas pointed to by HL, DE, and BC. The memory "window" is an 8 X 8 block of memory (64 memory locations) that is displayed, both contents and addresses, on the bottom of the screen. This window can be used to view a selected area of memory or to point to data areas to be modified.

The display is updated each time a command is executed or the registers are modified in any way. Thus, it always reflects the contents of the registers and memory at any instant, just as if it were hardwired into the CPU and the memory and address and data buses. Visible in the display are:

Contents of CPU registers: program counter (PC), stack pointer (SP), accumulator (A), and general purpose registers.

Contents of memory areas pointed to by general purpose registers: program area, stack area, and the areas pointed to by BC, DE, and HL.

A movable memory window which shows 64 contiguous memory locations: their address and their contents.

The status of the carry, sign, and zero flags, decoded into an easy-to-read form.

Whenever you look at the front panel display, you also have at your disposal a set of commands that let you change what you see. Hitting keys on the keyboard will allow the operator to:

Interrupt a running user program to bring up the front panel display.

Single step (run a program one instruction at a time), run with breakpoints, or return to full-speed execution of the user program.

Move the memory window to view the contents of any address.

Enter single bytes or long strings of bytes in hexadecimal form into memory, with instant verification of entered data and easy error correction.

Trace byte-reversed (lower-significance byte first) address pointers in memory by moving the memory window to the given address.

Move the memory window to point at the program, stack, or data areas currently being used by the user program.

The Monitor commands are primitives; when used in combination, they provide a powerful system for manipulating memory data and debugging machine language programs. There is, for example, no command for setting the contents of any given general register. Instead, there is a command for pointing the memory window at the place in the system stack where the contents of a given register are stored. This enables you to modify the contents of the register using the rest of the commands, such as the Jumbo (J) command, which lets you enter a full address in its normal byte order instead of the usual byte-reversed order of 8080 addresses. Another example is using the I (indirect) command after pointing the window at the register save area on the stack. This points the window at the memory area that the register points at.

In other words, if the register is the program counter, a sequence of "SPJ" will leave the window displaying the program area. The program area can then be modified, using the full power of the front panel commands.

## 2.1 Sample Front Panel Display

```

PC 008C 0C 0C 7E B7 C2 8B FE 8C
SP 0FFA FF 8C 00 FB 7C 2F 31 A0
HL 0C0C 49 48 D5 10 08 56 C6 DA
DE 0C51 21 00 88 A7 BA DC 0F 1F
BC 0000 FF FF FF 31 00 10 06 FF
AF FF86 C Z

```

```

1FE3 FF FF FF FF FF FF FF FF
1FEB FF FF FF FF FF FF FF FF
1FF3 FF FF FF FF FF FF FF FF
1FFB FF FF FF FF FF FF FF FF
2003 00 AA FE 3A 40 21 CE 8F
200B 76 C2 3C 03 2A S7 0C C9
2013 26 4F 3A 29 2A 44 0C C9
201B B9 83 B2 16 F0 C8 33 BA

```

The upper half of the front panel display shows the contents of all registers and the status of the carry, sign, and zero flags. It also shows what is stored in the memory locations addressed by the registers and the contents of adjacent memory locations. The bottom half shows a block of sixty-four memory locations, both addresses and contents. First we'll consider the upper half in some detail.

The left-hand column of the upper part of the display shows what item in the processor architecture we're looking at. The first line of the display begins with the letters PC, for program counter. To the right of PC is 008C. This indicates that the current contents of the program counter are 008C hex. In other words, the program counter is currently addressing or pointing to the memory location with address 008CH. Now note the up-pointing arrow at the bottom of this half of the display. The arrow indicates the column of display that shows the contents of the memory locations currently addressed by the registers. So we know that PC is pointing to memory address 008C hex, and that that memory address now contains the value B7. Since we are considering the program counter, what this tells us is that the first program instruction that will be executed when execution resumes is B7, which happens to be OR A.

As you can see, there are several other items in the PC line, the top line of the display. In addition to the contents of address 008C, we are being shown the contents of the three addresses prior to that location (to the left) and the four locations after that location (to the right). So we know what instructions have just previously been executed and what instructions are coming.

The other lines are much the same. The stack pointer is currently pointing to address 0FFA, which contains FB. Previous and subsequent pop instructions would pop the other addresses



shown. So here we can see an eight-entry chunk of the stack. HL is currently pointing to address 0C0C (or, to be exact, the value in HL is 0C0C hex), and 10 is stored at that address. DE and HL contain 0C51 and 0000 hex respectively, and the contents of those addresses (assuming that those values are to be considered addresses) are shown, as well as the contents of the neighboring addresses. The last line is the accumulator and flags (AF). First we see the program status word (PSW). The accumulator holds the value FF hex, and the flag byte value expressed in hex is 86. And we see three flag bits, carry, sign, and zero. C indicates that the last accumulator operation resulted in a carry; if it had not, this space would be blank. The next space is sign; it is blank, indicating that the last operation affecting that flag left it reset to positive; if negative, the letter M for minus would appear in this place. Z in the last place indicates that the last operation affecting the zero flag left it set to indicate that the result of the operation was zero. For non-zero, this space would be blank.

Now for the bottom half of the display. Here we see the contents of a continuous piece of memory space, sixty-four locations in all, in eight lines of eight locations each. The address of the first location in each line is shown at the left of the display.

Also at the left is a right-pointing arrow, pointing at the contents of a single memory location. This is the memory item that we can now force to take on a desired value. In the sample, we can change the value 00 hex, stored at address 2003 hex, to any desired value. We can also see the contents of the sixty-three surrounding memory locations.

Note that the arrow points directly at the item to be changed. Suppose you want to change CD, the value stored at address 2002 (the last item in the previous line-- i.e., the item just previous to the one the arrow now points at). To move back one address, hit the backspace. The arrow does not move; instead, CD will move down and over to be next to the arrow, all other items will move correspondingly, and the values in the address column will change accordingly.

## 2.2 Using the Front Panel Mode

Suppose we wish to construct a simple program in an available location in RAM. The demonstration we will use is a video display test which loads each location of video card memory with the less-significant address byte of each screen location. This has the effect of displaying all possible characters and graphics patterns on the screen in a cyclic group of 256 characters. The display is thus repeated four times.

The program looks like this in assembly language:

```

        LXI      H, 1800H      ; start at top of system
screen
LOOP:   MOV     M,L           ; put out each location's low
                                address byte
        INX     H             ; next location
        MOV     A,H          ; get high addr byte for
comparison
        CPI     lCH          ; is it off the screen yet?
        JNZ     LOOP         ; no - keep going
HLTAGN: HLT     HLTAGN       ; yes - OK, we're done, stop
        JMP     HLTAGN       ; go back to the HLT.

```

In hexadecimal machine code:

```

21 00 18 75 23 7C FE 1C C2 83 0C 76 C3 8B 0C

```

assuming that we want to load it at C80H, which is a free space in the system RAM. The problem is to correctly load this hex into the RAM at that address and then send the CPU off executing it.

Turn on the System 88 and push the front panel Load button. To use the front panel mode, type ENABLE first, then push Control-Z (hold down CTRL and push Z). The front panel display should appear on the screen. Note the memory window at the bottom. Remember, the window is a 64-byte section of memory which shows, in hex, the current memory modify location (i.e. the location that would be affected by a modification command), plus locations before and after the current modify position. The byte actually at the current modify position is indicated by a right-pointing arrow at the left center of the block. The address of this byte and the leftmost byte in each row is displayed at the far left of the screen, also in hex.

Now, to enter the test program into the RAM, we first point the window at the desired address:

LC80 (CR)

(CR) means "hit the carriage return." The display should now show 0C80 in the address next to the arrow. To enter the bytes of our program, we can simply type the hex for each instruction followed by a space. When hex is being entered, the termination character for each byte is interpreted as a valid command. In this case, the space indicates that the window pointer is to be incremented: each byte goes into the succeeding location. The program entry looks like this:

```

21(space)00(space)18(space).....76(space)C3(space)8B(space)
0C(space) where (space) means, of course, a blank space from
the space bar.

```

Suppose you make an error while entering data into memory. The window (i.e. the exact point where modification will occur) may be moved back one location with the backspace. You can correct the erroneous byte, then (after the usual space) enter the rest

of the program bytes. If you detect the error before typing a termination character, you need only continue typing in the hex for the correct code until the last two hex characters shown at the bottom of the screen are correct. The hex input routine used by the command interpreter shifts hex characters into a two-byte register from the bottom, so when it returns to the calling program on receipt of the termination character, it leaves only the last four hex characters in the register pair. In the case of hex input to the window location, the program uses only the bottom two nybbles. It ignores any previous hex digits. The use of the last hex characters typed in is built into all the other commands that expect a hexadecimal input of some kind.

One of the commands which expects a hex input is the J command. J stands for Jumbo, a mnemonic that indicates that a double word is to be entered at the current window location. The J command is followed by up to four hex characters and a carriage return, thus:

JF800 (CR)

The result is that the contents of the two locations at and following the window are modified to contain the "byte-reversed" double word that was entered. Actually, as mentioned above, only the four last characters typed in for hex are used, so if an error is made on entry, just keep typing until all four of the last characters are perfect. Since the register that the bytes are shifted into starts out with all zero contents, a small hex number need not be typed in with leading zeroes (unless, of course, it is being re-typed after an error). The way to enter an address of 0C80, then, is to type the J followed by C80 followed by a carriage return: JC80 (CR). One important fact about the J command is that it does not move the window pointer. The reason for this is to allow the use of the I command immediately after a J. Sometimes this combination can be useful.

The I command is the "indirect" operator. It takes the two bytes at and following the window location and puts them into the window pointer. It "jumps" to the address currently shown at the window pointer. This is a very useful function: for tracing programs that do JMPs or CALLs, place the window pointer over the address of the JMP or CALL and then type I. It is also used immediately after a "J" as a check of the address entered. If the address is correct, the window will show the data that are supposed to be pointed to.

It should be pointed out that the I and J commands work with double words in memory that are stored in what is known as "byte-reversed" format. The 8080 puts the more significant byte of an address stored in memory into the high-order (greater-significance) register of a pair when POP or LHL instructions are executed. PUSH and SHLD instructions operate similarly. Addresses in JMP and CALL instructions also follow this rule. Although it seems logical to arrange addresses this way, it is normal to enter data into memory while incrementing

addresses between bytes entered. This, unfortunately, means that the low-order address byte is typed in first. Addresses are also displayed backwards in the normal representation of data in memory: addresses increasing to the right. The seemingly backwards storage of addresses has come to be called "byte-reversed."

Now that the program has been entered correctly, we would like to run the program. The first thing to do is to set up the program counter to point at the first instruction of the program. To do this we will use one of the S commands: SP, which will point the window at the area on the system stack where the program counter is stored. Now of course, the actual program counter could not be stored on the stack, because the program we are running that displays the front panel and interprets our commands is moving the PC up and down in the monitor. But the program counter we will modify is the one that will be stored into the "real" program counter when we want to execute the program. Thus as far as we are concerned, the actual program counter is stored right there in memory, along with the values of all the other registers. Since the stack may have any value in it when we pressed control-Z, the locations actually used to store the register values are unknown. The monitor, however, keeps track of these locations and will point us at any one we want if we use the S type of command.

So, to set up the program counter, we point the window at the proper place on the stack with an SP command, and then do a J:

SPJC80(CR)

The front panel display at the top of the screen should now show the 0C80 we just entered in the PC register. The area to the right of the PC double word shows the memory pointed to by the PC, which is the program area. It should show part of the program we have loaded. The arrow at the bottom of this part of display points upwards at the actual locations that all the register pairs point to. The 21 hex that was the first opcode of our program should be visible above this arrow in the field next to the PC.

The memory window should also show the new PC value, except it will be backwards because of the "byte-reversed" address format. The window should show 80 followed by an 0C. Now, to check this value, let's see if the PC actually points at the program. Press I. The window should show the first instruction again: the 21H. For one last check before we run the program, hit the carriage return; the window will scroll up one row (8 bytes). We could move backwards one row by typing a line feed. This gets closer to the address in the JNZ instruction that we want to test. Space down to the locations following the JNZ (following the C2) and press I. The window should point at the address we called "LOOP" in the symbolic assembly program. The instruction at this address was a MOV M,L, which is hex opcode 7C. The 7C should be in the memory window after the I command is typed.

If this last test works, we are ready to step the program through one cycle of its loop to see what happens. The program counter is still set up to 0C80, so press the single-step command key, X. The program counter will advance to 0C83 and the HL register pair will be loaded with 1800, the data from the second two bytes in the LXI H instruction. On the next single-step, the first byte will be transmitted to the video screen, but since the front panel display is replaced on the screen after the byte the instruction transfers, we do not see anything happen. The next instruction increments HL to 1801. Next, A gets the contents of H, then it is compared with 1C hex in the CPI 01CH instruction (FE 1C). Finally, since 18 does not equal 1C, and the zero flag is not set, the JNZ instruction goes back to 0C83 to continue the loop.

The program seems to work when single stepping, so the final test is to execute it at full machine speed. This can be done by pressing G. The entire screen should fill with the test pattern of consecutive ASCII characters and graphic patterns, in a cyclic replication four times down the screen. The single-stepping loses the first character in the upper left corner of the screen, though.

When the test pattern is verified, front panel mode may be re-entered at any time by typing Control-Z. The front panel will appear, showing the program counter just as it was, halted at the end of the program on the 76 (HLT) instruction. The HL pair should have the last screen address used: 1C00H.

To return to the disk operating system, put its warm-start address (403H) into the program counter and start execution:

```
SPJ403(CR)G
```

Warning! Don't tamper with system data unless you're prepared to deal with the potentially chaotic results.

### 3. Monitor Commands

The following list comprises the set of primitive operators or commands available in the front panel mode. You can enter front panel mode at any time by striking Control-Z. Further commands on the keyboard have effects which are immediately reflected in the front panel display. When you want to leave front panel mode, you can re-start the interrupted program where it left off, since the entire status of the CPU is saved on the current system stack upon entry to the monitor.

#### Control-Z

Interrupt currently executing program. Front panel mode is entered. Status of CPU (PC, SP, registers, flags) is saved on the system stack.

X

Execute the next instruction of the interrupted program and return to front panel mode to display results.

G

Go to the next instruction of the interrupted program and do not return.

Lxx..xx (CR)

Look at address xx..xx with the memory modify display. The variable-length address (up to four last hex digits accepted) is placed into the memory modify display pointer.

SPACE bar

Move the memory modify display pointer forward one address and redisplay everything.

BACKSPACE

Move the memory modify display pointer back one address and redisplay everything.

RETURN key

Move the pointer forward eight positions. This scrolls the display up one line.

LINE FEED

Move the pointer back eight positions. This scrolls the display down one line.

xx xx(any command)

The last two hex characters before the command are entered into the location pointed to by the memory modify pointer. The command is then executed.

Jxxxx (CR)

Jumbo data word (double-word) is entered in byte-reversed format at and following the memory modify pointer. The last four hex characters before the carriage return are used.

I

Indirect display. The two bytes at and following the memory modify pointer are placed into it in reverse order, so that if they represent the address in a JMP instruction, the pointer will be moved to that address.

SP (Program Counter)SH (HL)SD (DE)SB (BC)SA (Accumulator/flags)

Stack modification. Move the memory modify pointer to that address on the stack where the indicated register pair was stored on program interrupt. If the location at the memory modify pointer is modified, the register display will show the contents of the appropriate register as having changed, and when the G command is executed, program execution will continue with the new value. To enter a double-word, use the J command. A single byte may be inserted in one register of a pair by simply entering it for the lower register and by spacing once over the lower register to enter it into the upper register. You can modify data at the address pointed to by a register pair by using the I command to move the memory modify pointer to the appropriate area of memory.

U (or other illegal command)

Update the display. This can be used to watch dynamically changing events such as the real time clock counter being incremented in system memory, or an I/O buffer filling.





## Appendix F: 8813 Hardware

## Table of Contents

## SECTION 1 - GENERAL DESCRIPTION

## 1.1 CHASSIS LAYOUT

- 1.1.1 Power Supply
- 1.1.2 Disk Drives
- 1.1.3 Backplane
- 1.1.4 Minicards
- 1.1.5 Rear Panel
- 1.1.6 Switches
- 1.1.7 Drives

## 1.2 SPECIFICATIONS

- 1.2.1 Power Supply
- 1.2.2 Disk Drives
- 1.2.3 Backplane
- 1.2.4 Memory
- 1.2.5 Central Processor

## SECTION 2 - INSTALLING OPTIONS

## 2.1 SAFETY

- 2.2 Cabinet
  - 2.2.1 Removing the Cover
  - 2.2.2 Installing the Cover

## 2.3 DISK DRIVES

- 2.3.1 Removing the Filler Panels
- 2.3.2 Installing Disk Drives
- 2.3.3 Removing Disk Drives

## 2.4 BACKPLANE

- 2.4.1 Card Hold-down
- 2.4.2 Installing Cards
- 2.4.3 Removing Cards
- 2.4.4 Memory Cards
- 2.4.5 I/O Cards

## 2.5 I/O CONNECTORS



## Appendix F:

## 8813 HARDWARE: GENERAL DESCRIPTION

## 1.1 CHASSIS LAYOUT

Several components are mounted within the main chassis. Viewed from the front, the power supplies are to the left. One or two optional minicards may be mounted on the rear panel just above the power supply. The right section contains the backplane assembly, mounted in the rear, and up to three disk drives in front.

## 1.1.1 Power Supply

The power transformers are mounted in the left rear, and in front of them is the power supply printed circuit card (PC card). Along the left of the transformers are two cooling fans. The power supply supplies unregulated power to the backplane and regulated power to up to three disk drives. The drives are connected through three connectors mounted on the right rear corner of the card.

## 1.1.2 Disk Drives

The minifloppy drive offers the user the random access storage capability of large disk drives in a package about the size of most cassette tape units. In addition, they provide superior data integrity and faster throughput. Up to three disk drives may be mounted in the 8813 chassis. Each drive can store 89,600 bytes of information on a single five inch minidisk, or 358,400 on a double-sided, double-density minidisk. Attached to each drive is a PC card containing the control electronics.

## 1.1.3 Backplane

The backplane contains ten edge connectors spaced at 3/4" intervals along the PC card, for the insertion of ten S-100 bus compatible cards. Behind the edge connectors are seven bus termination networks and the components for the real-time clock. Power enters the backplane through four slide connectors mounted at the left rear edge of the card. The adjacent connector is used for the real-time clock signal and front panel controls.

The 8813 in its minimum configuration uses four card positions, leaving six free for additional memory or input/output (I/O) cards. The first position contains the disk controller card. Mounted behind it are the CPU card and one memory card. The rearmost position contains the Video Terminal Interface (VTI) card. Other memory cards may be installed between the first one and the VTI card.

**WARNING:** The CPU card must always be installed in the second or third slot from the front of the card cage in order for the system to operate properly.

#### 1.1.4 Minicards

Space is provided on the rear panel for mounting two optional minicards. An audio cassette interface minicard and a serial printer interface minicard are available. Both of these cards plug into the CPU card, without using up positions on the S-100 backplane.

#### 1.1.5 Rear Panel

All connections to other devices are made through connectors mounted on the rear panel. The left side of the panel as viewed from the front contains the power line connector and fuse. Two accessory outlets are provided for powering the video monitor and the optional audio cassette recorder. The power line voltage and fuse rating are marked on the serial number sticker on the rear panel. Use only the type of fuse and power source indicated; other types can cause severe damage to the computer.

To the right of the accessory outlets are the mounting holes for the minicards. These are covered by a sticker unless the minicards are installed. The sticker ensures proper air circulation by covering up the unused holes, and should not be removed except as provided in the minicard installation instructions.

The right side of the rear panel contains cutouts for various I/O device connectors. At the edge of the panel is a column of four cutouts for 25-pin "D" connectors. The top connector is the keyboard connector in the standard configuration. To the left are four cutouts for "UHF" coaxial connectors. The topmost connector is the video monitor output in the standard configuration. Left one column are four more cutouts for 25-pin "D" connectors. Next are two cutouts for 24-pin micro-ribbon connectors. These are normally used for connection to the IEEE-488 instrumentation bus (also known as GPIB bus, HP-IB bus, and ANSI standard MC 1.1-1975). Below these are cutouts for two 37-pin "D" connectors.

#### 1.1.6 Switches

The 8813 has only two switches. A key-operated switch is on the left. Above it is a red LED (light-emitting diode) power-on indicator. The pushbutton is marked "Load." When pressed, it causes the system software to be loaded off of drive 1.

### 1.1.7 Drives

Up to three disk drives may be mounted. The leftmost drive is drive 1, usually the system drive. The middle drive is 2, and the rightmost drive is 3. The red LED indicators on each drive light up when that drive is being used. If a system has fewer than three drives, matching filler panels are installed in place of disk drives.

## 1.2 SPECIFICATIONS

### 1.2.1 Power Supply

#### AC Line:

Voltage: 110-125 or 220-250 VAC  
Frequency: 50 or 60 Hz

#### DC Output (to backplane w/ 3 drives):

Pins 1 and 51	+7.5 to 10 VDC 15 A (max.)
Pin 2	+15 to 20 VDC A (max.)
Pin 52	-15 to 20 VDC 1A (max.)

### 1.2.2 Disk Drives

	Standard	DSDD
Data storage per disk:	89,600	358,600 bytes
bit transfer rate:	125,000	250,000 bits/sec.
Sector transfer rate:	25	90 sectors/ sec.
Latency (average):	100	100 ave. mSec.
Track step:	40	5 ave. mSec.
Seek ave.:	700	87 ave. mSec.
Number of tracks:	35	35
Sides	1	2
Sectors per track:	10	10 (20 simulated)
Bytes per sector:	256	512 (256 simulated)
Maximum hard error rate:	1 in 10 <sup>11</sup> bits read	
Mean time between errors:	8,000 hours	

The DSDD (double sided, double density) operates at a faster track step time and has out-of-sequence sector read capabilities due to an intelligent controller which also buffers tracks. The average latency time is thus effectively lower than shown, because multiple sector accesses may begin reading at any sector. Also, some reads require no disk access.

### 1.2.3 Backplane

Bus Type: S-100 (except no DMA) Number of Card Slots: 10 Card Spacing: 3/4" center-to-center Maximum Card Size: Length: 10" Height (0.5" from edge): 5.3" Height (center) 5.8" Bus

Termination impedance: 220 ohms

#### 1.2.4 Memory

Maximum addressable:	57,344 Bytes (56KB)
Access Time:	500 uS
Error Rate (per bits read):	1 in 1,000,000,000,000

#### 1.2.5 Central Processor

Type:	8080
No. of user-accessible registers (8-bit):	7
Data Word Size:	8 or 16 bits
Register to Register Add Time:	2 uS

## Section 2

## INSTALLING OPTIONS

## 2.1 SAFETY

Whenever working on the 8813 chassis, perform these three steps before opening the chassis:

- 1) Remove all disks from drives.
- 2) Turn off power with key switch.
- 3) Remove line cord from rear panel socket.

Do not insert or remove any cards or minicards with power on or line cord attached. If this precaution is not observed, portions of the chassis circuitry and cards will probably be destroyed. Remember, there are dangerous voltages in the chassis until the line cord is unplugged (even when the power key is turned off).

## 2.2 CABINET

The 8813 desktop version is enclosed in an attractive walnut cabinet. To avoid damage, do not set wet or sharp objects on it. The surface of the wood is waxed, not varnished; do not apply varnish. To maintain the cabinet, occasionally wax it with a little high-quality furniture wax and a soft, clean cloth.

A 6" airspace on either side of the cabinet is necessary for proper airflow. Air enters through the right bottom opening in the cabinet and exits through the left opening. If the airflow is blocked, internal temperatures may rise enough to cause faulty operation, reduce the operating life of the computer, or cause component failure. Do not run the computer when it is resting on a soft surface.

## 2.2.1 Removing the Cabinet

To get inside the chassis, you must remove the cabinet. Follow the safety precautions above (2.1), and place the computer on a table or workbench so that the front of the chassis overhangs by about 1". On each side of the front panel, just inside the cabinet, are two trim strips. Slide the two trim strips down until they are free of their mounting hardware. Then remove the four 10-32 screws and plastic washers attaching the front panel to the cabinet. Be careful not to scratch the brushed aluminum front panel. Four more screws and washers are located on the rear panel; remove these. Then grip the lower edges and pull upward, sliding off the cabinet.

## 2.2.2 Installing the Cabinet

To install the cabinet, place it over the chassis so that the wooden front mounting brackets on the cabinet are between the front and rear metal mounting brackets on the chassis. Then slide the cabinet down and forward until the rear cabinet edge is flush with the rear panel on the chassis. Secure the cabinet front and rear with the 10-32 X 3/4" machine screws and washers (plastic washers for mounting trim strips in front). Then slide the two trim strips on from the bottom.

## 2.3 DISK DRIVES

From one to three floppy disk drives are mounted in the 8813 chassis. Any remaining drive openings are covered by filler panels. These panels must be removed to install additional disk drives. The drives may be added by obtaining an add-on drive kit containing a drive and all the necessary mounting hardware and cables. The part number for a drive add-on kit is 000915; please specify whether you are adding a second or third drive.

### 2.3.1 Removing the Filler Panels

To remove the panel, first remove the cabinet (see section 2.2.2 of this appendix). Then remove the 6-32 screw and associated nut and lockwasher attaching the panel to the front support bracket. Save the hardware for use in attaching the disk drive. Now turn the chassis on its left side (power supply side) and remove the hardware attaching the bottom filler panel angle bracket to the bottom of the chassis. Now slide out the panel.

### 2.3.2 Installing Disk Drives

Before installing a new disk drive, the termination network must be removed and the drive addressed as drive 2 or 3. Consult the installation instructions provided with the drive before proceeding.

Remove the card hold-down by removing the 6-32 screws securing it to the chassis. Remove the ribbon cable assembly connecting the drives to the controller card. Then remove the controller card.

Now take the drive power cable and insert one end through the cutout in the card guide bracket near the power supply PC card. Insert the plug into the next empty socket on the power supply PC card. Insert the new drive into the chassis through the front panel, and attach the power cable to the socket on the drive PC card. Make sure the plug is fully inserted into the socket. Make sure that the disk drive PC card does not become detached.

Now insert the drive all the way into the chassis and line up its mounting holes with the holes in the bottom of the chassis.



Secure the drive with two 6-32 X 3/8" machine screws and lockwashers provided in the kit. Put the chassis back on its feet and secure the top of the drive to the support bracket with the last of the 6-32 hardware.

Now reinstall the floppy disk controller card in the last card connector. Make sure it is firmly seated. Reconnect the disk drives and controller with the new drive cable provided. The striped side of the cable must be on the right side of the controller card connector and on the top end of each drive connector. The card hold-down must now be installed. Place it over the top of the card guide bracket at the center of the chassis, and secure it with 6-32 screws and lockwashers.

### 2.3.3 Removing Drives

To remove a disk drive, first remove the card hold-down, controller card, and drive cable as detailed in section 2.3.2 of this appendix. Now turn the chassis on its left (power supply) side and remove the three 6-32 screws and lockwashers securing the drive to the chassis and retaining bracket. Pull the drive out far enough to remove the power supply plug, then remove it completely.

Reinstall the controller card, drive cable, and card hold-down as in section 2.3.2 of this appendix.

## 2.4 BACKPLANE

### 2.4.1 Card Hold-Down

To protect cards during handling and shipment, card hold-downs are secured along the card guide brackets in the 8813 chassis. (Only one hold-down is provided in the rack-mountable configuration.) These hold-downs must be removed before removing or installing cards in the backplane and replaced after the task is complete. The hold-downs are secured by 6-32 screws and lockwashers.

### 2.4.2 Installing Cards

When installing cards, make sure the power is off before inserting the card, and make sure the card is fully inserted into the connector before reapplying power. To insert the card, slide its ends into the card guides located on the chassis at each end of the 100-pin connectors. Slide the card down until it reaches the connector. Finally, push down firmly on the card until it seats in the slot. The card must not be tilted in the connector, or the computer will be damaged when power is applied.

### 2.4.3 Removing Cards

When removing a card from the backplane, grasp each end of the

card firmly and pull upward, gently rocking the card from side to side. Do not bend the card and do not apply much force. When it comes free from the connector slot, draw it up through the card guides carefully.

#### 2.4.4 Memory Cards

When a memory card is installed into the system, the card's address must be set up properly. The memory in a standard system (with 32K of RAM) ends at 09FFFH. The next memory card should be installed with its address switches or jumpers set for address A000H. Consult the manufacturer's manual for address setup. The next card must always be installed at the end of current memory, or erratic operation or possible damage to the memory card will result. The system will hold up to 56KB of RAM. A typical configuration would be 3 16KB RAM cards at addresses 2000, 6000, and A000, and 1 8KB RAM card at E000.

After you install the memory card, test it with the Confidence Package Memory Test. If you find errors, consult the memory card's manual or get your dealer to repair the card. If the memory is not working properly, neither will the software.

#### 2.4.5 I/O Cards

If an I/O card is installed in the 8813, that card is not all that is needed. Software, called an I/O driver program, is necessary to make the card run with the system. If none is provided, you must write your own. Consult the System Programmer's Guide (available from PolyMorphic Systems; part number 810133) for information on how to install your own software. When installing I/O devices, remember that all I/O ports below 40H are reserved for system use, and you must not place cards at those addresses unless explicitly instructed to do so by PolyMorphic Systems.

PolyMorphic Systems provides instructions for attaching I/O driver software in its I/O card manuals or system user's manuals.

#### 2.5 I/O Connectors

Cutouts are provided on the rear panel for various types of connectors. Most connectors will mount with standard hardware. However, special hardware is needed if connector mounting screws are to be used with "D" connectors. This hardware is available from PolyMorphic Systems (part number 104415). Preassembled cables are available from PolyMorphic Systems for all of its I/O cards.

## APPENDIX G: 8810 HARDWARE

## Table of Contents

## Section 1: General Description

## 1.1 CHASSIS LAYOUT

- 1.1.1 Power supply
- 1.1.2 Disk drive
- 1.1.3 Backplane
- 1.1.4 Minicards
- 1.1.5 Rear panel
- 1.1.6 Switches

## 1.2 SPECIFICATIONS

- 1.2.1 Power supply
- 1.2.2 Disk drive
- 1.2.3 Backplane
- 1.2.4 Memory
- 1.2.5 Central processor

## Section 2: Installing Options

## 2.1 SAFETY

## 2.2 CABINET

- 2.2.1 Removing the cabinet
- 2.2.2 Removing the side panels

## 2.3 BACKPLANE

- 2.3.1 Card hold-downs
- 2.3.2 Installing cards
- 2.3.3 Removing cards
- 2.3.4 Memory cards
- 2.3.5 I/O cards. General Description



## 1.1 CHASSIS LAYOUT

Referring to Figure G-1, note that several components are mounted within the main chassis. Viewed from the front, the five-slot backplane and fan are to the left. One or two optional mini-cards may be mounted on the rear panel just above the powersupply. The right section contains power supply components, mounted in the rear, and a disk drive in front.

### 1.1.1 Power Supply

The power transformers are mounted in the right rear, and in front of them is the disk drive power supply printed circuit card (PC card). To the right of the transformers is the cooling fan. The power supply supplies power to the disk drive.

### 1.1.2 Disk Drive

The mini-floppy drive offers the user the random access storage capability of large disks in a package about the size of most cassette tape units. In addition, it provides superior data integrity and faster throughput. The double-density drive can store 358,600 bytes of information on a on a double-sided five-inch mini-disk. A single-density drive can store 89,000 bytes of information on a single-sided disk. Attached to the top of the drive is a PC card containing the control electronics.

### 1.1.3 Backplane

The backplane contains five edge connectors spaced at 3/4" intervals along the PC card, for the insertion of five S-100 bus compatible cards. In front of the edge connectors are the components for the backplane power supply and real-time clock. AC power enters the backplane through a connector mounted at the front of the card. The rear connector is used for the real-timeclock signal and power and load switch connections.

The 8810 in its minimum configuration uses four card positions, leaving one free for additional memory or an input/output (I/O) card. The bottom position contains the disk controller card. Mounted above it are the CPU card and one 16K RAM card. The top position contains the video terminal interface (VTI) card.

### 1.1.4 Minicards

Space is provided on the rear panel for mounting two optional minicards. An audio cassette interface minicard and a serial printer interface minicard are available. Both of these cards include cables that plug into the CPU card, without using up positions on the S-100 backplane.

### 1.1.5 Rear Panel

All connections to other devices are made through connectors mounted on the rear panel. The right side of the panel as viewed from the front contains the power line connector and fuse. An accessory outlet is provided for powering the video monitor. The power line voltage and frequency are marked on the serial number sticker on the rear panel. The fuse ratings for each voltage are marked on one side of the fuse holder. Use only the type of fuse and power source indicated; other types can severely damage the computer.

Six connector cutouts are provided on the rear panel. In the minimum configuration, the keyboard and video connectors are installed at the positions marked on the panel. The printer interface minicard may be installed in the cutout labeled PRINTER and the cassette interface minicard just above it at AUX 2. The cutout labeled AUX 3 takes a standard 25-pin D connector (Amphenol DB-25P or DB-25S) for use with other I/O cards. The long cutout at AUX 1 takes a 37-pin D connector (DB-37P or S).

#### 1.1.6 Switches

The 8810 has just two switches. The power switch is on the left. The switch is illuminated when power is on. The other switch is a pushbutton marked Load. When pressed, it causes the system software to be loaded off of the drive.

### 1.2 SPECIFICATIONS

#### 1.2.1 Power Supply

##### AC Line

Voltage:	110-1255 or 220-250 VAC
Frequency:	50 or 60 Hz

##### DC Output to Backplane

Pins 1 and 51:	+7.5 to 10 VDC @ 6A max.
Pin 2:	+15 to 20 VDC @ 0.75A max.
Pin 52:	-15 to 20 VDC @ 0.25A max.

## 1.2.2 Disk Drive

	Standard	DSDD
Data storage per disk:	89,600	358,600 bytes
bit transfer rate:	125,000	250,000 bits/sec.
Sector transfer rate:	25	90 sectors/ sec.
Latency (average):	100	100 ave. mSec.
Track step:	40	5 ave. mSec.
Seek ave.:	700	87 ave. mSec.
Number of tracks:	35	35
Sides	1	2
Sectors per track:	10	10 (20 simulated)
Bytes per sector:	256	512 (256 simulated)
Maximum hard error rate:	1 in $10^{11}$ bits read	
Mean time between errors:	8,000 hours	

The DSDD (double sided, double density) operates at a faster track step time and has out-of-sequence sector read capabilities due to an intelligent controller which also buffers tracks. The average latency time is thus effectively lower than shown, because multiple sector accesses may begin reading at any sector. Also, some reads require no disk access.

## 1.2.3 Backplane

Bus type:	S-100 (except no DMA)
Number of card slots:	5
Card spacing:	3/4" center-to-center
Maximum card size:	

length:	10"
height (measured at edge to 0.5" from edge):	5.3"
height (center)	5.8"

## 1.2.4 Memory

Maximum addressable User RAM:	57,344 bytes (56KB)
Access time:	500 nS
Error rate:	1 in $10^{12}$ bits read

## 1.2.5 Central Processor

Type:	8080
Number of user-accessible registers:	7 (8-bit)
Data word size:	8 or 16 bits
Register-to-register add time:	2,000 nS

## Section 2. INSTALLING OPTIONS

### 2.1 SAFETY

Whenever working on the 8810 chassis, perform these three steps before opening the chassis:

1. Remove disk from drive.
2. Turn off power switch.
3. Remove line cord from rear panel socket.

Do not insert or remove any cards or minicards with power on or line cord attached. If this precaution is not observed, parts of the chassis circuitry and cards will probably be destroyed. Remember, there are dangerous voltages in the chassis until the line cord is unplugged, even when the power key is turned off.

### 2.2 CABINET

The 8810 is enclosed in an attractive cabinet with walnut side panels. To avoid damage, do not set cups or sharp objects on it. The surface of the wood is waxed, not varnished; do not apply varnish. Occasionally rub the wood surfaces with a small amount of high-quality furniture wax and a soft, clean cloth.

Leave a 6" air space at the rear of the cabinet to allow proper air flow. If the flow of air is blocked, the temperature inside the main unit may rise enough to impair operation, shorten the life of the computer, or cause components to fail.

#### 2.2.1 Removing the Cover

The top cover is secured by two 6-32 screws and lockwashers located at the top left and right corners of the rear panel. The front of the panel slides into a receptacle underneath the top of the brushed aluminum front panel. To remove the cover, remove the two screws and slide it toward the back of the 8810.

#### 2.2.2 Removing the Side Panels.

Each of the side panels is secured by the two black anodized screws and decorative washers that protrude slightly from the wood panels. The left panel must be removed to access the five-slot backplane. The right panel need never be removed.

### 2.3 BACKPLANE

#### 2.3.1 Card Hold-Downs

To protect cards during handling and shipment, card hold-downs are secured to the top of the card guide brackets on the 8810 backplane. These must be removed before removing or installing cards in the backplane and replaced afterward. Each hold-down is secured by a 6-32 screw and lockwasher.



### 2.3.2 Installing Cards

When installing cards, make sure the power is off before inserting the card, and make sure the card is fully inserted into the connector before re-applying power. To insert the card, slide its ends into the card guides located on the brackets at each end of the 100-pin connectors. Slide the card into the guides until it reaches the connector, then carefully guide the card contacts into the connector. Support the backplane from behind by placing one hand behind the backplane PC card, then press firmly on the card and seat it in the slot.

**WARNING!** Failure to support the backplane will cause excessive stress on the backplane printed circuit card, which may result in damage to the printed traces.

The card must not be skewed in the connector, or the computer will be damaged when power is applied.

### 2.3.3 Removing Cards

When removing a card from the backplane, grasp each end of the card firmly and pull upward, gently rocking the card from side to side. Do not bend the card and do not apply much force. When it comes free from the connector, draw it up through the card guides carefully.

### 2.3.4 Memory Cards

When a memory card is installed into the system, its address must be set properly. The memory in a standard system (with 32K of RAM) ends at 9FFFH. The next memory card should be installed with its address switches or jumpers set for address A000H. Consult the manufacturer's manual for address setup. The next card must always be installed at the end of current memory, or erratic operation or damage to the memory card will result. The system will hold up to 56KB of RAM. A typical configuration would be 3 16KB RAM cards at addresses 2000, 6000, and A000, and 1 8KB RAM card at E000.

After installing a new memory card, use the Confidence Disk to perform the memory test on the new memory card. If you find errors, consult the memory card's manual or get your dealer to repair the card. If the memory is not working properly, neither will the software.

### 2.3.5 I/O Cards

If an I/O card is installed in the 8810, that card is not all that is needed. Software, called an I/O driver program, is necessary to make the card run with the system. If none is provided, you must write your own. Consult the System Programmer's Guide (available from PolyMorphic Systems; part

number 810133) for information on how to install your own software. When installing I/O devices, remember that all I/O ports below 40H are reserved for system use, and you must not place cards at those addresses unless told to do so by PolyMorphic Systems.

I/O cards from PolyMorphic Systems include instructions for attaching the I/O driver software.

## Appendix H

## PRINTER DRIVER INTERNAL STRUCTURE

This section describes the machine language level structure of the printer driver. It is intended to allow the assembly language programmer to use the machine level features of the printer driver and to write special assembly language drivers for special hardware interfaces.

The printer driver runs in system RAM from 2F00H-31FFH. There are two areas which contain machine code, and two parameter passing areas. The Wormhole driver area is the permanent part of the printer driver. The program in the device driver area is dependent on the hardware, and may be changed by the user. The Wormhole driver is installed by Prnt.OV when the Exec "Printer" command is invoked. The system is provided with only one driver, Sio.PS. All drivers are stored on the system disk with .PS extensions, so they can easily be copied from system to system.

```

2F00  -----
      Wormhole driver
3000  -----
      Device driver
31E0  -----
      Optional communication area
31EC  -----
      Wormhole communication area
3200  -----

```

## WORMHOLE DRIVERS

The wormhole drivers are the heart of the program which supports the functions provided by WH5 through WH7.

WH5 prints characters from the A register, handles the left EDGE, TAB & FF emulation, and keeps track of cpos & lpos (see description below), making many printers appear identical to the higher levels. It recognizes TAB, LF, VTAB, FF, and CR. TAB moves the print head horizontally to the next position whose location is divisible by 8. The left margin offset value is not significant for TABs. CR moves the print head directly to the left, with no vertical motion. LF moves the print head straight down, with no horizontal motion. VTAB moves to the next page, as does FF, but will not move from the top of a page; thus two successive VTABs are identical to one.

WH5 communicates with higher level programs such as Wordmaster through "data requests". Characters passed in the accumulator are printed if the 80H bit is clear, or else are interpreted as data requests according to the following table.

code	name	functional description
----	----	-----
80H	lpos	line position (0..lpp-1) counts up
81H	cpos	char position (0..cpl-1) counts up
82H	lpp	lines per page (1..256) 0 = 256
83H	cpl	char per line (0..255)

The data returned by a data request to WH5 is supplied in the A register. NOTE: The contents of all of the 8080 registers may be changed by the WH5 driver.

WH6 simply waits for and obtains a character from the printer if it has a keyboard. If the device doesn't have a keyboard WH6 returns a NULL (ASCII 0). All registers except the Accumulator are preserved by WH6.

WH7 is the normal output for characters to be printed. It does pagination (top and bottom margins) and generation of LF after CR automatically. LF still works as usual, but normal text files will print properly when sent to WH7, because they have only CR to terminate lines. All registers are preserved by WH7.

#### DEVICE DRIVER SPECIFICATIONS

The device driver is a user written assembly language program which handles the low level communication between the printer and the computer. It should be assembled to run starting at location 3000H with the first location being its entry point. A function code is passed in the B register and characters are passed to and from the device driver in the accumulator.

The function codes are defined as follows:

01H	Ouput a character in Accumulator to the device
02H	Input a character from the device into the Accumulator
03H	Initialize the device
04H	Disconnect the device

All registers may be destroyed by your program, with the obvious exception of the accumulator in function 2. An example of a simple device driver appears at the end of this appendix.

#### COMUNICATION AREAS

The following labels appear in the PrntRefs.SY symbol table file on the disk provided. They may all be REFed in the following way in MACRO-88.

REFS	PrntRefs
REF	MAGIC
MAGIC	

MAGIC is a macro which defines the addresses of all the labels

in the following tables.

#### Optional communication area

---

The optional communication area and part of the wormhole communication area are initialized by the "Printer printername" Exec command with the values defined by the user for the "printername."

The optional communication area may be used for program code if the standard dialog is NOT used, as defined when installing the printer driver using the CUSTOM command.

speed	contains the speed of the printer		
	11H = 50	12H = 75	13H = 110
	14H = 134.5	15H = 150	16H = 300
	17H = 600	18H = 900	19H = 1200
	1AH = 1800	1BH = 2400	1CH = 3600
	1DH = 4800	1EH = 7200	1FH = 9600

padchr	ASCII code for the padding character
crpad	The number of pads after a CR (0..255)
lfpad	The number of pads after a LF (0..255)
tabpad	The number of pads after a TAB (0..255)
bspad	The number of pads after a BS (0..255)
blim	(Device buffer limit)-1 0=not blocking
stxflg	should we send a start character
stxchr	ASCII code for the start of text character
etxchr	ASCII code for the end of text character
ackchr	ASCII code for the acknowledge character

#### Wormhole communication area

---

The wormhole communication area is shared between the wormhole driver and the printer driver, for margination, and for simulation of TABs, FFs, and VTABS for printers which do understand them.

Logsp	A six byte space used by the LOG command
lpos	current line position (0..lpp-1) counts up
cpos	current char position (0..cpl-1) counts up

lpp	lines per page (1..256) 0=256
cpl	char per line (0..255)
top	first printable line (top margin)
bottom	last printable line (bottom margin)
edge	offset for left edge (0..255)

(note: 0 <= top <= bottom <= lpp-1)

tabflg	understand TABs (0=no, nz=yes)
--------	--------------------------------

ffflg understand FFs (0=no, nz=yes)  
devtyp Similar to a Diablo (0=no, nz=yes)

```

;
; **** EXAMPLE DEVICE DRIVER ****
;
      ORG      3000H      ;Where we live
      IDNT    $,$        ;Load, start address
;
; hardware definitions.
;
STATUS EQU      80H      ;Status port
PrtrDy EQU      80H      ;Bit says printer is ready
;
PRTCTL EQU      STATUS   ;Control port
IntEn  EQU      80H      ;Enables keyboard interrupt
;
PRTDAT EQU      STATUS+1 ;Data port
Strobe EQU      80H      ;Active on falling edge
;
;*****
;
Start  MOV      C,A      ; Save in C for future use
      DCR      B
      JZ       Putc      ;1> output a char
      DCR      B
      JZ       Getc      ;2> input a char
      DCR      B
      JZ       Init      ;3> hookup device
      DCR      B
      JZ       Kill      ;4> disconnect device
      RET
;
; Output a character to the device
;
Putc   IN       STATUS
      ANI      PrtrDy
      JNZ      Putc      ;Wait till device is ready
      MOV      A,C
      ORI      Strobe    ;Set MSB (strobe)
      OUT      PRTDAT
      ANI      NOT Strobe ;Clear MSB
      OUT      PRTDAT    ;And output again
      RET
;
; Input a character from the device
;
Getc   LXI      H,kbuf    ;Pointer to flag
Getc1  MOV      A,M
      ORA      A
      JNZ      Getc1     ;Wait for a character
      DCR      M         ;Clear flag
      INX     H
      MOV      A,M       ;Grab char in A
      RET               ; and split
;
; Initialize the device

```

```
;
Init      LXI      H,Isr          ;Just setup ISR
          SHLD     SRA4
          MVI      A,IntEn       ;Let keyboard int in
          OUT      PRTCTL
          RET

;
; Shut off the device
;
Kill      MVI      A,NOT IntEn
          OUT      PRTCTL       ;Stifle interrupts
          LXI      H,Ioret
          SHLD     SRA4         ;Disconnect ISR
          RET

;
; Our simple interrupt service routine.
;
Isr       IN       PRTDAT       ;Get the character
          LXI      H,KBUF
          MVI      M,0         ;Zero the flag.
          INX      H
          MOV      M,A         ;Put char in buffer

Ioret    ;
          POP      H           ;Standard interrupt return
          POP      D
          POP      B
          POP      PSW
          EI
          RET

;
kbuf     DS        2
;
          END
```



## Appendix-I

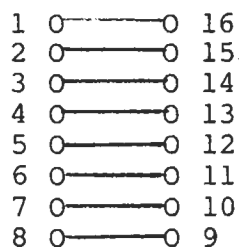
## FILE TRANSFER PROGRAM

## 1.0 General Description

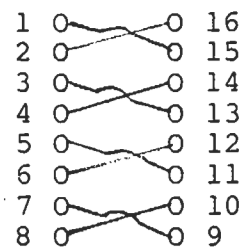
There is a file on your system disk called FTP. It is a file transfer program. It will allow you to transfer files on your disk to another system which is also running FTP. This transfer is accomplished over an RS232 link. This program enables you to transform files created on a single density, single-sided System 88 into files which can be read on a Double Density 8813.

The program's secondary function is to cause the system to act like a terminal. It can therefore be used as a remote terminal for another computer, either connected directly to the RS232 port on the other computer, or over a modem link. The program will also allow you to send files over a modem to another System 88 running FTP.

To use FTP you must first be sure that you have the right header in the printer interface on your computer. If you wish to use your computer as a terminal (with or without a modem) the header should be wired straight across, ie. pin 1 to pin 16, pin 2 to pin 15 etc. If you wish to connect two System 88s back to back, to transfer files, one system should have a header wired as just described. The other system should have it's header cross wired, ie. pin 1 to pin 15, pin 2 to pin 16, pin 3 to pin 13, pin 4 to pin 14 etc. See diagram below.



Straight Across



Crossed

## Header Wiring

When you use the file transfer program it will take control of the USART (Universal Synchronous Asynchronous Receiver Transmitter) chip and its associated circuitry. This means that the printer driver on the System 88 will no longer be connected to this circuitry. Consequently when you are finished with this program the printer driver should be set up again.

## 1.1 USING FTP TO TRANSFER FILES BETWEEN ADJACENT SYSTEMS.

To use FTP to transfer programs between two systems which are connected directly together (no modem link) first set up the headers on the printer interfaces as described above. Power up the systems and invoke the program on both systems.

FTP will first ask you "Baud? (110,300,9600)". Since you want a speedy transfer of data you should answer 9600 to the questions on both systems. Note: Both systems must be running at the same baud rate in order for the program to work.

FTP will then put a say "(File transfer terminal/04: 6/6/79)". This version number and date will let us know which version of the program you have and should be referred to in any correspondence regarding the program.

At this point the two systems should be communicating with each other. Anything typed on the keyboard of one one computer should appear on the screens of both computers.

To initiate a file transfer, type control S on the computer that already has the file. It will then ask you "Send File:" to which you should reply with the name of the file you want sent.

A message will then appear on the other computer's screen which reads "File: (filename) is being sent." The receiving computer will then ask "Receive file name (ESC (ret) for abort):". You should type into the receiving computer the filename on that system where you want the data stored. If you want to abort the transfer process at this point type the ESC key followed by a carriage return. To abort the transfer at any other point in the process type the ESC key on either computer.

When the process is aborted at the receiving end the message "Load Aborted" will appear on the screen of the receiving end and the message "Transmission terminated at receiving end." will appear on the screen of the transmitting computer. If the process was aborted by the transmitting computer the message "Transmission terminated." will appear on the screen of the transmitting computer, and the message "Transmission Aborted" will appear on the screen at the receiving end. In either case both computers will be dumped back into a terminal mode.

The load process will also be aborted automatically if the receiver gets a record number that is out of order. If this occurs the same messages will appear on both screens as if an ESC had been typed on the keyboard of the receiving computer.

As soon as you have entered the destination filename the data transfer will begin. A display of the record numbers being sent will appear on the left side of both screens. If an error is made in the transfer of any of the records the program will

send the record over again. This will be evidenced by a repeated record number on the left side of the screen.

When the entire file has been transferred the message "File mailed." will appear on the screen of the transmitting computer and the message "File Loaded." will appear on the screen of the receiving computer. At this point the program will return to its terminal mode. To exit the program type control Y. Note: When a control Y has been typed to exit from this program, execution CANNOT be continued by using the CONTINUE or REENTER commands.

### 1.2 Using FTP as a terminal.

If you wish to use your System 88 as a terminal you can do so using FTP. The header must first be set up as described in section 1.0. The system is then powered up and FTE is invoked.

FTE will ask you for the baud rate you wish to use. Three baud rates are provided. The 9600 baud rate is used when connecting the System 88 directly to the back of another computer. The 110 and 300 baud rates are used when connecting the computer through a modem to another computer. Two baud rates are provided to accommodate different types of modems.

There is only one restriction in using FTE as a terminal. That is that a control S cannot be typed or received. If one is the program will revert to file transfer mode (See section 1.1).

FTP can also be used to transfer a file over a modem by setting up the baud rate for a modem, selecting the right header, and using the procedures outlined in section 1.1.



## Appendix J

## MIRROR AND DUP

MIRROR and DUP are two programs on your System Disk which are intended for use on single drive systems. They allow you to copy files from one disk to the other, and to IMAGE whole disks on to other disks.

You must ENABLE your system before invoking MIRROR. If you want one disk to be the mirror image of another, type

```
$$MIRROR
```

You will then see the following message:

```
Normally, you should wait for drive light to go off
before removing disks, but in this program, you may swap
them immediately after I tell you.
```

```
Insert copy FROM disk.
Hit any key to continue. . . .
```

Follow these instructions. Note that the disk you are making a mirror image of does not have to be a system disk. You are being given this opportunity to remove your system disk and insert any disk that you want to "copy FROM."

Once you have inserted the "copy FROM" disk and "hit any key to continue," you will be asked to:

```
Insert copy TO disk.
Hit any key to continue . . . .
```

When you have inserted the "copy FROM" disk, you will see the following message:

```
THIS DISK WILL BE UNUSEABLE UNTIL THE PROCESS IS COMPLETED
```

The above message is warning you not to interrupt the IMAGE process if you plan to be able to use your "copy TO" disk. If the process is interrupted, the "copy TO" disk directory will not be properly constructed and the disk will not be useable.

You will be asked to repeat this process of exchanging disks until all of the data on the "copy FROM" disk has been recorded onto the "copy TO" disk.

When all of the information has been copied from the "copy FROM" to the "copy TO" disk, the screen will display the following message and return to Exec,

## OPERATION COMPLETED

If the disk in the drive is not a system disk, you will be asked to insert a system disk and hit any key to continue. Once you have inserted the system disk, the system will return to Exec.

If you want to copy one file from one disk to another disk, type the following:

\$DUP Pathname

You will then see the following message on your screen:

Normally, you should wait for drive light to go off before removing disks, but in this program, you may swap them immediately after I tell you.

Insert master disk. Hit any key to continue. . . .

Follow these instructions. Note that the disk which has the file you wish to copy does not have to be a System Disk as you are being given the opportunity to replace your System Disk with the "master" disk. Once you have inserted the master disk and hit return you will be asked to insert the "slave" disk. You will be asked to alternate the slave and the master disk until the copying process is finished. Then you will be given a chance to insert your System Disk. When the copying process is finished and there is a System Disk in the drive the system will return to Exec.

## GLOSSARY OF MANUAL TERMS

## ASCII

Because the machine deals only with binary numbers, all characters must be represented within the machine by a numerical code. The American Standard Code for Information Interchange (ASCII) is a numerical code representing letters, numbers and symbols-- all of the characters that can be typed in from your keyboard, including control characters. When you type a capital letter T from your keyboard, for example, the machine receives that symbol as the number 54H in hexadecimal or the number 1010100 in binary--the ASCII code for a capital T. (See Appendix BThe ASCII Character Set for the ASCII values for your system's complete character set.)

## Assembler

The "Assembler" is a machine language program that translates assembly language programs (see Assembly Language) into machine language that your machine can use as instructions. For information on using the Assembler, see Section 12, The Assembler, and the System Library volume The MACRO-88 Assembler.

## Assembly Language

At its most basic level of operation, a computer understands only numbers, in fact only 0 and 1. Every instruction that it follows is represented by a particular number made up of 0s and 1s. When you type in a BASIC command, that command is interpreted by BASIC, and BASIC then performs the actions called for by the command. However, BASIC is itself a machine language program--it is made up of numbers (machine language) which the machine understands as instructions. To write a machine language program, we use a computer language called "assembly language." This language consists of "mnemonics" (word substitutes) that represent the numerical instructions of machine language. This makes it much easier for humans to communicate with the machine. The assembly language program must still be translated into the machine language that your machine can recognize. This is the task of the Assembler (see Assembler).

## BASIC

BASIC is a widely used computer language. It is a much higher level language than assembly language--it resembles normal English much more closely than assembly language does. It is generally considered to be simple to use and easy to understand. See the System Library volume BASIC: A Manual for details on using BASIC.

## Binary

A binary number is a number written in base 2. We normally use decimal numbers, or numbers in base 10. ("Base 10" means that each place in a number represents a power of ten, and thus ten symbols are used to form the numbers in that base-- the symbols 0 through 9. "Base 2" means that places in a number represent powers of two, so two symbols, 0 and 1, are used to form numbers in binary notation.) Each digit in the decimal number 256 represents that digit multiplied by a power of ten (what power of ten depends upon the digit's place in the number). What the decimal number 256 really means is:

$$2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$$

Or,  $(2 \times 100) + (5 \times 10) + (6 \times 1)$ . Just so, the digits in a binary number represent powers of two. The decimal number 10 is represented as the binary number 1010. What this really means is:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

Or,  $(1 \times 8) + (0 \times 4) + (1 \times 2) + (0 \times 1)$ . At its most basic level, a computer understands only binary numbers.

## Bit

All information within your machine is represented as binary numbers (1s and 0s). Each digit of these numbers is called a "bit" (for binary digit) and is the smallest unit of information. The following number consists of 8 bits and represents the decimal number 84 or the hexadecimal number (base 16) 54H: 01010100.

## Byte

Binary information is grouped within your machine in groups of eight bits (eight 1s and 0s). Each of these groups is called a "byte." This is a convenient grouping, because the machine language instructions carried out by your machine are performed



on groups of eight or 16 bits (one or two bytes). One byte of eight bits is also the amount of bits needed to represent one character in ASCII code.

### Computer Language

A language is a set of symbols which can be combined according to certain rules to communicate information. A computer language is a set of symbols with which we communicate with the machine. The language closest to the functioning of the machine is machine language, which consists of binary numbers (translated for our use into hexadecimal numbers) representing machine instructions. A language closer to human language is called assembly language. It consists of word symbols that represent machine language. Even higher-level languages exist, like BASIC or FORTRAN.

### Control Characters

Control characters are characters entered from your keyboard that signal the system to perform operations on the monitor screen and/or printer, such as clear the screen, move the cursor, delete characters, etc. See Appendix B--ASCII Character Set for a list of control characters and their uses.

### Data

As used in this manual, data is simply another word for information. Numbers, characters, text, etc. are all considered data. When we speak of one byte of data, we are talking about a unit of information that can be represented in one byte of machine memory (e.g., one character). When speaking of BASIC programs, we sometimes make the distinction between information generated and used by programs (often referred to as "data") and the programs themselves. On the machine level, however, since programs are represented as numbers, as are the values they manipulate, no true distinction exists between programs, text, program data, etc.

### Decimal

A decimal number is a number written in base 10. (See Binary for an explanation of base 10 and base 2.)

### Editor

The Editor is a machine language program that helps you to create and change text files. You can create and edit (correct and change) assembly language programs, BASIC programs, and non-program text files. This manual was created using the

Editor. See Section 11, The Editor, for more information.

### Execute

To execute a program means to "run" that program. A program consists of a series of instructions followed in proper sequence by the machine. As the instructions are performed, we say that the program is executing. (American computers execute instructions; British computers obey orders.)

### Expression

An expression is a group of mathematical terms and variables connected by arithmetic operator symbols. An expression will be evaluated by the machine-- that is, it will be reduced to its simplest expression. An expression may consist of only one term or many terms.

Examples of expressions:

$3+25*(\text{SIN}(45/A))$

78

$65+45.678-(234.1/56)$

### File

Any data that you want to save can comprise a disk file. A file may thus contain text, machine language, BASIC programs, numerical data, etc. You can think of a file as a manila folder that you use to hold any type of information. After putting it in the file cabinet (i.e., the disk), you can make a copy of the contents of the file folder. You work on the "copy" that is in the temporary memory inside the main unit, not on the "original" that is in the "filing cabinet" or disk. Eventually, however, you may replace the "original" on the disk with the up-dated "copy."

### Form Feed

Typing the control character Control-L results in a form feed-- the screen is cleared and the cursor appears in the upper left hand corner of the screen after the next carriage return. The Editor is the only exception: when editing, you insert a form feed into your text by typing Control-L (the Greek letter appears to show you where the form feed will occur). When your text is printed on a printer, the form feed will force the printer to go on to the beginning of the next sheet of paper.

### Hexadecimal

Many of the numbers mentioned in this manual are written with an H after them (e.g., 2000H). The H indicates that the number is in hexadecimal (base 16) form. Why do we use hexadecimal numbers? Data are grouped within your machine in groups of eight bits (1s and 0s), called bytes. Each memory location holds exactly one byte of data. (Bytes are convenient groupings because one byte represents one character in ASCII code and because most machine operations are performed on one byte at a time.) Hexadecimal numbers may be very easily translated into bytes of binary numbers (the actual numbers used by the machine). Any hexadecimal digit can be expressed in no more than four bits-- a half of a byte. Therefore two hexadecimal numbers exactly represents the contents of one memory location; that is, one byte of data.

Because hexadecimal is base-16 rather than base-10 (i.e., ordinary decimal), it uses sixteen symbols instead of ten: the ten digits 0-9 and the first six letters A-F. The largest number expressible in two digits in hexadecimal is FF, written FFH and equal to 255 in decimal.

## Input

To input data is to transfer it to a device (memory, disk file, etc.) from another device (keyboard, memory, disk file, etc.). For instance, when you type a character, you are "inputting" that data into the memory inside of the machine. In addition to being a verb, input is also a noun identifying the data transferred-- the character that you inputted to memory is itself known as "input."

## Invocation

Invoking a file means bringing a copy of a program file into memory and executing it. Invocation thus implies that the file is a runnable file (a series of program instructions that can be carried out by the machine) rather than a data file (a file of information to be manipulated in a program), for example.

## I/O

The common term for the combined operations of input and output is "I/O" (Input/Output). Disk I/O means data transfer both into and out of a file (read and write operations).

## Language

See Computer Language.

## Line

A line of text is all of the characters between two carriage returns. In the case of this paragraph, for example, the first line of the paragraph consists of the characters A line of... ..two carriage. Usually a line appears on the screen as just that-- one separate line. But a line may be longer than the width of the monitor screen. If it is, it will "wrap around" and continue on the next line position, even though you haven't put in a carriage return. Be careful that everything that you mean to be a line is delimited with carriage returns.

### Load Address

Machine language program files are saved along with certain information the system uses when that program is again brought into memory and executed. The load address tells the system at what memory address to begin writing the program when it is placed in memory. A machine language program could thus be saved from memory location 3200H but written back into memory at location 4000H (memory addresses are in hexadecimal form, and so are followed by an "H").

### Machine Language

At its most basic level, your machine operates in a very simple way. The machine actually understands only about eighty instructions which in machine language are represented by binary numbers (translated for our use into hexadecimal numbers). These instructions are very simple; they can do no more than cause numerical data to be moved from one place in memory to another and cause simple arithmetic operations to be performed on that data. In order to write programs in machine language, programmers almost always actually write in a language that uses mnemonic word-substitutes for the machine language (see Assembly Language) and which is then translated into machine language by the Assembler (see Assembler).

### Memory

The temporary storage area for the data your machine is actually using at any given moment is called "memory." The "amount" of memory that your machine has is equal to the number of memory locations that you can use. Each memory location stores one byte of data. When you invoke a file, a copy of that file is placed into memory. All data, whether it is a program, text, a copy of a data file, etc., is placed into memory in order for it to be accessible to the machine.

### Memory Address

Every memory location in your machine is reached via its

address (a two-byte hexadecimal number). It is important to keep clear the difference between the contents of a memory location and the location's address. Memory location 3000H, for example, will always hold one byte of data (eight bits). That data can be anything you desire that can be represented in numerical form, and can be changed at any time. But that particular location will always have the address 3000H, no matter what contents that location may hold.

### Mode

Mode means "way," and the system has many different ways of functioning. For instance, we can talk about "keyboard modes" (FULL, fold, flip) which cause characters typed in from the keyboard to be translated in different ways. We can talk about "system modes" (enabled and disabled) which reflect different states of system operation. A mode implies a temporary or changeable method of operation-- if you are in enabled mode, for example, you may in the future be in disabled mode.

### Output

The process of transferring data out of a device (a keyboard or a disk, for instance) to another is called output. Writing data out from memory is outputting that data. Besides being the term for the process of data transfer as viewed from the data's point of origin, output is also what we call the data itself that is being transferred. If a BASIC program asks for a number from the user of that program (asks for input), and then writes that data out to a file, that number is output to a file and is itself called output.

### Path Name

When you want to specify a file or directory to the system, you must type its path name. The path name is made up of component names. The components which can be in a path name are: drive names, sub-directory names, file names and extensions. See Section 4 for a complete explanation of how to enter path names.

### Printer Driver

When you want to output data to a printer so as to get the data printed, you must have software that processes the output and enables the printer to operate. The System 88 uses an adaptable printer driver to communicate with a variety of printers.

### Program

A series of instructions for the system to follow in order is

called a program. The instructions may be expressed in the terms of the computer language called BASIC (a BASIC program) or in machine language (a machine language program) or in some other computer language. You might even consider a command file (a series of system commands) a program.

### Prompt

A prompt is a symbol used by the system or various files on the system (e.g., BASIC or the Assembler) as an indication that the system is waiting for a new user input. The prompt symbol for the system level (communicating with Exec) is a single or double dollar sign symbol \$ or \$\$, depending upon whether the machine is operating in disabled or enabled mode. The BASIC prompt symbol is a single or double carat symbol, > or >>, depending upon whether execution of the BASIC program currently in memory cannot or can be continued by BASIC.

### RAM

The most common type of machine memory is RAM-- Random-Access Memory. Unlike Read-Only Memory (see ROM), Random-Access Memory may be both read from and written into. Every memory location has an "address" (see Memory Address) by which you select that particular location; this means that you can access RAM locations in any order (not just sequentially). This is why this type of memory is called Random-Access Memory.

### Register

Certain areas of machine memory, called registers, are set aside as special data handling areas. When you write a machine language program, the great majority of the instructions of your program are concerned with moving data between these registers. All arithmetic operations performed by the system are done on the data in these registers.

### ROM

A very useful type of memory is called Read-Only Memory (ROM). A ROM contains data and programs the system needs to function. This information, unlike data in ordinary machine memory (see RAM), will always remain in the ROM whether the machine is turned on or off. (Usually data in memory disappears when your machine is turned off.) ROMs can be read from but not written into-- their contents are never altered or erased. The ROMs in the System 88 contain the 4.0 Monitor and the core or "Root" of the disk operating system.

## Run

To run a program is to perform the program's series of sequential instructions. (See Execute.)

## Save

When you write a machine language or BASIC program, you want to store a copy of that program so that you can run later. Often you want to store a copy of data or text for future reference. The process of storing information (whether programs, text, etc.) is called "saving" data. The processes for saving these various forms of data differ depending upon the type of data.

## Save Address

When you save a machine language program, you must tell the system at what memory address to begin saving that program (that is, the memory address at which your program begins). This address is a hexadecimal number.

## Sector

The diskette used by your machine is divided into ten "pie-slice" wedges. In addition, beginning at the central cutout circle, the diskette is divided into 35 concentric circles, called "tracks." The area on one wedge and along one track is called a sector. There are 350 sectors (35 tracks multiplied by ten wedges) on a disk. The disk directory shows the amount of information stored on the disk and the length of the files in sectors.

## Start Address

When you save a machine language program, you must include information used by the system when that program is later retrieved and executed. The start address tells the system what memory location will hold the beginning instruction of your program.

## String

BASIC distinguishes between two types of data-- string and numerical. Numerical data (represented by numerical variables) may have arithmetic operations performed on it and be assigned as values to numerical variables. Strings (represented by string variables) are identified as such by being enclosed with-in quotation marks. Strings of characters within quotationmarks are treated as a unit by the program, and are

not processed except to be reproduced literally by the program. Data placed in a file as strings must be read from that file by using string variables.

## TAB

Use of the TAB key on your keyboard (and the Control-I command) causes the cursor (the white rectangle that indicates your position on the screen) to move to the right eight spaces. If you are creating text, the TAB key also inserts a tab character into the text at that point. The tab character is not visible on the screen, but it is in the file and will cause the printer to execute a tab when the file is output to a printer. The BASIC TAB command causes the PRINT statement to "tab" over an arbitrary amount (depending upon the value specified in the TAB command) before printing data.

## Text

Machine language programs are collections of numbers. They are understood directly by the computer as instructions. Other types of data are given in encoded form (see ASCII), which then must be translated by the system. Characters given in ASCII code are called text. The computer does not perform arithmetic operations directly on numbers that are ASCII characters. BASIC will perform the translation between ASCII numbers and machine language numbers for you when you ask for arithmetic operations. The actual ASCII characters, however, form a type of data that we call text. Text files contain characters that are ASCII-encoded data and not runnable machine language. Technically, therefore, all files except for machine language files are text files, since they are stored in ASCII form. In general, however, when we speak of text files we are speaking not of BASIC programs or BASIC data files (even though they are stored in ASCII form), but of files consisting of text material such as memos, articles, research papers, etc.

## Type-ahead

Your machine is occasionally preoccupied and unable to attend to keyboard input (for example, when running a BASIC program). The characters that you type during these times are stored by the machine and displayed and processed later, when the machine is again free. This capability is called type-ahead. You can enter up to 64 characters to the type-ahead processor.

## Variable

A variable is a symbol used to represent data. In BASIC, a variable may be either a numerical or string variable.



Numerical values or strings may be assigned to the proper type of variable. When we say `A=10`, we mean that wherever the variable `A` occurs, the value of 10 will be substituted for that variable. When we say `A$="This is a string"`, we mean that the phrase assigned to `A$` will be substituted for the string variable `A$`. (For example, when we say `PRINT A$` in a program line, the string "This is a string" will be printed--without its surrounding quotation marks.) A BASIC numerical variable is an upper case letter optionally followed by one numerical digit (e.g., `A1`, `X`, `B3`). A BASIC string variable is any legal numerical variable followed by a dollar sign (e.g., `A1$`, `X$`, `B3$`).

#### Warm-start

The start (or "cold-start") address of a machine language program refers to the memory location containing the program's starting instruction. Program execution begins with this instruction if the program is "starting cold"-- if it has not been running yet. Often this cold-start instruction is at the beginning of a block of code that performs initializations meant to be performed only the first time through the entire program. It may be that instructions in the program cause execution to start over again "from the beginning" many times, but without initialization being repeated. Another start address must be supplied just after the initialization block. This address is the warm-start address.

#### Write-protect

Ordinarily, you can write data to and read it from a disk. Occasionally, you may want to protect the integrity of the data on your disk by prohibiting anyone from writing new data onto it (and thus erasing your data). You can do this by "write-protecting" your disk: put a write-protect tab (supplied with a box of blank disks) over the write-enable notch (see Section 2) The write-protect procedure for the 8" diskette is exactly the opposite of that for the 5" diskette. An 8" diskette is write-protected when the tab over the write-protect notch has been removed.



## INDEX

Words written all upper-case (e.g. DELETE) are usually the names of commands.

- Active files, 39
- Angle brackets, 30, 33
- Angle brackets in path names, 32, 35
- Appending, 102
- Applications programs, 93
- Arrow keys, 100
- ASCII code, 83, 135, 136 (chart 137) 139, 191
- Assembler, 5, 111, 131, 133, 191
- Assembler options, 112
- Assembly language, 22, 84, 111, 191
- Asterisk, 24, 52, 58
- Auto-execute mode, 79, 80, 93
- Auto-repeat, 17
- Backplane, 161, 162, 165, 167, 173, 175, 177
- Base, 47
- BASIC, 5, 10, 77, 111
- BASIC commands .
  - BYE, 78, 80, 81
  - EXEC, 78, 80
  - LOAD, 78
  - SAVE, 81
  - SCR, 81
- BASIC command files, 89, 90
- BASIC programs, 10, 22, 77, 94
- BASIC prompt, 10, 12, 79
- Binary, 21, 83, 188
- Bit, 83, 188
- BYE, 78, 80, 81
- Byte, 15, 83, 188
- Cabinet, 165, 177
- CAPS LOCK key, 9, 17
- Carriage return, 10
- Central Processor, 167, 176
- Characters, 18-19
- Chassis, 161, 173
- CHECKSUM, 51
- Combining files, 105
- Complete path names, 32, 33
- Component names, 32
- Cmdf abort, 46, 87, 134
- Command files, 13, 22, 46, 87, 93
- Command Syntax, 69, 70, 71
- Comments, 13
- Computer language, 187, 189
- CONTINUE, 65, 81
- Control characters, 18, 23, 108, 110, 193
- Control-A, 102-3, 108, 136
- Control-B, 102, 108
- Control-C, 103, 108
- Control-D, 108

Control-E, 102, 109  
Control-F, 103, 109  
Control-I, 18, 109, 135  
Control-K, 18  
Control-L, 18, 109, 135  
Control-M, 109  
Control-N, 102, 109  
Control-O, 103, 109  
Control-P, 102, 109  
Control-Q, 108, 135  
Control-R, 108, 135  
Control-S, 108, 135  
Control-T, 108, 135  
Control-U, 100, 101  
Control-V, 105, 109  
Control-W, 18, 109, 135  
Control-X, 18, 100, 109, 136  
Control-Y, 11, 18, 37, 41, 87, 110, 134, 136  
Control-Z, 110, 45  
COPY, 42, 57, 81  
Copying a block of text, 103  
Copying a disk, see IMAGE  
Copying files, 42  
Cursor, 8, 100  
Custom printer, 121, (sample program) 183  
Data, 85, 192  
Data files, 23, 91  
Debugging, 64  
Decimal, 85, 192  
DEF, 93  
DEFAULT, 117, 121  
Default drive, 34  
Default printer, 117, 121  
Default REENTER address, 66, 86  
DELETE, 13, 51, 52, 83  
DELETE key, 9, 18, 100  
Deleting a block of text, 103  
Deleted files, 13, 51  
Deleting characters, 18, 101  
Deleting files, 13, 51  
Deleting lines, 19, 100  
Deleting words, 18, 100  
DIO, 127-128  
DIRECTORY, 51, 122  
Directory, 9, 41, 47, 132  
Directory header, 52, 41, 47  
DISABLE, 47  
Disabled mode, 47, 73  
Disk, 8, 15  
Diskette, See Disk  
Disk directory, see Directory  
Disk drives, 17, 132, 163, 165-169, 173-175  
Disk initialization, see INIT  
Disk name, see INIT, DNAME  
Disk specifier, 30, 32, 132

Disk surface test, see INIT  
Displaying disk directory, 41, 51  
Displaying files, 106  
Dividing files, 106  
DNAME, 60  
Double-density, 5, 15  
Double-sided, 5, 15  
Drive, see Disk drives  
Drive lights, 10, 17  
DUP, 187ff  
EDIT, 13, 97  
Editor, 13, 23, 97, 193  
ENABLE, 19, 46, 106  
Enabled mode, 46, 68, 69  
END, (in Assembler), 113  
Erasing, see deleting  
Erasing a disk, see INIT  
Error messages, 45, 125 ff  
ESC (escape) key, 103, 104, 106, 107  
Escape sequences, 110  
EXEC, 45, 46, 47  
Exec, (see Executive)  
Execute, 192  
Executive, 45 ff, 80, 89, 91, 113, 122, 125  
EXIT, 96  
Exiting from Editor, 120  
Expression, 192  
Extensions, 26, 52, 60, 49, 80, 82, 130  
Files, 4, 25, 193  
File commands, see BASIC commands  
File names, 25, 129, 130  
Filename, 64, 65  
File specification, See Path name  
File transfer program, 185ff  
Firmware, 149  
Flip, 19  
fold, 19  
Form feed, 14, 155  
From address, 64  
Front panel, 46, 66, 149  
FULL, 18-19  
GET, 66, 47, 87  
GFID, 129-130  
Graphics characters, 139ff  
Graphics in BASIC programs, 16  
Graphics in machine language programs, 142  
Graphics character set (chart), 145  
Hardware, 149, 161ff  
    8810 hardware, 173ff  
    8813 hardware, 163ff  
Hexadecimal, 49, 85, 194  
IMAGE, 67, 83  
INIT, 54, 68, 134  
Initializing a disk, 54, 68, 69  
INITIAL program, 10, 11, 75, 93

Input, 44, 102, 193  
Input file, 13, 198-100, 105  
Input Processing, 18, 44  
Invocation, 193  
Invoking Assembler, 112  
Invoking BASIC, 9, 34, 77, 91  
Invoking BASIC programs, 79  
Invoking Editor, 98-100  
Invoking files, 39, 40, 45, 86  
Invoking machine language files, 86  
Inserting disks, 8, 20  
I/O, 21, 128-129, 193  
I/O connectors, 139  
I/O hazards, 20-21  
Keyboard, 9, 17  
LIST, 10, 50  
Listing assembly-language programs, 112  
LOAD, 80  
Load address, 48, 63, 66, 87, 133, 196  
Load button, 19, 70, 72  
Machine language, 21, 55, 85, 86, 112, 196  
Machine language program, 41, 65, 66, 67, 85ff, 111, 131  
Macros, 111  
Memory, 4, 166, 175, 196  
Memory address, 196  
Memory map, 147-148  
Minicards, 164, 173  
MIRROR, 189ff  
Mode, 197  
Monitor, 74, 149  
Monitor commands, 151-158  
Moving a block of text, 103  
NEW (printer), 117  
Output, 195  
Output file, 13, 97-98, 105-106  
Overlays, 22, 47, 130  
PACK, 57, 58  
Packing disks, 57  
PAGE, 59  
Partial path name, 34, 35  
Path names, 32-35  
Power supply, 163, 165, 173, 174  
PRINT, 9, 58, 72, 83, 115  
Printer, 115  
Printer editor, 115ff  
Printer driver, 115ff, 130, 195  
Printer SET, 124  
Printer SHOW, 124  
Printing a block of text, 105  
Printing from BASIC, 123  
Printing from Exec, 122-123  
Program, 23, 27, 86, 87, 197  
Prompt, 8, 195  
Question mark, see Special symbols ? and #  
RAM, 19, 149, 197

Rear panel, 162, 173  
Recovering deleted characters, 101  
Recovering deleted files, see Undeleting files  
REENTER, 66, 87, 106, 130  
Register, 196, 150  
RENAME, 60-62  
RENAME (printer), 116  
Renaming files, 60-62  
Restarting system, 19  
Reversing upper and lower case, 105  
Roll-over, see Type-ahead  
ROM, 74, 149, 198  
RUN, 11, 82, 84  
Safety, 167, 176  
SAVE, 9, 63, 83  
SAVE (in BASIC), 62-63  
Save address, 199  
Saving, 9, 133, 199  
Saving BASIC programs, 9, 82  
Saving machine language programs, 63  
SCR, 84  
Sector, 16, 63, 64, 199  
Setup, 115  
SHIFT key, 17  
Single-density, 5, 15  
Single-sided, 5, 15  
Software, 149  
Special symbols ? and #, 37, 38, 39, 50, 51, 55, 56, 57, 58,  
59, 62, 64, 67, 68, 87, 91, 132  
START, 65, 73, 83, 87, 130  
Start-up, 7, 19, 73  
String, 197  
Switches, 165, 173  
Symbols, 112  
System commands, 46ff  
CONTINUE, 67, 71  
COPY, 59, 70, 71, 83  
DELETE, 51-55, 71, 83  
DIRECTORY 50, 71  
DISABLE, 47, 71  
DNAME, 69, 71  
ENABLE, 47  
GET, 65, 71, 87  
IMAGE, 67, 71  
INIT, 59, 68, 71  
LIST, 9, 50, 71  
PACK, 57, 72  
PAGE, 59, 72  
PRINT, 9, 58, 72  
REENTER, 66, 72, 87  
Reference list, 71  
RENAME, 60, 73  
SAVE, 9, 63, 73  
START, 65, 73, 83, 87  
TYPE, 10, 58, 73, 83

UNDELETE, 55, 73  
ZAP, 69, 73  
System Disk, 4, 5, 67, 68, 74, 129  
System Drive, 4, 5, 34, 74, 129  
System extensions, 26  
System file, 24, 44, 48, 59, 63, 83, 93, 133  
System overlay, 23, 24, 48, 131, 133  
System Programmer's Guide, 6, 23  
System prompt, 8, 45, 74, 97  
System Start-up, 8, 19, 22, 74  
TAB key, 12, 20, 200  
Text, 12, 85, 97ff, 200  
Text file, 12, 25, 89  
Top of memory, 74, 130  
TYPE, 10, 58, 74, 83  
Type-ahead, 17, 201  
UNDELETE, 55, 73  
Undeleting files, 55, 56  
Variable, 198  
Verify, see INIT,  
VIEW, 116  
Warm-start, 66, 67, 201  
Warm-start address, 66, 88  
Wild card, 26, 54, 60, 61  
Wrap-around, 101  
Write-enable notch, 16, 128  
Write protect, 16, 128, 201  
Write protect tab, 16, 128  
ZAP, 69, 73