DOS COMMAND SUMMARY

BLOAD	X,Aa,Ss,Dd,Vv
BRUN	X,Aa,Ss,Dd,Vv
BSAVE	X,Aa,Ll,Ss,Dd,Vv
CATALOG	Ss,Dd,Vv
CHAIN	X,Ss,Dd,Vv
DELETE	X,Ss,Dd,Vv
INIT	X,Ss,Dd,Vv
LOAD	X,Ss,Dd,Vv
LOCK	X,Ss,Dd,Vv
RENAME	X,Y,Ss,Dd,Vv
RUN	X,Ss,Dd,Vv
SAVE	X,Ss,Dd,Vv
UNLOCK	X,Ss,Dd,Vv
VERIFY	X,Ss,Dd,Vv

Parameters	Definition	Range
a	Memory Address	0-65535
d	Drive Number	1-2
l	Length	1-32767
s	Slot Number	1-7
v	Volume Number	1-254
X	File Name	1-30 Characters
Y	New File Name	1-30 Characters

CONTROL COMMANDS

FP	Ss,Dd
IN#	s
INT	
MAXFILES	n
MON	C,I,O
NOMON	C,I,O
PR#	s

Parameters	Definition	Range
C	Commands	
d	Drive Number	1-2
I	Inputs	
n	Number of Files	1-16
O	Outputs	
s	Slot Number	1-7

INSTALLATION MANUAL FLOPPY DISK SYSTEM A2

MICRO-SCI

TABLE OF CONTENTS

Page

NOTICE

MICRO-SCI reserves the right to make improvements in their products at any time, and without prior notice. While every precaution has been taken in the preparation of this manual, MICRO-SCI assumes no responsibility for errors or omissions in this document; nor is any liability assumed for direct, indirect, incidental, or consequential damages resulting from the use of the information contained herein.

2158 South Hathaway Street
Santa Ana, California 92705

(714) 662-2801
(714) 662-2906

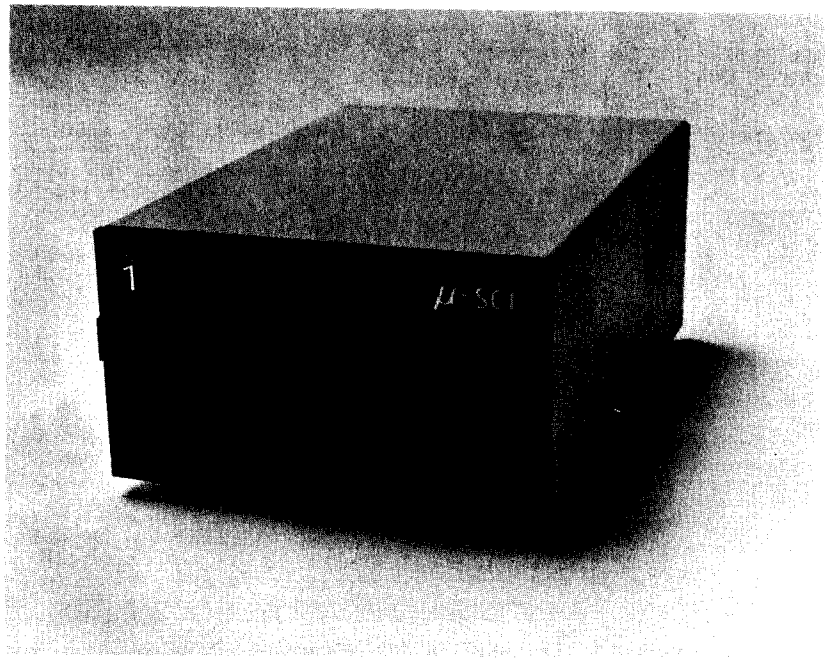
1. INTRODUCTION

This manual describes the operation of the MICRO-SCI A2 disk subsystem. The A2 subsystem is intended to be used with an Apple II or Apple II Plus computer system. The A2 subsystem is a direct replacement for the Apple Disk II floppy disk drives and controller.

Before attempting to connect the A2 subsystem to the computer, it is recommended you read through the chapters on Unpacking, Familiarization, and Installation. This will take only a few minutes but could eliminate an installation problem which might cost you several hours. In fact, the vast majority of the problems with any new piece of computer equipment are usually "Operator Problems." This is to be expected. The disk subsystem and its control program, the Disk Operating System (DOS), are fairly complex and, like a programming language (Basic, Fortran, or Pascal), there are procedures and rules which must be followed.

The first chapters of this manual will guide a first-time disk user through the process of unpacking and installing the A2 subsystem. To verify that the disk subsystem is operational, Chapter 5 will take you through a short checkout procedure. Chapter 6 deals with the operation of the DOS. The DOS is the program which controls the disk drives. To take full advantage of the computer system, you will need to become very familiar with DOS.

If this is your first disk subsystem, we recommend you spend some time reading a few sections of Chapter 6 before attempting to install and check out your new drives. Reading Chapter 6 up through Section 6.5.4, General Housekeeping, will give you a good overview of what the disk drives can do for you, and at least a preliminary idea of how to operate them.



THE DRIVE UNIT

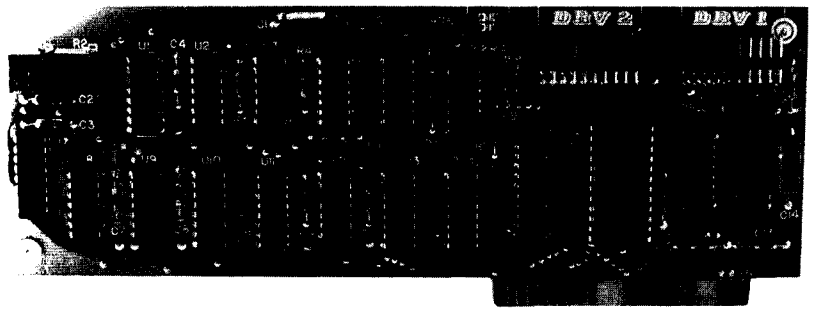
2. UNPACKING

After removing your equipment from the shipping container, you should verify that you have received the following items:

1. A2 Disk Drive with cable attached
2. Controller Board
3. Disk Drive ID Tags
4. A2 Utility Diskette
5. This manual

The A2 shipping box has been specifically designed to protect the disk drive and controller during shipment. We suggest you keep this packaging material for use in the event you have to ship the equipment to a dealer or the factory for service.

Once you have unpacked the equipment, look for any obvious shipping damage. If something appears to be bent or broken, it is best if you contact your dealer or MICRO-SCI before connecting the equipment to the computer.



THE CONTROLLER UNIT

3. FAMILIARIZATION

The MICRO-SCI A2 disk subsystem consists of two components, the drive unit and the controller unit. The controller is the small electronics board which will be installed in one of the Apple computer expansion slots. The drive unit is the large rectangular box with a flat ribbon cable attached to the back.

If you look closely at the controller card, you will see two connectors, one labeled DRV 1 and the other DRV 2. During the installation procedure, you will be required to connect the cables from the disk drives to these connectors on the controller board.

Next to the DRV 2 connector is a small 4-pin connector with a removable gold clip between the center two pins. This clip is called the configuration jumper. Using your fingers you can remove this jumper from the 4-pin connector. There are three possible connections for the jumper. When the jumper is connected to the left two pins (there is a small 13 under these two pins), the controller is configured to Boot DOS 3.2 type diskettes. (This will be discussed in more detail in Chapter 6.) When the jumper is connected to the center two pins, the controller will not Boot any diskettes but will display a message on the computer display screen indicating the proper positions for the jumper with DOS 3.2 and DOS 3.3 type diskettes. When the jumper is connected to the right two pins (there is a small 16 under these two pins), the controller will Boot DOS 3.3 type diskettes. The controller is normally shipped with the jumper on the middle two pins.

On the extreme left side of the controller, there are two small rectangular boxes. These are called potentiometers and are used to calibrate the controller. They are set at the factory, and under no circumstances should you attempt to adjust them.

A third, and very important component of your disk subsystem, is the diskette. Diskettes are thin, flat, and lightweight, which makes them easy to store and transport. It also makes them very susceptible to abuse. One of the major factors in the reliability of your disk subsystem will be the care you exercise in handling your diskettes.

Diskettes are similar to audio magnetic tapes. The "tape" in this case is circular and enclosed in a protective plastic jacket. The recording surface is visible through the oblong cutout at one end of the diskette. You should be very careful when handling and storing diskettes that this area is always protected. Dust, dirt, fingerprints, or spilled liquids on the recording surface can affect the information stored on the diskette. Furthermore, if you put a contaminated diskette in a disk drive, in an attempt to recover the information stored there, the contamination can be deposited on the disk drive Read/Write head and subsequently be transferred to other diskettes. Scratches on the recording surface can also affect the data stored on the diskette. Information is stored on the diskette in .012 x .0002-inch areas. With dimensions this small, even invisible scratches can be damaging. Obviously bending or folding a diskette is not recommended.

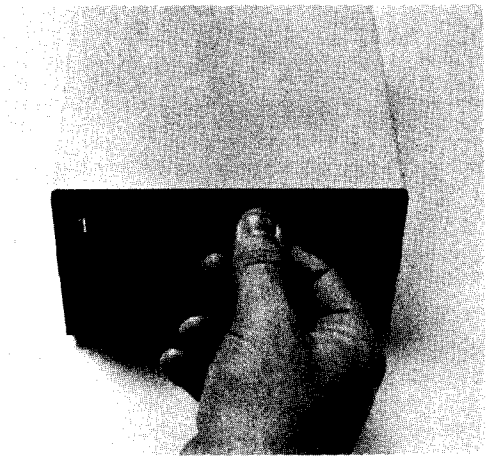
As you begin to collect diskettes, you will want to label each one. The suggested method is to first write on the label, then put the label on the diskette. If you need to change something on a label you should only use a felt tip pen. Ball point pens can easily scratch the recording surface, even through the protective jacket.

As a last word of caution, both heat and magnetic fields can erase information stored on a diskette. Be careful where you set or leave your diskettes.

If you hold a diskette with the label facing you and the oblong slot down, you will probably have a small notch about 1 inch from the top on the right side. This is called the Write-Protect notch. When this notch is covered, or missing, in the case of some Program Master diskettes, the disk drive Write circuitry is disabled. This is the safest way to insure that you do not accidentally "overwrite" a piece of valuable information. When you purchase a box of new diskettes you will usually receive a supply of small write-protect stickers. If you wish to "write-protect" a diskette, wrap one of these stickers around the edge of the diskette until the write-protect notch is covered on both sides. If you ever want to write on the diskette again, you must remove the write-protect sticker.

If you look closely at the front of the drive unit, you will see it has a small door which you can swing open with your fingers. To put a diskette into the drive you should hold it with the label facing up and between your fingers. Slide the diskette into the drive -- notice that the oblong slot end goes into the drive first. It is very important that you push the diskette all the way into the drive, or improper registration between the diskette and the drive may cause disk Input/Output (I/O) errors. With the diskette in the drive you can now close the door.

In the lower left-hand corner of the front of the drive, there is a small red light. This is called the "IN USE" indicator. Each time the computer accesses the disk drive, this light will come on for a few seconds and go back off. It is recommended that you do not remove a diskette from the drive when this light is on.

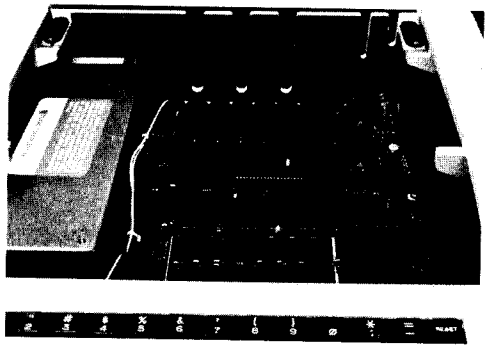


OPENING THE DOOR ON THE DRIVE

4. INSTALLATION

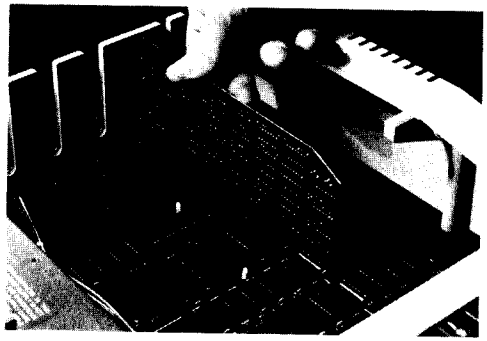
During the installation procedure, you will be instructed when to turn on the computer power and when to turn it off. Keep in mind that you should never plug or unplug any cables or boards with the power applied to the computer.

1. Turn off the computer power.



APPLE II I/O SLOTS

2. Remove the top from your computer. Install the A2 controller into Slot Number 6 of your computer. The Apple II Reference Manual, Page Number 89, describes these slots. Slot Number 6 is the second connector from the right. The A2 controller will work if installed in any of the slots but Slot 0. However, the recommended position is Slot 6 for your first two drives, then Slot 5 for your second disk controller and third and fourth drives, and Slot 4 for a third controller, etc.



INSTALLING THE CONTROLLER IN YOUR APPLE COMPUTER

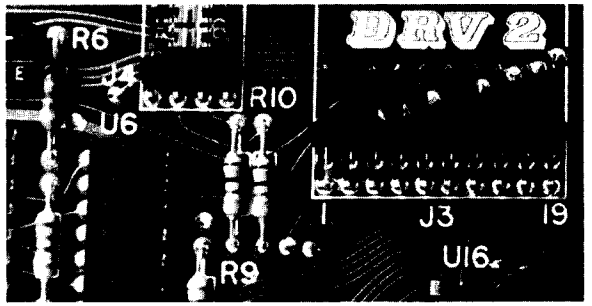
3. Make sure the configuration jumper is installed on the center two pins. You can now turn on the computer. If you have an Apple II Plus, the Auto-start feature will have executed the "Boot" program in the A2 controller. With the configuration jumper on the center two pins, this program will display a message on the computer display monitor. If you have a standard Apple II, when you turn on the power the computer executes the Monitor program. To "Boot" the disk drives, enter Basic (Control-B, Return) and then issue an IN#6 command.

Your computer monitor should have the message displayed in Figure 1. If you have this message, we now have some confidence that several parts of the controller are operational and we can continue on with the installation procedure.

```
--KEYBOARD              MICRO-SCI                REAR--

------------------------JUMPER-------------------------

   !                       !                        !
   V                       V                        V
### ! !                 ! ### !                  ! ! ###
### ! !                 ! ### !                  ! ! ###
### ! !                 ! ### !                  ! ! ###
### ! !                 ! ### !                  ! ! ###
### ! !                 ! ### !                  ! ! ###
XXXXXXX                 XXXXXXX                  XXXXXXX

DOS 3.2                   INFO                   DOS 3.3
                                                 &PASCAL
```

Figure 1

4. Turn off the computer power.



SELECTING DOS 3.2 OR 3.3 WITH THE CONFIGURATION JUMPER

5. Unplug the A2 controller. It is now time to connect the cable from the disk drive to the controller. There are two 20-pin connectors on the controller board labeled DRV 1 and DRV 2. If you have only one drive, you must connect that drive to the DRV 1 connector. The connector

attached to the drive cable is keyed. Notice that this connector is flat on one side and has a raised arrow on the other side. Place the flat side of the connector against the controller circuit board and push the cable connector down onto the controller connector. Once attached, the flat cable should exit away from the controller board and should not be between the connector and the controller board.

```
**********
* WARNING *
**********
```

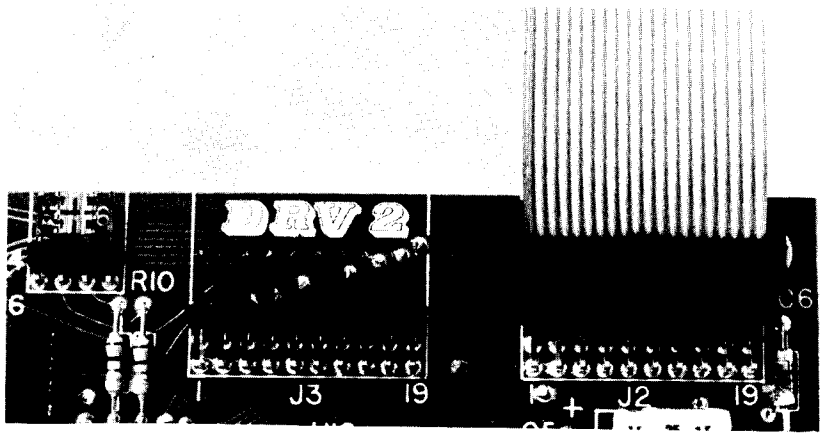
Installing the cable incorrectly will cause damage to the drive electronics.

If you have a second disk drive, you should connect it to the DRV 2 connector. This connector should be installed just like the DRV 1 connector.

6. Move the configuration jumper to the DOS 3.3 format position. This is with the jumper connected to the right two pins. Notice there is a small 16 under these pins.

7. Reinstall the controller into Slot Number 6 of the computer. Once you have the controller installed, fold the flat cables so that they exit to the rear of the computer.

This completes the installation of your disk drives. The physical location of the disk drives in relation to the computer and display monitor is important. The display monitor is an electromagnetic device and it emits electrical signals which can interfere with the operation of the disk drives. It is best if you do not place the disk drive directly under or next to the display monitor. One possible configuration is with your monitor on top of the computer and the disk drive on the table next to the computer.



CONNECTING THE CABLE TO THE CONTROLLER

5. CHECKOUT

The diskette labeled A2 Utility Diskette that you received with your A2 system contains a diagnostic checkout program. This program will check the Read, Write, Speed, and Write-Protect circuitry of the A2 system. To check out your system, put the A2 Utility Diskette into Drive 1. Check the configuration jumper on the controller and verify that it is in the DOS 3.3 (16 sector) position. Turn on the computer power switch. If you have an Apple II Plus, the disk drive should be active; that is, the Activity Light should be on, the drive motor spinning, and the Read/Write head stepping back to Track 0. You cannot see the drive motor or stepper motor, but you can hear them. If you have a standard Apple II, you will need to enter Basic (Control-B, Return) and then execute an IN#6 command to start the Boot process. If you have problems getting the A2 Utility diskette to Boot, you should recheck the installation procedure and review Section 6.5.1, Booting DOS. If you are still having problems, you will need to contact your dealer or MICRO-SCI for assistance.

The Boot process should take only a few seconds. When it is completed, the computer monitor will display a message indicating that the diagnostic program has successfully Booted. Once the diagnostic program is loaded, it will direct you with a series of messages. The following test requires that you leave the A2 Utility Diskette in the drive.

The first test is the Seek and Read test. At the completion of this test, the diagnostic program will display a pass or fail message. Assuming the drive passes the test, you should continue on to the next test. If, however, the drive fails the test, there are still a few things for you to try. First try recentering the diskette by opening the door, removing, and reinserting the diskette. Close the door and select the retry option. If the test passes this time, continue on with the rest of the tests.

If the drive still fails the Seek and Read test, it could be the diskette. If you have another DOS 3.3 formatted diskette (not a copy-protected diskette such as a game) then put a Write-Protect sticker on that diskette. Replace the A2 Utility Diskette with the other DOS diskette, and then select the retry option. If the drive passes this time, assume the A2 Utility Diskette has a bad sector and that the drive is operational. Before continuing on with the tests, we suggest you remove your DOS diskette and reinsert the A2 Utility Diskette. The Speed and Write tests could destroy some files on the DOS diskette.

If you are still unsuccessful in getting the Seek and Read test to run, you should contact your dealer or MICRO-SCI for assistance.

The second test is the Speed Check. The rotational speed of the drive is adjustable. However, this check is strictly a Go/No-Go test. The speed adjustment potentiometer is inside the drive unit. The A2 system has four adjustable potentiometers: two for the Read circuitry, one for the Write current, and one for Speed. Speed calibration is the only user adjustment. If you turn one of the other three potentiometers by mistake, you will have to return the system to a dealer or the factory for calibration. We strongly recommend that you do not experiment with the potentiometer settings.

At the completion of the Speed Check, the diagnostic program will display a pass or fail message. Assuming the drive passes the Speed Check, continue on to the next test. If the drive fails the

Speed Check, you should select the retry option several times to see if the failure is hard or intermittent. The A2 Utility Diskette has a Speed Calibration program which can be used to adjust the speed of your drive. Section 7 of this manual describes the operation of the program. If your drive fails the Speed Check, refer to Section 7 of this manual for more information.

The third test is the Write-Protect Check. When you start this test, the program will display the state of the Write-Protect status. The A2 Utility Diskette is not write-protected, and the diagnostic program should display a NOT PROTECTED status. If you open the door and partially remove the diskette the write-protect status should change from NOT PROTECTED to PROTECTED. Push the diskette back into the drive and the status should change back to NOT PROTECTED.

If the drive fails this test, indicated by the failure of the status response to change, the drive unit will require service. Please contact your dealer or MICRO-SCI for assistance.

The last test is the Write Test. This test reads one sector from the diskette, changes a byte, writes the sector back to the diskette, reads the sector again, and then verifies the data. The diskette must not be write-protected. At the completion of the Write Test, the program will display a pass or fail message. Assuming the drive passes the test, you should continue on to the next phase. If the drive fails the Write Test, you should check to see that the A2 Utility Diskette is not write-protected. If this is not the problem, you will need to contact your dealer or MICRO-SCI for assistance.

After four tests, the program will display a menu which will allow you to select the other drive for test or re-Boot the system. If you have only a single drive, or have already tested both drives, then you are finished with the diagnostic program. This completes the checkout of the A2 subsystem.

6. OPERATION

6.1 TABLE OF CONTENTS

6.2 INTRODUCTION

Your new disk drive subsystem greatly expands the capabilities of your Apple II. By increasing the amount of information that your computer can immediately access, you have opened your system to a whole new world of applications. These new applications require software to control the storage and retrieval of information between the Apple and the disk subsystem. The program that controls the disk subsystem is called the Disk Operating System (DOS). The purpose of the DOS is to provide an easy-to-use set of commands to allow a programmer to access the information stored on the disk drives. Before we get too involved with the description of the DOS command set, let's take a closer look at how a DOS works.

There are actually several DOS's available for your Apple II computer and disk drive subsystem. For discussion purposes, we will divide these DOS's into two categories: general purpose and special purpose DOS's. A general purpose DOS is one which supports multiple application programs and usually several languages. Some examples of these DOS's are: Apple DOS 3.2, Apple DOS 3.3, Pascal, and CP/M.

In addition to the general purpose DOS's, there are many more special purpose DOS's. These DOS's are usually a part of an application program and are used only with that program. Some examples of special purpose DOS's for the Apple II are: VISICALC by Personal Software, DB MASTER by Stoneware, the DATA FACTORY by Micro-Lab, and virtually all of the game programs which are distributed on individual diskettes.

At this point you might wonder why there are so many different DOS's for the same computer. Keep in mind you have purchased a general purpose computer system. This means your Apple II is capable of being used in many different applications. With a given application, a specific data organization on the diskette may be very efficient, while in another application that same data organization may be almost unusable. Therefore, we have ended up with quite a few different DOS's. Now, if you are totally confused and are wondering which is the best DOS for you, the answer is quite simple. As we previously explained, a special purpose DOS is part of an application program. Therefore, these DOS's are evaluated on the merit of the individual application program.

On the other hand, the general purpose DOS's (DOS 3.3, CP/M, and Pascal) must be evaluated on their ease of use, the languages supported, the speed of the disk accesses, their versatility, the cost effectiveness, and the number of application programs available. In other words, depending on your application and your budget, any one or all three of these DOS's could be useful to you.

Apple DOS 3.3 is the most common DOS used on the Apple II computer. For the rest of this chapter we will attempt to describe DOS 3.3. Since 3.3 is a general purpose DOS, most of the concepts discussed here can also be applied to CP/M and Pascal.

Most DOS's can be divided into three general areas: the Equipment Interface, the File Manager, and the User Interface. To take full advantage of a DOS, you will only need to understand the User Interface section. However, a general understanding of the other two sections is helpful.

The following descriptions of the Equipment Interface and the File Manager are included as an overview of how a DOS works. The

information included here is not intended to provide a detailed understanding of either of these sections of the DOS. For a more detailed description of Apple DOS, the publication "Beneath Apple DOS" is highly recommended and is obtainable from:

Quality Software
6660 Reseda Boulevard, Suite 105
Reseda, California 91335

(213) 344-6599

6.3 THE EQUIPMENT INTERFACE

This section of Apple DOS is usually called the RWTS (Read/Write Subroutine). This is the only section of the DOS which communicates with the hardware. The functions performed by the RWTS are: Drive Selection, Motor Control, Head Seeking, plus all of the Read and Write functions. This is also the section of the DOS which determines the physical format of the diskette.

All floppy disk subsystems (Apple, Tandy, Atari, IBM, etc.) divide the total diskette storage capacity into smaller "chunks" called tracks and sectors. A track is similar to a groove on a record; however, on a floppy disk the tracks do not "spiral" in toward the center. With both the Apple Disk II and the MICRO-SCI A2, the tracks are .020 inch from center line to center line. Of this .020 inch, only .012 inch contains the recorded information. The rest (.008 inch) is erased to provide a "guard band" between tracks. The Apple Disk II and MICRO-SCI A2 can read or write 35 tracks per diskette. All floppy disk systems number the tracks starting at the outside of the diskette and working in. That is, Track 00 is the track farthest from the center of the diskette.

To summarize, a diskette is first divided into tracks which are concentric bands of recorded information separated by small guard bands. The drive mechanism is capable of positioning the Read/Write head to any one of these tracks.

It should be apparent that the higher numbered tracks (those closer to the center of the diskette) have a smaller circumference than the outside tracks. With most floppy systems (including Apple), the number of bits recorded per track is approximately 50,000. This means that the length of a bit on the inside track is smaller than a bit recorded on the outside track. In fact, bits on the inside track are as small as .000180 inch, while the same bit on an outside track is .000280 inch. 50,000 bits is 6,250 bytes per track. Although your computer can hold up to 48,000 bytes in RAM (Random Access Memory), handling 6,250 bytes per disk access would be rather inefficient. Therefore, each track is further divided into smaller, more convenient "chunks" called sectors. With Apple DOS 3.3, there are 16 sectors of 256 bytes each per track. The number of sectors per track (16) times the number of tracks (35) equals the number of sectors per diskette (560).

The RWTS must be able to individually access any of these sectors. To accomplish this, each sector has an associated address field which uniquely identifies that sector. The address field contains the track number, the sector number, and the diskette volume number. The RWTS uses these address fields to locate the desired 256-byte data sector.

To review, we have now defined the physical layout of the diskette in terms of tracks and sectors. There are 35 tracks with 16

sectors on each track, for a total of 560 sectors. Each sector has 256 bytes, for a total of 143,360 bytes per diskette. The RWTS is the section of the DOS which handles the transfer of data to and from any of the sectors on the diskette.

To use the RWTS, a programmer, or the DOS, must build a table called an IOB (Input/Output Block) and "call" the RWTS. This table defines the controller slot number, the drive number, the track number, the sector number, the volume number, the memory buffer address, and the command (READ or WRITE). The RWTS uses this information and selects the proper drive, makes sure the motor is up to speed, moves the Read/Write head to the appropriate track, locates the sector, and then for a READ command moves the entire 256-byte sector to the addressed memory buffer; or for a WRITE command moves the 256 bytes from the memory buffer to the proper sector on the diskette.

To summarize, the RWTS performs the function of locating a specific data sector on a diskette and either reading or writing that sector. Additionally, the RWTS does all of the error-checking for each READ or WRITE command.

6.4 THE FILE MANAGER

The RWTS performs the function of moving a specific sector from the diskette to memory (READ) or from memory to the diskette (WRITE). As previously explained, the RWTS must be told which sector to address and where to put that sector in memory. The File Manager is the part of the DOS which keeps track of what information is stored on each sector of the diskette. The File Manager does this by keeping a list of all of the "Files" stored on the diskette.

Before we continue, we should define a "File" and some other terms. By now you should be familiar with Basic and should know how to write a program. As you know, a program is a collection of commands (usually several bytes each) which the computer stores in its memory. The advantage of your disk drive subsystem is that it can permanently store the program on a diskette for future use. Since an average program is only a few thousand bytes long and a diskette holds over 100,000 bytes, it would not be very efficient to store only one program per diskette. To overcome this limitation we have to introduce the concept of a "File." Most simply stated, a file is a collection of related information. For instance, if we use the example of a file cabinet, each drawer can be considered as one of the drives connected to your computer. Within each drawer there can be many file folders (files) with each file folder containing a collection of related information (records).

In other words, when it comes time to save your program, you will have to create a "File" on a diskette to hold that program. This means the DOS must reserve a few sectors on the diskette to store your program.

As we originally stated, the File Manager is responsible for keeping track of the available and allocated sectors on the diskette. The File Manager keeps track of the available sectors on the diskette with one sector called the VTOC (Volume Table of Contents). The contents of this sector contain a Bit-per-Sector Map of the diskette. That is, each sector on the diskette has a corresponding bit in the VTOC Bit Map. Free (available) sectors have a 1 in their position in the bit map, and when they are

assigned to a file their corresponding bit in the map is set to a 0, signifying the sector is allocated.

To keep track of which sectors have been allocated to each file, the File Manager uses several sectors of the diskette as a Directory. The Directory is a list of all of the files which have been stored on the diskette. Each diskette has its own VTOC and Directory which pertain only to the information stored on that particular diskette. The Directory entries must contain a name, a file type, and a pointer to a sector which contains a list of the other sectors assigned to that file. It is these "other" sectors which actually contain the information which we want to store in the file.

To demonstrate how the File Manager works, the following examples will show the sequence of events for a program SAVE operation and then a program LOAD operation. Let's assume we have a program in memory ready to "SAVE" to a diskette. The operator must issue a SAVE command with a file name to the DOS (explained in Section 6.5.5). The File Manager will build an IOB and issue a READ command to the RWTS specifying the Directory sector. Once the Directory is in the computer memory, the File Manager will add the new file name to the Directory and, using the VTOC Bit Map, find the next available sector on the diskette. This first sector will be used to store a list of the rest of the sectors allocated to the file. Once a file name has been added to the Directory and a Track and Sector List sector has been allocated, the File Manager will begin to move the program from memory to the diskette. The program is stored on the diskette in 256-byte segments per sector. As each new sector is added to the file, it is removed from the VTOC Bit Map and added to the Track and Sector List. When the entire program has been stored on the diskette, the File Manager returns control of the computer back to the operator.

To load this program back into the computer memory, the operator must issue a LOAD command (explained in Section 6.5.5) with the file name that was used to store the program. The File Manager first reads the Directory and then searches for the file name. When it finds the file name, it extracts the Track and Sector List pointer and reads in the Track and Sector List. The File Manager then proceeds to load the program into memory using the Track and Sector List as a guide.

In summary, the File Manager uses several sectors on the diskette to keep track of "what" and "where" things are stored on the diskette. One sector called the VTOC keeps track of which sectors have been used and which ones are still available. Another 15 sectors are reserved for the diskette directory. The Directory contains the names of all of the files that have been stored on the diskette. Finally, each individual file consists of at least one sector called a Track and Sector List which contains a list of the rest of the sectors in the file. The actual information for each file is then stored in the sectors pointed at by the Track and Sector List.

6.5 THE USER INTERFACE

This section of the DOS, as its name implies, communicates with both the computer operator and any programs which access the disk drives. This section deals more with specifics and less with general concepts. We hope that the general overview of the RWTS and the File Manager will help you better understand the operation of the DOS. However, the rest of the information in this section

should be read thoroughly and should not be considered an overview. To get the most out of your new disk subsystem, you must become completely familiar with the DOS commands.

6.5.1 Booting DOS

Your Apple II computer comes with a built-in Basic language, either Integer or Applesoft. These languages are stored in ROM's (Read-Only Memories) that retain their instructions even when the power is removed from the computer. Integer uses about 8,000 bytes and Applesoft about 12,000 bytes of ROM memory. In addition to this ROM, your computer has 16, 32, or 48 thousand bytes of Read/Write RAM (Random Access Memory). If you intend to use DOS, you must have at least 32,000 bytes of RAM. By now you should be pretty familiar with what Basic can do. The process of "Booting" DOS will add more commands to Basic. These extra commands require about another 10,000 bytes of instructions for the computer. They are called DOS and are loaded into the highest addresses of the available RAM.

To Boot you must have a diskette with a DOS. The DOS always resides on Tracks 0, 1, and 2. The disk controller contains a small 256-byte program capable of reading a few sectors from Track 0. These few sectors on Track 0 are called the First Stage Boot and they in turn are capable of reading-in the entire DOS. To start the Boot process, you must execute the 256-byte program in the "Boot Prom" on the controller. If you plug a disk controller into an Apple II Plus and turn on the power, the computer will automatically execute the Boot program. On an Apple II Integer machine (or an Apple II Plus), you can execute the Boot program from the monitor or from Basic. In the monitor, you can issue a CS00G (where S is the slot number of the disk controller). From Basic you can issue an IN#S (where S is again the slot number of the controller).

Once the Boot process has started, the DOS will be loaded into RAM and connected to Basic. The available space for Basic programs is reduced by the size of the DOS (about 10,000 bytes) and a whole new set of instructions is added to the Basic.

You can Boot from any slot which contains a disk controller, but you can only Boot from Drive 1 connected to that controller. The Auto-start feature on an Apple II Plus will start looking for a disk controller in Slot 7 -- and then Slot 6, 5, 4, 3, 2, and 1. It will Boot from the first controller it encounters.

6.5.2 Getting Started

Let us assume you have just Booted DOS. Your computer is in Basic command mode and you can issue any of the standard Basic commands; or, if you want, you can write a program. In other words, you shouldn't notice anything different about your computer; however, there is a difference -- your Basic now contains a whole new set of commands. These new commands are the DOS commands which will permit you to use your disk drives. In many respects a DOS command is just like any other computer command. It consists of a command word followed by several parameters. One of the major differences between DOS commands and Basic commands is that you cannot put multiple DOS commands on the same line separated by colons.

Many of the DOS commands have the same parameters. Rather than describe these parameters over and over again with each command, we will present them here at the beginning of the DOS command descriptions.

File Name: In the description of the File Manager we explained the concept of a "File." Each file on a diskette must have its own unique name. A file name consists of 1 to 30 characters. The name must begin with a letter and cannot contain a comma (,). If you put control characters into the file name, they will not be displayed; however, they are part of the file name.

Each time you wish to access a file, you must specify the file name. By adding, deleting, or changing one character in a file name, you are referencing a different file. While you are getting accustomed to the DOS, it is best if you keep your file names short.

Slot and Drive Number: DOS 3.3 can address up to fourteen drives. That is, two drives per controller and seven controllers installed in Slots 1 through 7. To address any of these drives, there are two optional parameters: S for slot and D for drive. Unless a slot or drive parameter is issued, the DOS will default to the previously addressed drive. For example, after Booting DOS from Slot 6 Drive 1, the DOS will assume all of the commands are to Slot 6 Drive 1 until a command with a new S or D parameter is issued.

The S parameter must be followed by a number from 1 to 7, corresponding to the slot number for the new controller. If a slot without a controller is specified, an I/O ERROR will result. This I/O ERROR is not a serious problem, but the next DOS command must wait for the "drive" connected to this nonexistent controller to stop spinning. This creates a real problem. Usually the system hangs forever. If the system hangs, hit Reset. If you have issued a command with a bad S parameter, you must give the DOS a new command with a valid S parameter (such as CATALOG, S6). With Auto-start, you should be back in Basic. With an Integer machine, type 3D0G and you will be back in Basic. Once you are back in Basic, you can try the command again with the proper slot number.

The D parameter must be followed by the Number 1 or 2. Selecting a nonexistent drive with the D parameter is not as serious as the wrong S parameter. It will also result in an I/O ERROR, but this usually doesn't hang the computer. The S and D parameters are independent of each other and do not need to be issued together. If you wish to change drives on the same controller, you need only issue the D parameter. Likewise, if you wish to change controllers and not the drive number, then you need only issue the S parameter. When using both the S and D parameters, the order is not important. The S may precede the D or vice-versa.

Before we finish our discussion of the S and D parameters, we must introduce the term "currently logged drive." If a DOS command doesn't specify an S or D parameter, the command will access the currently logged drive. An S or D parameter will update the currently logged drive to the specified slot and drive.

Volume Number: Up to now we haven't introduced the concept of volume numbers, but we have mentioned them. If you remember in the description of the RWTS, we stated that the address field for each sector on the diskette contains its track number, sector number, and "volume number." A program (programmer) uses a Volume Number to identify a specific diskette. The only time a Volume

Number can be assigned to a diskette is when it is first "initialized." (See the description of the INIT command for details.) It cannot be changed without re-initializing the diskette.

The best way to visualize the value of Volume Numbers is to compare them to S and D parameters for selecting between physical drives, File Names for selecting various logical groups, and Volume Numbers for selecting between physical diskettes. When you are initializing a diskette, you may optionally specify a V parameter (Volume Number). The V parameter must be in the range of V1 to V254. If you do not specify a V parameter, the DOS will assign a default Volume Number of 254.

When you are using the rest of the DOS commands, if you include a V parameter, the DOS will not execute the specified command unless the Volume Number on the diskette matches the V parameter exactly. If you do not specify a V parameter, the DOS will use V0 which is a "wild card" response and matches any Volume Number assigned to the diskette. Likewise, specifying V0 in a command will match any Volume Number.

When the DOS encounters a volume mismatch error, it will terminate the current command before it executes and prints the error message VOLUME MISMATCH.

6.5.3 File Types

Sections 6.3 and 6.4 have described the physical and logical organization of the diskette. We know that all of the information on the diskette is stored in logical groups called "Files," and that each file is made up of several physical 256-byte "Sectors." The real value of the DOS is the ease with which we can create and manipulate these "Files." With DOS 3.3 there are actually three different types of files: Program Files, Binary Files, and Text Files.

If you are just learning to use your computer, you probably will feel most comfortable with the Program Files. This type of file stores a Basic program (Applesoft or Integer). To create a Program File you need to write a program, and then type: SAVE X where X is the file name. The DOS will then move your program to the diskette and add the name you just specified to the directory. Now a little review is in order. The SAVE X command could have taken the form:

```
SAVE X,Ss,Dd,Vv
```

where X is the file name parameter described in the last section, and S, D, and V are optional Slot, Drive, and Volume parameters. But you knew all of that because we have already discussed those parameters.

Now that we have created a Program File, let's do something with it. Let's say we want to RUN the program we just created. The command RUN, without a file name parameter, will execute the program that is currently in memory. However, if we type:

```
RUN X
```

the DOS will check the "currently logged drive" for the file name X and, if it finds it, the DOS will load it into memory and immediately execute it. If the DOS doesn't find the file, it will respond with the error message FILE NOT FOUND.

We can now create any number of different program files by using different file names for each program we want to store on the diskette. To bring these programs back into memory and execute them, we need only type:

```
RUN X
```

where X specifies the proper file name. There is one more thing we might want to do to a program file and that is to load it into memory so that we can edit (change) the program. The DOS command LOAD X will do just that. It should be obvious that X is again the file name.

The three DOS commands we have just introduced are used to manipulate Basic program files. There is a second type of program file which we refer to as a Binary File. A Binary File is usually an assembly language file (6502 machine code) that is used by more advanced programmers. We will discuss Binary Files in more detail later in this section, but for now let's just introduce the DOS commands that are used with Binary Files. They are BSAVE, BRUN, and BLOAD. As you can see, they have the same form as the LOAD, SAVE, and RUN commands but are preceded by a "B" signifying a Binary File.

Finally, the third type of file is called a Text File. Up to this point the only things we could store on the diskette were programs. What if we want to store some data for a program? The data could be numbers, like payroll or your checkbook balance, or maybe the results of some involved calculation. On the other hand, the data could be mostly words like a letter, some messages, or items in an inventory system. Anyway, for the DOS to be really useful, it must allow us to store this type of information as well as programs. DOS 3.3 does allow us to store this kind of information and it does so in a file called a Text File. We will discuss these Text Files in more detail later; for now we just want to introduce the Text File commands:

```
OPEN
CLOSE
READ
WRITE
APPEND
POSITION
EXEC
```

You have now been introduced to the three file types: Program, Binary, and Text Files. You have been introduced to the concept of SAVEing, LOADing or RUNning a Basic program. You have seen that there are also Binary Files which you would BSAVE, BLOAD, or BRUN (if you knew all of the specifics). And finally, there are the Text Files which store data for programs and require a whole bunch of DOS commands which don't seem quite as friendly as program file commands.

6.5.4 General Housekeeping

We know that if we "Boot" a DOS diskette, we can get DOS hooked up to Basic. This means we have a bunch of new commands that make the disk drives work. We know that we can store different kinds of information on the diskette in different file types. In this section we are going to describe seven new DOS commands that can be used with all three file types. They are:

CATALOG
DELETE
LOCK
UNLOCK
RENAME
VERIFY
INIT

These seven commands have two things in common. First, none of these commands will alter the program memory. If you have a program in memory and you issue one of these commands, the program will not be changed. Second, all seven of these commands can use the optional Slot, Drive, and Volume Number parameters.

CATALOG This command allows the operator to see what is stored on a diskette. CATALOG is one of the few DOS commands which does not require a file name. When the operator types CATALOG, the DOS will list all of the files on the diskette in the currently logged drive. In addition, it will display: the LOCK/UNLOCK indicator (* means a file is locked); the File Type (I means Integer, A means Applesoft, B means Binary, and T means Text); the file length in sectors (from 1 to 255; files longer than 255 sectors start over from 000); and, finally, it will display the file name (control characters are not displayed by the CATALOG command). The CATALOG command also displays the diskette Volume Number. If more files are stored on the diskette than will fit on the monitor screen, the DOS will display the first 20 files and then wait for the operator to hit a key. It then displays 20 more files and waits for a key again, and so forth until all of the files have been displayed.

DELETE This command is used to remove a file from a diskette and make the sectors that the file occupied available to other files. In other words, let's say you have a few old programs stored on a diskette and you don't need them anymore. You can type:

```
DELETE X
```

where X is one of the old file names and the DOS will remove that file from the directory and put the sectors used by that file back into the VTOC Bit Map so that they can be used by other files. You can then continue to use the DELETE command until you have removed all of the old files. You now have some more space on that diskette to use with your new programs.

LOCK/UNLOCK After spending several hours or days getting a program ready to use, the last thing you would want would be to accidentally erase or change that program. The LOCK command can be used to write-protect a particular file. When you issue the DOS command

```
LOCK X
```

the DOS will set a flag in the directory entry for the File X so that the file cannot be erased or written to. The LOCK command has no effect on loading, running, or reading the file.

Once a file is locked, the only way to erase or change the file is to first unlock it. The DOS command

```
UNLOCK X
```


will reset the flag in the directory entry. Locked files are displayed by the CATALOG command with an asterisk (*) preceding the file type.

RENAME Let's say we have a program that seems to be working pretty well and we decide to add some new features to it. When we are done, we will want to call the new version of the program by the same name as the previous version. However, being a smart programmer, we know the old version worked pretty well and we don't want to throw it away, so we issue the DOS command

```
RENAME X,Y
```

and the DOS will change our old file X to a new name Y. We now have the old version of our program stored under the new name Y, and we can use the old name X for our new version.

VERIFY This command is used to check if the RWTS can successfully read all of the sectors in a file. Let's say we are having some trouble with one of our disk drives and we are occasionally getting I/O ERRORS. The VERIFY command will find bad sectors which the RWTS cannot read. This command does not check the actual information in a file and, therefore, if you mess up a program and then store it in a file, it will verify as a good file. The DOS command

```
VERIFY X
```

will check the File X for bad sectors. If VERIFY encounters a bad sector, DOS will display the I/O ERROR message.

INIT Up until now we have told you how to do all kinds of neat things with your new disk drives by using the DOS commands. However, we haven't told you how to make new diskettes so that you can do all of these things. If you go down to your local computer store and buy a brand new box of diskettes and put one of them in a disk drive and type CATALOG, you would probably expect the DOS to tell you there are no files on that diskette. Well, actually, the DOS will say I/O ERROR. This is because the RWTS won't be able to find any address fields (Track, Sector, Volume Numbers, etc.) and so it can't tell the difference between a totally blank diskette (the one you just tried to CATALOG) and a diskette that was accidentally clobbered. This is where the INIT command comes in. New diskettes must be INITialized before they can be used. Old diskettes that start picking up a lot of I/O ERRORS can sometimes be reINITialized and used again.

To use the INIT command, you must first put a Basic program in memory. You can LOAD a program from another diskette or write one from scratch. This program will be the greeting program. That means whenever you Boot this new diskette, created by the INIT command, this program will be automatically invoked.

The DOS command

```
INIT X,Vv
```

will format all of the tracks with the proper address fields containing the Track, Sector, and Volume Numbers. It will then create the VTOC sector with the proper bit map configuration; create the empty directory sectors; move the DOS to Tracks 0, 1, and 2; and, finally, it will save the program you put in memory

under the file name X. The file X will then become the greeting program. That is, whenever you Boot this new diskette, program X will automatically be RUN at the completion of the Boot operation.

It is not a bad idea to go back and look at the description of the volume parameter in Section 6.5.2. The INIT command is the only way to create or change a volume number.

> CAUTION: Keep in mind that the INIT command totally erases a diskette. It doesn't have a conscience.

If you put a valuable diskette, with a lot of programs, in a drive and type INIT, the DOS will go right ahead and erase the whole damn thing and there goes one valuable diskette! This is potentially your most dangerous command and you should be aware of it.

In summary, we have now described how you can "straighten up" your diskette files. These seven commands are best described as housekeeping operations. All seven of these commands have the general form of

```
COMMAND X,Ss,Dd,Vv
```

where X is a file name and S, D, and V are the optional Slot, Drive, and Volume Numbers. These commands are the most frequently used commands and it is important that you become very familiar with them.

6.5.5 Program Files

We have generally described how to LOAD, SAVE, and RUN the Basic program files. In this section we will describe the rest of the details pertaining to these commands.

SAVE X This DOS command is used to move a Basic program from the computer memory to a diskette file called X. The file type (Integer or Applesoft) is defined by the program. If the file X does not exist on the diskette, the DOS will create a file called X and store the program in that file. If the file X exists, the DOS will automatically replace the contents of file X with this new memory image and will not inform the operator that file X already existed. (This can be dangerous because you can inadvertently overwrite a program file that you did not want to destroy. Use of the LOCK command could prevent this problem.)

If you attempt to SAVE a program that is larger than the available disk space, the DOS will respond with the error message DISK FULL ERROR. To recover from this, you need to change diskettes to one with enough space, or delete a file to make more space available on the current diskette. Either of these options will work.

You don't need to worry about the program you have in memory. The SAVE command does not alter the contents of the program memory in any way. As a matter of fact it is a good idea to SAVE your programs periodically. To do this just type SAVE X and then go back to work on your program.

The SAVE command can have the optional Slot, Drive, and Volume Number parameters.

LOAD X This DOS command is used to move the contents of the program file X from the diskette to the computer memory. The file type must be an Integer or Applesoft file. If you attempt to

LOAD a Binary or Text File, the DOS will respond with the error message FILE TYPE MISMATCH. If you attempt to LOAD an Integer file with only Applesoft in your computer (or an Applesoft file with only Integer present), the DOS will respond with a LANGUAGE NOT AVAILABLE error. If you have a 16K RAM card and an Apple DOS 3.3 System Master, you can have both Applesoft and Integer in your computer at the same time. (One of the languages is in the ROM's and the other language is in the RAM card.) If you Boot a System Master (or a copy of a System Master), the greeting program will put the appropriate language into the 16K RAM card. With both languages present, the DOS will automatically select the language specified by the file type.

The LOAD command can have the optional Slot, Drive, and Volume Number parameters.

RUN This command is the same as a LOAD command, but after the program file has been LOADed into the computer memory, it will automatically be RUN. All of the restrictions for the LOAD command also apply to the RUN command. A RUN command will clear all of the program memory and then LOAD the new program.

The RUN command may have the optional Slot, Drive, and Volume Number parameters.

CHAIN This is a new DOS command that we have not previously mentioned. This command is virtually identical to the RUN command. It is used only with Integer Basic programs. The major difference between RUN and CHAIN is that this command does not clear the variables before loading the new program. This means that you can write a very large Integer Basic program in sections and then link these sections with CHAIN commands. By linking with CHAIN commands instead of RUN commands, the programs can use the same variables. (Remember, RUN clears all of the variables and CHAIN does not.) The format of this command is the same as the LOAD or RUN commands. That is,

```
CHAIN X
```

where X is the file name. The same restrictions that apply to the LOAD and RUN commands also apply to the CHAIN command.

The CHAIN command can use the optional Slot, Drive, and Volume Number parameters.

To CHAIN programs in Applesoft that do not pass variables is actually quite easy. You can LOAD a new Applesoft program by issuing a DOS RUN X command as the last statement in a program. This is described in Section 6.5.10.

To pass parameters between Applesoft programs, you will need a Binary program called CHAIN. This program comes on the Apple DOS 3.3 System Master. To use CHAIN, you must insert the following commands into your Basic program:

```
5000 PRINT CHR$(4);"BLOAD CHAIN, A520"
5005 CALL 520"X"
```

These commands should be at the end of each program that is passing parameters to the next program. The two line numbers can be any valid line numbers, but the two command lines must be sequential. The first line will "BLOAD" the Binary program CHAIN into the computer. The file CHAIN must be on the same diskette as the new Applesoft program. The second command line actually chains

the two programs. The new Applesoft file name X must be in quotes and must start immediately after the 520 without a space.

To move the CHAIN program to various diskettes, you should use the FID program. (FID is described in Section 6.5.19.)

In summary, we have now described what you need to know about Program Files. You can LOAD, SAVE, RUN, or CHAIN these files. All of these commands have the form:

```
COMMAND X,Ss,Dd,Vv
```

where X is a file name and S, D, and V are the Slot, Drive, and Volume Number parameters.

6.5.6 Binary Files

These types of files usually involve machine language programs. If you are strictly a Basic programmer, then this area should be only of general interest to you. If, however, you are an assembly language programmer, or you wish to become one, then this section is very important to you. Just like Basic, the DOS does not help you write (create) a machine language program. There are several programs available that can help you create a machine language program. One of these is EDASM on the DOS Tool Kit from Apple. This is actually two programs in one; it is both a text editor and an assembler. The creation of assembly language programs is beyond the scope of this manual. If you intend to write more than just small binary programs, then you will need thorough understanding of the Apple II memory organization. The publication "Beneath Apple DOS" is the best available document for this purpose. It is far superior to the Apple DOS Manual in this area and we strongly recommend that you obtain a copy.

There are three commands that deal with Binary Files. They are BLOAD, BSAVE, and BRUN. These three commands are very similar to their Basic Program File counterparts LOAD, SAVE, and RUN. All of the DOS errors (FILE TYPE MISMATCH, FILE LOCKED, FILE NOT FOUND, and VOLUME MISMATCH) pertain to Binary files as well as Basic files.

BSAVE X, A$1000,L200 This DOS command is used to move a specific area of memory to the file X. The memory area must be explicitly stated in the command. The A parameter must specify the starting memory address. The address must be in the range of 0 to 65,535 and can be stated in decimal or hex, where $ designates a hex address. The L parameter is used to designate the length of the memory area to be saved. It must be in the range of 1 to 32,767. Again, the length can be stated in decimal or hex, where $ designates a hex value.

Both the A and L parameters must be stated or a DOS SYNTAX ERROR will result. If either the A or L parameter is not within its previously stated range, a DOS RANGE ERROR will result.

The BSAVE command can also contain the optional Slot, Drive, and Volume Number parameters.

BLOAD X, A$1000 This command is used to move the Binary File X from the diskette to the computer memory. The A parameter is optional. If it is stated, the file will be loaded into memory starting at the specified address. If the A parameter is not stated, then the file will be loaded at the address stated in the

BSAVE command when the file was created. The number of bytes loaded by this command is always the same as the length parameter stated during the BSAVE command that created the file. This command can have the optional Slot, Drive, and Volume Number parameters. It cannot have an L parameter.

> CAUTION: This command can destroy DOS.

If you BLOAD a file that overlays the DOS, many strange and unexplained things can happen. Therefore, if you just start playing around with various Binary Files that you don't understand -- Good Luck!

BRUN X, A$1000 Just like the RUN command, the BRUN command first BLOAD's a Binary File and then executes it. All of the things we explained about BLOAD also pertain to BRUN. The optional A parameter becomes both the load address and the execute address. If the A parameter is omitted, the BRUN command will default to the address used with the BSAVE command that created the file. The BRUN command can use the optional Slot, Drive, and Volume Number parameters.

> CAUTION: Since this command does both a BLOAD and then executes the machine code stored in the file, it is twice as dangerous as the BLOAD command.

Again, if you don't know what you are doing and you play around with the BRUN command -- Extra Good Luck!

In summary, these commands require an understanding of machine or assembly language programming. If you understand these programs, the operation of these three commands should be very obvious to you. If you don't understand these types of programs, then you probably won't need these DOS commands unless you are specifically instructed to BRUN a certain file. In that case you merely need to type

```
BRUN X
```

which will load the Binary File X and then execute it.

6.5.7 DOS Error Messages

Up until now we have assumed that when you issue a command to the DOS, it will execute that command without any errors. That would be nice, but not too realistic. When DOS errors are encountered, they will usually terminate the current command (and any program that is running) and display a message. These messages are slightly different than an Applesoft or Integer message. Applesoft errors are preceded by ??? and Integer errors are preceded by ***. DOS errors are not preceded by any special characters. All of the DOS error messages and their code numbers are listed below. (These numbers may be useful to more advanced programmers.)

Code	Message	Code	Message
1	LANGUAGE NOT AVAILABLE	9	DISK FULL
2,3	RANGE ERROR	10	FILE LOCKED
4	WRITE PROTECTED	11	SYNTAX ERROR
5	END OF DATA	12	NO BUFFERS AVAILABLE
6	FILE NOT FOUND	13	FILE TYPE MISMATCH
7	VOLUME MISMATCH	14	PROGRAM TOO LARGE
8	I/O ERROR	15	NOT DIRECT COMMAND

LANGUAGE NOT AVAILABLE There are two types of Basic files. If you attempt to access a file without the proper Basic in your computer, this error will result. The DOS commands FP and INT can be used to change the language available in your computer. Section 6.5.17 describes these commands.

RANGE ERROR This error occurs when a command parameter is either too large or too small. The proper ranges for each parameter are:

Parameter	Letter	Min	Max
Slot	S	1	7
Drive	D	1	2
Volume	V	0	254
Byte	B	0	32K
Relative Field	R	0	32K
Absolute Field	R	0	32K
Record Length	L	1	32K
Record Number	R	0	32K
Starting Address	A	0	64K
Number of Bytes	L	1	32K

Command	Min	Max
PR#	0	7
IN#	0	7
MAXFILES	1	16

32K is defined as 32,767
64K is defined as 65,535

All DOS parameters must fall into the range of 0 to 65,535. If a nuumber is outside this range, a DOS SYNTAX ERROR will result.

WRITE PROTECTED This error occurs when the DOS attempts to write to a drive that has the diskette write protect notch covered. See Section 3, Familiarization, for more information.

END OF DATA This error can only occur when a program is reading a Text File. For sequential text files, this error means the program has read (using GET or INPUT) past the last character in the file, the B (byte) parameter has specified a byte beyond the last character stored in the file, or the R parameter for a POSITION command has positioned past the last field in the file.

With Random-Access Files, this error can mean one of two conditions. First, the DOS is attempting to read (with GET or INPUT) a portion of a record which has never been written. This can happen by issuing too many GET or INPUT commands for a specific record or by using the B parameter with a value greater than the number of characters that have been stored in the record.

Second, the DOS is attempting to read a record beyond the end of the file. This can occur by specifying an R parameter with a value that moves the record pointer past the last record in the file.

Finally, this error will occur if the R parameter with the EXEC command specifies the second field past the end of the file.

FILE NOT FOUND This error is caused by attempting to access a specific file and that file is not on the addressed diskette. The commands SAVE, BSAVE, INIT, and OPEN can create files and, therefore, will never respond with a FILE NOT FOUND error.

VOLUME MISMATCH If a DOS command specifies a non-zero V parameter and the Volume Number of the addressed diskette is not the same as the V parameter, a VOLUME MISMATCH occurs.

I/O ERROR This is a message from the RWTS that indicates the last attempted data transfer between the computer and the disk drive was in error. This can occur because the RWTS was unsuccessful in locating the proper address field or because the internal check characters for a data sector were incorrect for that data sector.

DISK FULL Any DOS command that adds information to the diskette can encounter this error. This error occurs when the DOS File Manager needs another sector for a file and there are no more sectors left on the diskettes. When this error occurs, the file which needed the next sector is incomplete and the operator must make room on this (or another) diskette to save all of the information.

FILE LOCKED Attempting to Delete or Write to a file which is locked will result in a FILE LOCKED message.

SYNTAX ERROR This error occurs when the DOS encounters a DOS command with a bad file name, a bad parameter symbol, a missing mandatory parameter, or a missing or incorrect separator.

NO BUFFERS AVAILABLE This error occurs when a program is using Text Files and the program attempts to OPEN more files than the MAXFILES command has allocated. Upon booting, the DOS defaults to three open file buffers. The MAXFILES command is used to change the number of available OPEN files. Section 6.5.17 describes MAXFILES.

FILE TYPE MISMATCH This error occurs when a DOS command attempts to use a file whose file type does not agree with the command. Section 6.5.3, entitled "File Types," describes which file types can be used with each command.

PROGRAM TOO LARGE As the name implies, this error occurs when the DOS is attempting to load a program into memory and finds insufficient memory available. Usually this is caused by trying to load a 48K program on a 32K computer. It can also be caused by the operator or a program leaving HIMEM set too low. FP or INT (see Section 6.5.17) will reset HIMEM.

NOT DIRECT COMMAND This error occurs when the operator attempts to use a Text File command (OPEN, READ, WRITE, APPEND, or POSITION) in the immediate execution mode.

6.5.8 Program File Commands

So far we have discussed the following DOS commands:

CATALOG	Ss,Dd,Vv	VERIFY	X,Ss,Dd,Vv	CHAIN	X,Ss,Dd,Vv
DELETE	X,Ss,Dd,Vv	INIT	X,Ss,Dd,Vv	BSAVE	X,Aa,Ll,Ss,Dd,Vv
LOCK	X,Ss,Dd,Vv	SAVE	X,Ss,Dd,Vv	BLOAD	X,Aa,Ss,Dd,Vv
UNLOCK	X,Ss,Dd,Vv	LOAD	X,Ss,Dd,Vv	BRUN	X,Aa,Ss,Dd,Vv
RENAME	X,Y,Ss,Dd,Vv	RUN	X,Ss,Dd,Vv		

where X is always the file name; S, D, and V are the optional Slot, Drive, and Volume Numbers; and Y is the new file name for a RENAME DOS command. A is the starting address parameter associated with Binary File commands, and L is the length parameter associated with the BSAVE command.

At this point you should understand the purpose for each of these commands. As you can see, these commands always take the form of: command word, file name, and the optional Slot, Drive, and Volume Number parameters. All of these commands will start a disk access, even if that just involves reading the Directory. This means that you could encounter almost any of the DOS errors described in the previous section. With the exception of I/O ERROR, these DOS errors should not alarm you. They are merely messages from the DOS that it cannot complete your command. This may be because you entered something incorrectly or because one of the built-in checks in the DOS is trying to prevent you from making a mistake. In either case, you should stop for a minute and evaluate the DOS error message in respect to the command you have just issued. In almost all cases it should be obvious how you can correct the error.

To become familiar with these DOS commands, you must start using them. We have presented all of the information you need to know about these commands; now it's up to you to sit down at your computer and try each of the commands, one at a time. Don't be alarmed if you encounter several DOS errors; these should be viewed as warnings, not catastrophies.

The next section is intended to lead you through a few examples of how to use Apple DOS. We call it a lesson "guide" because we expect you to do most of the work.

6.5.9 Your Program Command Lesson Guide

This section will take you through some "hands-on" experience with your new disks and DOS 3.3. There are some very important prerequisites for this course. You must have already read through the Apple II Reference Manual and the Basic Programming Reference Manual. If you have not read these manuals you are trying to go too fast. Your disk drives are considered peripheral devices and are an extension of your computer. Trying to learn to use a peripheral device before you know how to operate the computer is putting "the cart before the horse."

If you can turn on your computer, get into Basic, write a simple program, list that program, and run it, you are ready to start learning how to use DOS. If you don't feel you can handle these steps, you should go back to the Apple II Reference Manual and the Basic Programming Reference Manual.

The first thing we must do to get started is to get the DOS loaded into the computer. Section 6.5.1 explained how to Boot DOS. If you have a problem, a reveiw of Section 4 on installation may solve it.

In the following steps you will be asked to WRITE A PROGRAM. Since we are concerned with the operation of the DOS, rather than the Basic, these programs will be very simple. For instance, if Step 3 requests that you WRITE PRG7 we expect you to enter:


```
NEW
10 PRINT "PROGRAM 7"
20 END
LIST
RUN
```

This will clear memory (NEW) and create a program that prints the word PROGRAM 7. LIST will list the program for you to check it, and RUN will execute the program to make sure it is operational.

If another step requests you WRITE PRG2, the only difference is the PRINT statement. You should change it to:

```
10 PRINT "PROGRAM 2"
```

This section is a lesson "guide." By that we mean we expect you to take your time and think about each step. We expect you may encounter a few DOS errors. This is a normal part of using your computer; don't be alarmed. Section 6.5.7 describes each of the DOS errors and the probable cause for that error.

With each of the following steps, we have included an action or a DOS command. The DOS commands are enclosed in quotation marks. These commands should be typed just as they are shown, but without the quotation marks.

The actions are self-explanatory. As we have already stated, the action WRITE PRG1 tells you to write a simple program, list it, and run it.

STEP 1: Boot the DOS

This is explained in Section 6.5.1. After DOS is Booted, you should have the standard Basic prompt character.

STEP 2: "CATALOG"

This DOS command will list the names of the files on the diskette. If you get the message SYNTAX ERROR preceded by *** or ???, you do not have DOS connected to Basic. You must re-Boot and try again.

STEP 3: WRITE PRG1

We now want you to write your first program. Remember the proper sequence is:

```
NEW
10 PRINT "PROGRAM 1"
20 END
LIST
RUN
```

STEP 4: "INIT PRG1,V1"

CAUTION: Make sure you removed your DOS diskette and inserted a blank diskette. You should go back and review the description of the INIT command.

STEP 5: "CATALOG"

This diskette should be Volume 1, with PRG1 as the only file.

STEP 6: WRITE PRG2

```
Again:  NEW
        10 PRINT "PROGRAM 2"
        20 END
        LIST
        RUN
```

STEP 7: "SAVE PRG2"

This command will move your second program to the file name PRG2.

STEP 8: "CATALOG"

The diskette should now have 2 files: PRG1 and PRG2.

STEP 9: "LOAD PRG1"

Use LIST to verify PRG1 loaded.

STEP 10: "RUN PRG2"

The message PROGRAM 2 will verify that DOS has loaded a new program and executed it.

STEP 11: WRITE PRG3

STEP 12: "SAVE PRG3"

STEP 13: "CATALOG"

You now have three files on the diskette: PRG1, PRG2, and PRG3.

STEP 14: "DELETE PRG2"

STEP 15: "CATALOG"

Now you should have only two files left: PRG1 and PRG3.

STEP 16: "LOCK PRG3"

STEP 17: "DELETE PRG3"

You will get the DOS Error message FILE LOCKED.

STEP 18: "RUN PRG3"

This command will execute, indicating you <u>can</u> LOAD or RUN a locked file.

STEP 19: "SAVE PRG3"

You will get the DOS Error FILE LOCKED.

STEP 20: "CATALOG"

Notice PRG3 has the locked indicator set.

STEP 21: "UNLOCK PRG3"

STEP 22: "CATALOG"

Notice PRG3 is now unlocked.

STEP 23: "RENAME PRG3,PRG4"

STEP 24: "CATALOG"

You still have only two files, but they are now PRG1 and PRG4.

STEP 25: "VERIFY PRG4"

STEP 26: "LOAD NO PROGRAM"

You will get the DOS Error message FILE NOT FOUND. We never had a file NO PROGRAM.

STEP 27: "DELETE PRG1"

STEP 28: "IN#6"

This will re-Boot DOS. But because you deleted PRG1, after DOS Boots, it will display FILE NOT FOUND. You should not delete the greeting program.

STEP 29: "RENAME PRG4,PRG1"

We now have a greeting program called PRG1.

STEP 30: "CATALOG"

Notice we still have only 1 file: PRG1.

STEP 31: "LOAD PRG1,V200"

You will get the DOS Error message VOLUME MISMATCH. Remember INIT made this Volume 1.

6.5.10 <u>Using DOS Commands from Basic</u>

So far we have introduced you to eleven DOS commands and have showed you how to use these commands as direct commands. A direct command is one which you issue from outside a program. There are times when you want to issue a DOS command from inside a program. To do this you use a PRINT statement with the following format:

```
50 PRINT D$;"CATALOG"
```

The number 50 is the program line number; and, as you know, this can be any legal program line number. PRINT is a standard Basic command. The character string D$; is the important operator in this function, and we will have more to say about it shortly. The actual DOS command is enclosed in quotes. This part of the command is the same as using DOS commands as direct commands. For instance, a RUN command from Basic has the same format as a direct RUN command:

```
60 PRINT D$;"RUN X,Ss,Dd,Vv"
```

where X, S, D, and V are the normal command parameters.

The character string D$ is a single-character string of Control-D. To create this string in Applesoft, you should use the following:

```
20 D$=CHR$(4):REM CHR$(4)IS A CTRL-D
```

The ASCII code for a CONTROL-D is a 4.

To create this string from Integer, you should type:

```
50 D$="":REM YOU CAN'T SEE THE CTRL-D
```

With this format you must put a Control-D between the quotes. Control-D is input by holding down the Control key and pressing the character D. Control characters do not display on your monitor screen and, therefore, you won't see anything between the quotes.

> Special Note: Using the right arrow to copy a Basic program line will erase the Control-D character. The right arrow does not copy control characters.

Now that we have made such a big deal out of this Control-D character, we hope you realize that it is a very important parameter. The Control-D informs the DOS that this PRINT command is not to go to the monitor screen but is to be executed as a DOS command. If the Control-D is omitted, the DOS will not execute the command. Instead, the command is treated as a message and PRINTed on the monitor screen.

Let's assume you want your greeting program to do a CATALOG right after you Boot. The following program will work with either Integer or Applesoft.

```
10 D$="":REM DON'T FORGET THE CTRL-D
20 PRINT "THIS DISKETTE WAS CREATED ON 1/1/81"
30 PRINT D$;"CATALOG"
40 END
```

Some final cautions about DOS commands from within a program. You can only put one DOS command on each program line. The previous PRINT command must have ended with a RETURN. This means you cannot use the semicolon (;) option to terminate a PRINT statement just before a PRINT statement with a DOS command.

6.5.11 Introduction to Text Files

One of the more attractive features of DOS is its ability to store and retrieve information other than programs. We have already defined the "File Types" that hold programs (A = Applesoft; I = Integer; and B = Binary Files). The file type that holds non-program information is called a Text File. A Text File can have any ASCII character, including all upper and lower case letters, numbers, special characters, and control characters. This means if the keyboard or a Basic program can generate the character, then it can be stored in a Text File. However, DOS will only allow you to access a Text File from inside a Basic program. This means there are no DOS commands that will allow you to put characters directly from the keyboard into a Text File or take characters directly from a Text File and put them on the monitor. You must write a Basic program to move characters into or out of a Text File.

The DOS commands that control Text Files are:

OPEN	WRITE
CLOSE	POSITION
READ	APPEND

Suppose you wanted to store a list of names, addresses, and telephone numbers in a file called NAMES. We know we have to write a program to do this. Your Basic Programming Manual (Integer or Applesoft) explains how to get characters from the keyboard and store them in string variables. In this manner we could collect one name, address, and telephone number in three string variables and have them ready to store in our Text File. The normal sequence to store these strings in a Text File is to OPEN a file called NAMES; then issue the WRITE command specifying the file NAMES; next, move the three strings containing the name, address, and telephone number to the file using a standard PRINT statement; and finally when you are done, CLOSE the file.

Now let's review each of these steps in more detail. The first step, OPEN, does exactly that. As we explained in the DOS File Manager overview, all information stored on the diskette must be stored in a file. The OPEN command informs the File Manager that we are about to use a Text File; in this case called NAMES. If the file does not exist, the File Manager will create the new file and allocate a Track and Sector List. If the file does exist, the File Manager will get the pointers to the existing Track and Sector List. In either case, the File Manager is now ready to start transferring data into, or in some cases out of, the OPENed file.

The second step, WRITE, informs the DOS that we intend to transfer data to the Text File. This command does not transfer any data to the Text File but merely informs the DOS that the data from the next PRINT statement should be placed in the Text File and not on the monitor screen. We will have more to say about the PRINT statement shortly. The DOS supports more than one OPEN Text File at a time. In fact, DOS supports up to 16 OPEN Text Files.

Since the File Manager must simultaneously support all of these OPEN files, it must be informed which file is currently active. Therefore, the WRITE command is used to specify both the data direction and the current file.

The PRINT statement is a standard Basic command. As you remember from the Basic Reference Manuals, this command allows you to display character strings, numeric constants, or variables on the computer monitor. When you are using Text Files, this command has a new function. By issuing a DOS WRITE command just before the PRINT statement, the output data from the PRINT statement is placed into a Text File instead of being displayed on the monitor. This means anything you could output to the monitor can instead be placed into a Text File and saved for future reference.

Finally, in our brief example we ended up with a DOS CLOSE command. This command informs the File Manager that the program is finished using this Text File. After receiving the CLOSE command, the File Manager will update the file's Track and Sector List sector. If you fail to CLOSE a file, the Track and Sector List will not be updated and some of the information you have just stored in the file will be lost.

This brief example has introduced you to the concept of writing data into a Text File. Now let's look at reading the data back

out of the Text File. You must first OPEN the Text File and then issue the DOS command READ, followed by a Basic GET or INPUT statement; and, finally, when you are through, CLOSE the Text File.

In Step 1, the OPEN command performs the same function when reading a file as when you are writing. It informs the File Manager to get the pointers for this Text File. As we explained with the WRITE operation, DOS supports up to 16 OPEN Text Files. In Step 2, the DOS READ command informs the DOS that the next GET or INPUT statement is directed at a Text File -- not the keyboard. (More on the GET or INPUT statement in the next paragraph.) Like the DOS WRITE command, the READ command performs two functions. First, it temporarily redirects the Basic GET and INPUT statements to access a Text File for its data instead of the keyboard. Second, it informs the DOS which of the 16 possible Text Files is the currently active file.

The Basic Program Reference Manuals explained the operation of the GET and INPUT statements as they pertain to the keyboard. When you are using a Text File and precede a GET or INPUT statement with a DOS READ command, the Text File replaces the keyboard as the source for the data.

Whenever you are using a Text File, you must CLOSE the file when you are finished.

Since these commands must be issued from within a Basic program, the program line will always have the format:

```
100 PRINT D$;"OPEN X,S6,D1,V254"
```

The line number (100) may be any valid Basic program line number. The PRINT statement immediately followed by a Control-D informs the DOS that this is a DOS command, not a message. The actual DOS command is then enclosed in quotes. The parameters for Text File commands are slightly different than the DOS Housekeeping commands we previously explained. In the following sections, we will explicitly define which parameters may be used with each command. Finally, since these are DOS commands, you cannot put more than one command per line.

6.5.12 Sequential and Random-Access Text Files

In the previous section we explained how to store a small amount of information in a Text File. The real value of a Text File is its ability to store a large amount of data. When you are handling large amounts of data, the organization of the data becomes very important. Data are organized into fields and records. We call a field a group of characters ending with a carriage return. We call a record a group of fields. In the last section we gave an example of a Text File containing a name, address, and telephone number. If we were to further define that example, we could say we had one record with three fields: a name field, an address field, and a telephone number field.

We could then add more records to the file, where each record has its own name, address, and telephone number fields. As we add more records we need to understand how these records are stored in a Text File. There are two different ways the records can be stored: one is called Sequential and the other is Random-Access. A Sequential File is the simplest and most efficient type in terms of storage. With this storage method, each character is placed in

the next available location in the file. This means there is no unused space in the file. When you want to find something in this file you usually start at the beginning and search "sequentially" until you find it. This type of file is very similar to a tape file.

The second type, Random-Access File, is capable of maintaining more individuality between records. With this file type, the program must define the exact length of a record. Usually the defined record length is longer than the actual space needed to store all of the fields. This leaves some unused space between records. Going back to our example of name, address, and telephone number, we can set up a Random-Access File to illustrate this point. Let's assume we allocate 30 characters for the name, 50 characters for the address, and 10 characters for the telephone number. This means all of the records in our file are a fixed length of 90 characters each (30+50+10=90). Within each of these records we can have three variable length fields called name, address, and telephone number. For instance, we could have a record with a 15-character name field, a 34-character address field, and a 10-character telephone number field, for a total of 59 characters. The remaining 31 characters for this record would not be used; however, DOS would leave room for these 31 characters in the record.

At this point it is probably not obvious why there are two types of Text Files. We do know that if we had 100 records with an average of 53 characters each, we could put them in a Sequential Text File. This file would have 5300 bytes (100x53=5300). If we took the same 100 records and put them in a Random-Access File and used a file length longer than the longest record, say 65 characters, we would have a 6500-byte file (100x65=6500). We also know that the Random-Access File would have some unused space between most of the 100 records in the file.

From this example we see that Sequential Files can store the same amount of information in less space. However, if we want to add one character to a Sequential File, we could have a pretty big task on our hands. With a Sequential File, we don't have any unused space; so to make room for our new character, we have to move all of the characters over one space. On the other hand, with a Random-Access File, we left some unused space between records. If we add a character to a Random-Access File, the change will affect only one record and the effects of the change won't ripple down through all of the rest of the records.

Now, we know that there are two types of Text Files -- Sequential and Random-Access. Even if we could decide which of these two types is best suited for a given application, we would still be at a loss as to how to create and manipulate these files. When we discussed Program Files, we did not teach you how to write a Basic program. DOS merely provides you with a way to store or retrieve the program once you have created it. The same thing is true of Text Files. You, as a programmer, must decide what you want to store in a Text File and how you want to organize the data. Once you have made these decisions, the DOS provides you with a few simple commands that permit you to store and retrieve information from Text Files.

6.5.13 Getting Started with Text Files

Before we get down to the details of Sequential and Random-Access Files, let's discuss a general approach toward using a Text File.

When you were learning the language Basic, there were two things you had to learn to deal with: first was Approach and second was Implementation. The Approach requires analyzing the given data, the variables, and the objectives and devising a method to solve a problem. Implementation is then taking this "approach" and selecting the proper sequence of Basic statements to achieve the desired results.

We must follow the same steps when using a Text File. In the Approach phase, we must take the time to decide what information we want to put in the file. It is best if this is done with pencil and paper and not just in your head. Going back to our name, address, and telephone number example, we might develop the following list:

NAME	FIRST
	MIDDLE
	LAST
ADDRESS	STREET NUMBER
	STREET NAME
	APARTMENT, SUITE, UNIT
	CITY
	STATE
	COUNTY
	ZIP CODE
TELEPHONE NUMBER	AREA CODE
	NUMBER

This is a very general example, and anyone can relate to a file with names, addresses, and telephone numbers. But, from your own experience, is this enough information? What else would you want in each record?

Each person approaching this problem may have some other unique information which only pertains to their circumstances. This points out the importance of the Approach phase. It is impossible to begin to write a Basic program to create and manipulate a Text File if you don't know what you are going to put in that file.

Once you know "what" is in the file, the second step is to organize that data. This could mean that you would take the information and split it up into several files. Then the file (or files) must be organized into fields and records. You must decide the order of the fields. You should give some thought to the length of the fields and records. How many records are going to be in the file? How big is the total file? Do you need to access and update individual records? There are as many questions and details in the Approach phase as there are applications. Only you as a programmer, with a specific problem in mind, can organize the Text Files.

The second step is Implementation. This step requires the ability to write a Basic program (or programs) that will create and use the Text Files. This is a key question: Do you know how to write a Basic program? If the answer is no, then you won't be able to use the DOS Text File commands.

Don't be discouraged. Basic is not that hard to learn. But until you can collect information from the keyboard, process it, and display it on the monitor, you are not ready to jump into the DOS Text File commands.

If you feel that, with the aid of your Basic Reference Manuals, you can get information from the keyboard and display it on the monitor, then you are ready to use the DOS Text File commands.

Using the Text Files is as simple as the GET, INPUT, and PRINT statements in Basic. In fact, using these statements is the only way you can access a Text File. There is, however, another factor which makes things a bit more complicated. The factor is called "Position." As you know, a Text File is a big collection of characters, grouped into fields which are in turn grouped into records. We have talked about two ways to organize these characters, fields, and records; the first is called Sequential and the second is called Random-Access. Positioning is the method whereby we can direct a GET, INPUT, or PRINT statement to a specific point in the Text File. Think about this for a second. This is a very important idea.

Let's go back to our example of name, address, and telephone number and assume we have already created a file with 100 records. Let's further assume we want to examine the 73rd record of that file. One way to get to the 73rd record is to start at the beginning and let the Basic program count its way through the records. This will work just fine; but it requires a little "smarter" Basic program, and it probably won't be very fast, especially if you start at the beginning each time you want the next record.

The DOS Text File commands have a positioning capability, and it is this positioning capability which is one of the fundamental differences between Sequential and Random-Access files. To discuss positioning, we have to introduce the term "position-in-the-file pointer."

All GET, INPUT, or PRINT statements initiate data transfers starting at the position-in-the-file pointer. For instance, let's say we have a 10,000-byte file and the position-in-the-file pointer is located at the 3456th byte. A GET statement would fetch this byte from the file and increment the position-in-the-file pointer. An INPUT statement would fetch the next n consecutive bytes until it encountered a Return character. For each character transferred from the file, the position-in-the-file pointer would be incremented by one. A PRINT statement will place its output data into the file starting at the position-in-the-file pointer. For each character placed in the file, the position-in-the-file pointer is incremented by one.

The DOS POSITION command and positioning parameters allow a program to automatically move the position-in-the-file pointer past any number of characters without reading or writing to these locations. The details for moving the position-in-the-file pointer differ for Sequential and Random-Access Files. These details will be discussed in the next two sections.

6.5.14 Sequential Files

By now you should know the organization of a Sequential File. This section will take you through the details of using the six commands associated with these files. Section 6.5.15 will take you through Random-Access Files. Although both file types have commands with the same names, the operation and parameters for these commands are dependent on the type of file being accessed. It is important that you know the differences between the commands, especially if you intend to use both file types.

In this section we will introduce a new term called "Memory Buffer Work Area." For each OPEN file, the DOS must have a 595-byte work area. 256 bytes are used to buffer the Track and Sector List for the file. Another 256 bytes are used to buffer one data sector for the file. The remaining 83 bytes are used as pointers and scratch work areas by the File Manager. The DOS command MAXFILES (this command will be discussed in Section 6.5.17) is used to change the number of 595-byte buffers available to the File Manager. DOS normally allocates three 595-byte buffers, which means only three files may be opened simultaneously.

The six commands discussed in this section are:

OPEN	X,Ss,Dd,Vv
CLOSE	X
READ	X,Bb
WRITE	X,Bb
POSITION	X,Rf
APPEND	X,Ss,Dd,Vv

OPEN X,Ss,Dd,Vv This command informs the File Manager to prepare the Text File X for processing. The subsequent processing may be either reading or writing. The File Manager searches the diskette directory for the file X, and if it doesn't find X, it creates a file called X. The File Manager then allocates a 595-byte memory buffer work area for the file. If the file already exists, this command will set up the memory buffer work area. If the file was already OPEN, this command CLOSEs the file and then reOPENs the file. The OPEN command will always move the position-in-the-file pointer to the first byte of the file.

CLOSE X This DOS command is used to inform the File Manager that the Basic program has completed transferring data to or from the file X. The File Manager then moves certain information from the 595-byte memory buffer work area to the diskette. If this information is not moved from the memory buffer work area to the diskette, it will be lost.

The CLOSE command has several unique features. First, it does not need to have a file name. If the DOS encounters a CLOSE command without a file name, it CLOSEs all of the OPEN files. Second, the CLOSE command does not require a Slot, Drive, or Volume Number parameter. Using the S, D, or V parameters with the CLOSE command will cause a SYNTAX ERROR. Third, this command can be issued as a direct command; in other words, outside a Basic program. If a Basic program terminates with an error, the Text Files for that program may still be OPEN. By issuing the DOS command CLOSE, the File Manager will move the information in the memory buffer work areas to the diskette.

READ X,Bb This DOS command is used by a Basic program to retrieve information from the text file X. The text file X must already be OPENed. This command must have a file name and can have an optional B parameter. (The B parameter will be discussed in a later paragraph.) This command does not use the S, D, or V parameters, and assigning one of these parameters will cause a SYNTAX ERROR. This command causes the Basic GET and INPUT statements to retrieve information from the "current" Text File instead of the keyboard. Since DOS supports up to 16 simultaneously OPENed Text Files, the "current" Text File is defined as the file specified by the READ command immediately preceding the GET or INPUT statement.

Once a READ command has been issued, it will remain active until another DOS command is issued. This means that after a READ command, all GET or INPUT statements are directed to the Text File until the READ command is terminated by another DOS command. (Any DOS command except READ will terminate the READ operation.) A PRINT statement with just a Control-D is considered a null DOS command and will terminate a READ or WRITE operation.

The B parameter is used to move the position-in-the-file pointer forward from its current position. The B value is a relative displacement from 0 to 32,767 bytes. The position-in-the-file pointer can only be moved forward in the file. The only way to move the pointer backward in the file is to reOPEN the file. This will reposition the position-in-the-file pointer to the beginning of the file. Use of the B parameter is an advanced programming technique. Unless you have had considerable experience with programming disk files, you should not be concerned with this parameter. For more information on the use of the B parameter, see the POSITION command description.

WRITE X,Bb This command is used to redirect the data from a PRINT statement into a Text File. The file X must OPEN before the DOS encounters a WRITE command. The DOS parameters S, D, and V cannot be used with this command, or a DOS SYNTAX ERROR will result. Once the WRITE command is issued, it will remain active until a new DOS command is encountered. This means that the data from all of the PRINT statements, following a DOS WRITE command, will be placed into the active Text File until any other DOS command is issued.

The WRITE command can have an optional B parameter just like the READ command. The description of the B parameter with the READ and POSITION commands also pertains to the WRITE command.

POSITION X,Rr This command is used to move the position-in-the-file pointer forward r fields. The definition of a field is a group of characters terminated by a Return character. When using sequential files, the B parameter for READ or WRITE and the R parameter for the POSITION command always position forward and are relative to the current position-in-the-file.

When the DOS encounters a POSITION command for a Sequential File, the File Manager will scan forward in the file, starting at the current position-in-the-file, while counting Return characters. When the File Manager has counted the number of Return characters specified by the R parameter, the position-in-the-file pointer will be one byte past the last Return character.

The R parameter must be in the range of 0 to 32,767. If the R value is 0, the subsequent READ or WRITE command begins in the current field.

The POSITION command, like any other DOS command, will terminate the current READ or WRITE command. Therefore, POSITION commands must be issued before a READ or WRITE command.

We have now introduced two positioning techniques for Sequential Files. They are the B parameter with the READ or WRITE commands and the R parameter with the POSITION command. (These two parameters, R and B, are also used with Random-Access Files, but in a different way.)

The trick to using the positioning capabilities is knowing what byte or record contains the information you want. This gets back

to the organization of data in the file. If you remember, we stressed this point quite heavily in the last section.

With Sequential Files, the R and B parameters are best used in conjunction with each other. The R parameter can move the position-in-the-file pointer to the start of a specific field. Then the B parameter can move the pointer to a specific byte in that field.

APPEND X,Ss,Dd,Vv This command is almost identical to the OPEN command. The major difference is that the position-in-the-file pointer is placed at the end of the file instead of the beginning. The purpose of this command is to provide an easy way to add characters to the end of a Sequential Text File. Normally, an APPEND command is immediately followed by a WRITE command and then some PRINT statements to add characters to the file. A READ or POSITION command after an APPEND command will always cause an END OF DATA DOS error. You should never issue an OPEN command immediately after an APPEND command. This defeats the purpose of the APPEND command because the OPEN command resets the position-in-the-file pointer back to the beginning of the file.

6.5.15 Random-Access Files

In the previous section we discussed Sequential Text Files and the six commands used to process those files. This section will present the four DOS commands that control Random-Access Files. These four commands are:

```
OPEN      X,Ll,Ss,Dd,Vv
CLOSE     X
READ      X,Rr,Bb
WRITE     X,Rr,Bb
```

As you can see, these commands have the same names as the Sequential File commands. However, these commands have different parameters and you should not confuse the two.

The major difference between Sequential and Random-Access Files is that Random-Access Files have fixed-length records. The advantages of fixed-length records are two-fold. First, it permits a greater degree of individuality between records; and second, it provides a fast and easy way to retrieve information from the file.

To understand why Random-Access Files have these advantages, we must take a closer look at a Random-Access record. We know that each record in the file has the same length. Usually the programmer selects a record length longer than the number of characters he would expect to put in the record. This means each record will have a built-in pad between the last character stored in the record and the actual end of the record. This "pad" comes in very handy when we go back to update the contents of a specific record. When we add or delete characters from this record, we will affect only the contents of the one record. Think of this in contrast to a Sequential File where adding or deleting a character will alter the position of all the rest of the characters in the file. This is what we mean by the individuality of the records.

Since each record in the file is the same length, it is easy to compute the starting byte for any record in the file. This means positioning to a specific record is much easier and faster than with a Sequential File.

Now that we have pointed out these advantages, we know that there must be some disadvantages, or there wouldn't be two types of files. The first disadvantage is space. Since all of the characters in each record are not used, Random-Access Files require more storage than Sequential Files. A second disadvantage is that not all types of information can be divided in individual groups. For instance, a document cannot be broken up into convenient fixed-length records. This type of information is best suited for Sequential Files.

We have mentioned the unused space at the end of each record several times, but we have never explained what happens if a program tries to read these unused characters. When a record is created, the DOS fills the unused characters with Nulls. (Null characters are ASCII 00.) Nulls have a special meaning to the DOS; they signify the END OF DATA within a record. For example, if we have a 100-byte fixed-length record with three fields that use a total of 75 characters, we will have 25 Nulls stored at the end of the record. When a Basic program reads this record, it can INPUT the three fields with no problems. If for some reason the program attempts to INPUT a fourth field, the DOS File Manager will encounter a Null character as the first character of this field. This Null character signals the DOS that this area of the record has never been written. The DOS will respond with the DOS Error END OF DATA instead of transferring the Null characters in response to the INPUT statement.

We will now present the four Random-Access DOS commands.

OPEN X,Ll,Ss,Dd,Vv This command performs the same functions as the Sequential File OPEN command. There is a very important difference, however; the L parameter is mandatory for a Random-Access File. The L parameter defines the record length. It must be in the range of 1 to 32,767. Once you have defined the length of a Random-Access record, you cannot change it. Each time you OPEN the file you must specify the same record length. There is no way to determine the record length from Basic and, therefore, it is up to the programmer to keep track of this number.

Both OPEN commands have the following similarities. First, the File Manager will check the directory to see if this file exists and if not, it will create the file. Second, the File Manager will allocate a 595-byte memory buffer work area. Third, the File Manager will pull the files Track and Sector List. Fourth, if the file is already OPEN, DOS will CLOSE the file and then reOPEN it. Finally, OPEN always sets the position of the file pointer to the first byte of the file.

CLOSE X This command is identical to the Sequential File CLOSE command. In review, the CLOSE command is used to inform the DOS that a program is finished accessing a particular file (or files). The DOS will then move the required information from the memory buffer work area to the diskette. Once this information is stored on the diskette, the DOS will de-allocate the 595-byte memory buffer work area.

The CLOSE command can also be used as a direct command. This is important, especially if a program is interrupted by a Control-C or an Error and the Text Files are left OPEN. The CLOSE command without a file name will close all of the OPEN files.

READ X,Rr,Bb This command is very similar to the Sequential File READ command, with the major difference being the addition of the R parameter. The R parameter is used to identify the specific

record to be used with the next GET or INPUT statement. With Random-Access Files, the R parameter is an absolute pointer. For example, if R specifies 5, this always means the 5th record in the file. In contrast, with Sequential Files, R specifies a relative forward position from the current position-in-the-file pointer.

The R parameter will move the position-in-the-file pointer to the first character of the specified record. R must be in the range of 0 to 32,767. If R is omitted, the DOS will default to R0 which moves the position-in-the-file pointer to the first character in the file.

The function of the B parameter here is the same as the Sequential File B parameter. In precedence, the DOS will first position to the specified record (the r value) and then move the position-in-the-file pointer forward b bytes. Subsequent GET or INPUT statements will fetch their characters from this point in the file.

In review, the READ command must specify the file name (X), will usually specify a record number (R), and can optionally specify a byte displacement (B) forward from the first character of the specified record. With the exception of these positioning capabilities, the rest of the functions of the READ are identical to the Sequential File READ command.

WRITE X,Rr,Bb This command is also very similar to its Sequential File counterpart. The major difference is again the addition of the R parameter. The R and B parameters perform the same functions with the WRITE command as we explained with the READ command. The R parameter is used to specify a specific record. The B parameter can then optionally move the position-in-the-file pointer forward into the selected record. The WRITE command must always specify a file name (X), will usually specify a record number (R), and can optionally specify a byte displacement (B). With the exception of the position parameters R and B, the rest of the functions of the WRITE command are the same as the description with the Sequential File WRITE command.

Some final notes on Random-Access files. You must always specify an L parameter with the OPEN command for a Random-Access file. This L parameter (record length) must always be the same each time you reOPEN the file. The READ and WRITE commands should always specify a record number (R). If the R parameter is omitted, the DOS will default to the first record in the file. With Random-Access Files, the R parameter is a specific record number, not a relative displacement from the current position-in-the-file. If the R parameter specifies a record number larger than the last record in the file, the DOS will respond with an END OF DATA error to the next GET or INPUT statement. Likewise, if a B parameter positions past the last character in a record, the DOS will also respond with an END OF DATA error to the next GET or INPUT statements.

With these few "positioning" exceptions, the use of the OPEN, CLOSE, READ, or WRITE commands is the same as we have described in the previous sections.

6.5.16 EXEC Files

When we discussed Sequential Files we stated that you could put anything that you can generate at the keyboard into a file. Suppose we create a Text File that contains DOS commands, Basic statements, or Monitor commands. Then what if we could direct the computer to use this file as the source for its input, instead of

using the keyboard. This means that we would have a Text File that could control the computer. Well, DOS 3.3 does have this feature and it's called an EXEC File. The file itself is a Sequential Text File, but the information in the file is the DOS, Monitor, and Basic commands that you would normally type at the keyboard. This file is created from a Basic program just like any other Sequential File. Once you have created an EXEC File, you use the DOS command EXEC to run the computer.

This new command EXEC is a cross between a RUN command and the DOS Text File commands, OPEN and READ. The format for the EXEC command is:

```
EXEC X,Ss,Dd,Vv
```

When DOS encounters an EXEC command, it OPENs the specified Sequential File and READs the first sector of that file into the 595-byte memory buffer work area. DOS then redirects all of the keyboard requests to this EXEC File. At this point, as far as the computer is concerned, an operator is inputting information at the keyboard.

For instance, an EXEC File could be used to modify a Basic program. To do this you would normally LOAD the program, change the appropriate Basic lines, and then SAVE the program. The following example will change a program called CALC:

```
LOAD CALC
10 A=1024
25 C=A*B
RENAME CALC,OLDCALC
SAVE CALC
```

This EXEC File would LOAD the program CALC, then change lines 10 and 25 of that program. It would then RENAME the original CALC program to OLDCALC and, finally, SAVE the modified version of the program under the original name.

Notice that the commands in the Text File are in the same format as you would "type" them at the keyboard. These commands are not preceded by any special characters (such as Control-D) or enclosed in quotes. In addition, the Text File commands do not occupy any of the program memory space. The DOS uses the 595-byte memory buffer work area to hold the EXEC File data. If the EXEC file is larger than one sector, the DOS will automatically fetch the next sector as it is needed. For this reason you should never remove the diskette containing the EXEC File. Once the DOS has OPENed a Text File, it uses the Track and Sector List it has stored in the memory buffer work area to fetch the next sector. If you remove the diskette containing the EXEC File and replace it with a new diskette, DOS will blindly fetch the next sector from the new diskette. This will usually be a disaster.

In summary, an EXEC File is merely a Sequential File. The data in this Sequential File can be any valid computer commands that you would normally enter at the keyboard. Once you have created the file, you use the DOS command EXEC to begin executing the commands you have stored in the file.

You can RUN programs from an EXEC file. While the program is executing, the EXEC File remains OPEN and patiently waits for the program to END. When the program has ended, the next command is fetched from the EXEC File. A word of caution about running programs from an EXEC File -- once an EXEC File is OPEN, all keyboard

inputs will come from the EXEC File. This can be an advantage or a disaster. If you have taken this into account, no problem; but if you haven't, you could get some very strange results.

Suppose we have a program listing, line numbers and all, in a Sequential File. (We will tell you how to do this in a minute.) If we EXEC this file, the numbered program lines will be put into the program memory. This can be very useful for adding commonly used Basic subroutines to a new program. It can also be used to convert from Integer to Applesoft, or vice-versa. You know that not all of the Basic statements will directly convert, but it's still better than rekeying the whole program.

To move a program, or a portion of a program, to a Text File, you should add the following lines to the end of that program:

```
5000 D$="":REM DON'T FORGET CTRL-D
5010 PRINT D$;"OPEN LISTFILE"
5020 PRINT D$;"WRITE LISTFILE"
5030 POKE 33,30
5040 LIST 10,4999
5050 PRINT D$;"CLOSE"
5060 END
```

These few statements will OPEN a Sequential File (there is no L parameter with OPEN, so it's Sequential) and redirect the computer output data into the Text File. The LIST command would normally put the program listing on the monitor, but in this case it will go into the OPEN file, line numbers and all. Once the listing is in a Sequential File, you can then EXEC this file to put the information back into memory. The command

```
EXEC LISTFILE
```

will do just that. Since all of the lines in the EXEC File are numbered, they will not be executed but merely put back into memory.

To Basic, this appears as if an operator is entering a program at the keyboard. This is a very useful example of an EXEC File and as you begin to write larger Basic programs, this can save you some time.

Some final notes on EXEC files. You cannot stop an EXEC file with a Control-C. Only one EXEC File can be OPEN at a time. As soon as you OPEN the second EXEC File, the remainder of the first EXEC File will not be executed. If an EXEC File contains illegal DOS, Monitor, or Basic commands, the error message SYNTAX ERROR is generated and DOS will continue to execute the EXEC file. The EXEC command can have an R positioning parameter. The R parameter is interpreted in the same manner as would be with any Sequential File. When EXEC OPENs the Text File, the position-in-the-file pointer is placed at the first character in the file. The value with the R parameter will then specify the number of the field in the file where execution will begin.

6.5.17 DOS Access Commands

There are seven more DOS commands which we have not discussed. They are:

MAXFILES	NOMON	PR#	INT
MON	IN#	FP	

These commands have their own unique functions and will be discussed separately.

MAXFILES When we discussed the Text File commands, we mentioned that each open file needed a 595-byte memory buffer work area. We mentioned that DOS supports up to 16 open files. If DOS provided enough work area for all 16 files, it would have a total of 9520 bytes of buffer. Since most programs never have more than a couple of open files, most of this space would be wasted. The command MAXFILES is used to inform the DOS as to how many of these buffers are needed. When you Boot DOS, there are three buffers allocated. If you wish to change the number of buffers available, you use the MAXFILES commands with the format:

```
MAXFILES N
```

where the N parameter must be from 1 to 16. You cannot change MAXFILES with a program loaded into memory. This must be done before you LOAD or RUN a program.

MON/NOMON These commands are primarily used when debugging a program. They inform the DOS to display all of the commands and all of the data going to or coming from a Text File. The format for enabling this debug function is:

```
MON C,I,O
```

The parameters mean: C is for command, I for input from the disk, and O for output to the disk. These parameters may be used in conjunction with each other, or one at a time. To disable this function, you use the command:

```
NOMON C,I,O
```

Again, the three parameters have the same meaning. After Booting or Reset, the DOS defaults to NOMON C,I,O.

IN#/PR# From the Basic Programming Reference Manuals, you know that these commands redirect the input and output for the computer. Since DOS also redirects the input and output for the computer (when you're using Text Files), these commands can cause problems if they are not intercepted by DOS. When you are using DOS, you can use these commands in immediate execution mode. This means that typing IN#S or PR#S (where S is the slot number) while in the command state causes no problems. However, from a Basic program, these commands should be treated as DOS commands and issued with:

```
10 PRINT D$;"IN#6"
```

where the D$ is a Control-D and, therefore, this becomes a DOS command. The same format should be followed for the PR# command.

FP Ss,Dd This command can be used to switch from Integer Basic to Applesoft without disturbing DOS. However, you must have Applesoft available in one of three places: (1) in a ROM card, (2) on the diskette in the selected slot and drive (or on the default drive if you haven't specified the S or D parameters), or (3) already loaded into your Language Card. If Applesoft is not available, the DOS error message LANGUAGE NOT AVAILABLE will be displayed.

The HELLO program on the DOS 3.3 System Master will check your system to see if a Language Card is installed. If you have one,

this program will check to see which Basic language you have in ROM and then load the other language into the Language Card. This means your computer will then have both Basics available.

The DOS command FP will also clear memory and reset HIMEM.

INT This DOS command can be used to move from Applesoft to Integer Basic. Unlike FP, this command will not access the disk drive for the Integer Basic language. If you use the S and D parameters with this command you will get a SYNTAX ERROR. This command should only be used on Integer computers or computers with Integer in the Language Card. The INT command will also clear memory and reset HIMEM.

6.5.18 Text File Lesson Guide

When we completed the description of the DOS Housekeeping commands, we presented a lesson guide to help you become familiar with those commands. In this section we will present several examples of programs which use Text Files. We hope that by studying these examples, you will begin to feel comfortable with Text Files.

Our first sample program will build a Sequential Text File. This is a general purpose program and you can use it to build any Sequential File, including an EXEC file.

To get started you should first Boot DOS. Next, type CATALOG to be sure DOS is properly Booted. Before you begin to enter our sample program be sure to type NEW to clear memory. To be sure you have cleared memory, it is a good practice to type LIST. If no Basic statements are displayed, you can be sure memory is clear. Our sample program is 21 lines long and you can now enter this program into memory.

MAKE FILE

```
  5  DIM B$(100),C$(100)
 10  D$="":REM DON'T FORGET CTRL-D
 20  A$="":REM THIS IS A CTRL-A
 30  CALL - 936
 40  PRINT "THIS PROGRAM BUILDS A SEQUENTIAL"
 50  PRINT "TEXT FILE. WHAT DO YOU WANT?"
 60  PRINT "TO CALL THIS FILE?"
 70  VTAB 10 : INPUT B$
 80  PRINT D$;"OPEN";B$
 90  CALL - 936
100  PRINT "PLEASE ENTER A FIELD FOR YOUR FILE."
110  PRINT "TO QUIT AND CLOSE THE FILE"
120  PRINT "PLEASE TYPE CONTROL-A."
130  VTAB 10 : INPUT C$
140  IF C$=A$ THEN 190
150  PRINT D$;"WRITE";B$
160  PRINT C$
170  PRINT D$
180  GOTO 90
190  PRINT D$;"CLOSE";B$
200  END
```

Once you have entered this program, you should SAVE it on the disk. To do this you should type:

SAVE MAKEFILE

where we have used MAKEFILE as the name of our program.

Before we RUN MAKEFILE, you should review the program to make sure you know what it will do.

Line 5 is for Integer programs only. With Integer Basic you must tell the Apple the maximum length of any string you use.

Lines 10 and 20 define two character strings called D$ and A$. These two strings are single-control character strings which will be used by other parts of the program. Lines 30 through 60 clear the screen and print a message. This message tells you what the program will do and asks you for a file name. Line 70 positions the monitor cursor and waits for you to enter the file name.

The input statement gets a file name from the keyboard and saves that name in a string called B$. If you are a good typist you might have named this string FILENAME$. For most of us B$ is much more convenient.

Now that we know the name of the file, line 80 can OPEN the file. This line is the standard format for a DOS command from inside a program. The D$ informs the DOS that this is a command. We place the OPEN inside quotes and then send the B$, which is the file name. We have now opened a file with the operator-specified name. Lines 90 through 120 output a second message requesting the operator to enter a field to be placed in the file. Line 130 inputs this field into C$. Line 140 compares C$ to A$, which line 20 defined as a Control-A character. If C$ is a Control-A, this means to terminate the program and execution goes to line 190, which will CLOSE the file and END the program.

If C$ does not equal a Control-A, then C$ becomes part of the file. Line 150 is a DOS WRITE command and line 160 moves C$ into the file. Line 170 is a null DOS command (a Control-D by itself) which cancels the DOS WRITE command.

Finally, Line 180 will cause the program to loop back for another field. The only way out of this loop is to input a Control-A.

It is now time to RUN the program. If the program is still in memory you can type RUN; or if you want to reload the program from the disk, you can type RUN MAKEFILE. The program will clear the screen and ask you for a file name. Let's call this first file TEST1. Type TEST1 and hit Return. Next, the program will clear the screen and ask for a field to place into the file. You should now type your name and hit Return. The program will clear the screen and ask for another field. This time you can type your address or telephone number and hit Return. Each time you enter a line and hit Return, that field will be placed in the file. When you have entered as many fields as you want, you can stop the program by typing a Control-A immediately followed by a Return. This will CLOSE the file and END the program.

We have now built a Sequential File and stored it on the diskette. If you type CATALOG you should see your new file in the diskette directory. Our next sample program will allow you to display a Sequential File. We will call this program READFILE. With DOS still Booted, you should clear memory (NEW and the LIST to be sure) and enter this program.

READ FILE

```
  5  DIM B$(100),C$(100)
 10  D$="":REM DON'T FORGET THE CTRL-D
 20  A$="":REM THIS IS A CONTROL-A
 30  CALL - 936
 40  PRINT "THIS PROGRAM WILL READ AND"
 50  PRINT "DISPLAY A SEQUENTIAL FILE."
 60  PRINT "WHAT IS THE FILE NAME?"
 70  VTAB 10 : INPUT B$
 80  PRINT D$;"OPEN";B$
 90  CALL - 936
100  PRINT D$;"READ";B$
110  INPUT C$
120  PRINT D$
130  PRINT C$
140  VTAB 20
150  PRINT "TO READ THE NEXT FIELD"
160  PRINT "HIT RETURN. TO STOP THE"
170  PRINT "PROGRAM TYPE CONTROL-A."
180  INPUT E$
190  IF E$=A$ THEN 210
200  GOTO 90
210  PRINT D$;"CLOSE"
220  END
```

Once you have entered the program, it is a good idea to SAVE it on the diskette. Before you RUN the file READFILE, let's review what it does.

Line 5 is strictly for Integer Basic programs. Lines 10 and 20 define two character strings (D$ and A$) which are used later in the program. Lines 30 through 60 clear the screen and output a message which tells what the program does and asks for a file name. Line 70 positions the cursor and waits for you to enter the file name. The INPUT statement will store the file name in B$. Line 80 can then OPEN the file. Line 90 will clear the monitor, line 100 will issue the DOS READ command, and then line 110 can READ a field from our file. The INPUT statement in line 110 is preceded by a DOS READ command, so this input statement is directed to the Text File and not the keyboard.

We now have a field from our file in C$. Line 120 is a Null DOS command and is used to cancel the DOS READ command. Line 130 is used to display C$ on the monitor. This line accomplishes the purpose of this program; that is, it displays the contents of a Sequential File, one field at a time. Lines 140 through 170 print a second message. This message requests that you hit Return to display the next field of the file or type Control-A followed by a Return to terminate the program. Line 180 gets the operator response, and Line 190 checks the response for a Control-A. If the response is not a Control-A, the program continues to Line 200 which loops back and READs the next field from the file. If the response is a Control-A, the program executes Line 210, which CLOSEs the file. Line 220 ENDs the program.

To start the program type:

```
RUN READFILE
```

The program will clear the screen and request a file name. You should enter TEST1 and hit Return. (TEST1 is the file we created with MAKEFILE.) The program will clear the screen and READ the first field from our file. If you entered your name as the first

field, you should see your name displayed on the screen. To display the next field of the file, hit Return. In this manner you can display each of the fields in the file. To terminate the program, you can type a Control-A followed by a Return. If you do not terminate the program, it will continue to display each field until the end of the file is encountered. At this point, the DOS error END OF DATA will be displayed and the program will terminate.

You can now use the programs MAKEFILE and READFILE to create and display Sequential Files. One useful application for these programs is to create an EXEC file. As you know, an EXEC file is merely a Sequential File that contains DOS, Basic, or Monitor commands. To create an EXEC file, let's RUN MAKEFILE. When the program requests the file name, use TESTEXEC. We have an example of an EXEC File for you. As MAKEFILE asks for each new field of the file, you should enter the next line of our EXEC File example. Please enter each line just as you see it. After you have entered the last line (DELETE MADE BY EXEC), you should terminate MAKEFILE with a Control-A.

```
CATALOG
NEW
10 REM: THIS IS A TEST
15 REM: THIS IS ONLY A TEST
20 PRINT "ONLY A TEST"
25 END
LIST
SAVE MADE BY EXEC
LOCK MADE BY EXEC
CALL - 151
FF69L
3D0G
RUN MADE BY EXEC
UNLOCK MADE BY EXEC
DELETE MADE BY EXEC
```

Once you have entered the information into TESTEXEC, you can use READFILE to check its contents. To check the file, RUN READFILE and use the file name TESTEXEC. When you are sure the contents of TESTEXEC are correct, it is time to execute our EXEC File. Type:

```
EXEC TESTEXEC
```

DOS will now OPEN our sample EXEC File and READ the first field. This is a CATALOG command, which DOS will immediately execute. You should see the directory of the diskette displayed on the monitor. Next, DOS will READ the second field of our EXEC file. This is the Basic command NEW. DOS will pass this command to Basic for execution. Once Basic has executed the NEW command, it will return to DOS for another command. Again, DOS will fetch the next field from the EXEC File. This time the field is a numbered Basic line. This will be passed to Basic where it will be stored in memory as part of a program. The next three lines are treated the same way. (Using our EXEC File, we have now placed a small program in memory.) The next field of our EXEC File (LIST) is another Basic command which will list this new program.

The next line of our EXEC File will SAVE MADE BY EXEC on the diskette. This is the DOS SAVE command and we have called the program in memory MADE BY EXEC. After we SAVE the program, the next line of our EXEC example will LOCK the file. We have now demonstrated several DOS and Basic commands from an EXEC File. The next three lines will use Monitor commands. CALL - 151 moves

us from Basic to the Monitor. FF69L will list the disassembled machine code for the Monitor entry point. Your computer Monitor will display several lines of disassembled 6502 code. Finally, 3D0G will cause the 6502 to exit the Monitor and return to Basic. Back in Basic, the EXEC File will RUN the program MADE BY EXEC. After the program ends, the EXEC File will UNLOCK MADE BY EXEC and then DELETE MADE BY EXEC.

We have now demonstrated the use of an EXEC File. We have showed you how to use DOS, Basic, and Monitor commands from inside the EXEC File.

6.5.19 DOS Utilities

So far we have described the DOS and how to use the DOS commands. This section will describe two programs that run under DOS. These two programs are FID and COPY.

The primary purpose for FID is to allow you to copy individual files from diskette to diskette and to determine the unused space on a diskette. The FID program is available on the DOS 3.3 System Master Diskette. FID is a Binary program. To execute the program, you should type:

```
BRUN FID
```

You should not enter a starting address. The DOS will determine the proper starting address; and after DOS has loaded FID, you should see a sign-on message followed by a menu. FID is a menu-driven program with operator prompting. You should find this program very easy to use. To execute one of the FID commands, you merely type the number next to the command. The commands are:

1. COPY FILES
2. CATALOG
3. SPACE ON DISK
4. UNLOCK
5. LOCK
6. DELETE
7. RESET SLOT AND DRIVE
8. VERIFY
9. QUIT

Five of these commands are standard DOS commands (CATALOG, UNLOCK, LOCK, DELETE, and VERIFY). The functions for these commands are the same with DOS. To execute one of the commands, type only the number next to the command. DOS will then prompt you for the required parameters.

SPACE ON DISK will tell you the number of sectors already used and the number still available. This command is very useful when you have a large number of files on a diskette.

RESET SLOT AND DRIVE is used to force FID to reprompt you for a new slot and drive number. With most of the commands, FID will only ask you for the slot and drive number once. Thereafter, it will use this slot and drive as the default.

The QUIT command is used to exit the program and return to Basic.

This leaves the COPY command which is the primary purpose for FID. This is the easiest and fastest way to move individual files from one diskette to another. With FID, this can be done using one or

two drives. The first time you request the COPY command, FID will prompt you for the source slot and drive and the destination slot and drive. You can use one drive as both source and destination, or you can use two separate drives.

Once you have entered the source and destination drives, FID will prompt you for a FILENAME. After you enter the FILENAME, FID will move the file from the source to the destination drive. FID will transfer any file type (I, A, B, or T).

FID has another feature which makes handling files much easier. This feature allows you to specify a partial file name. The FID "wildcard" character = can be used to replace part or all of a file name. For instance, if you specify = in place of the FILENAME, FID will copy all of the files on the diskette. If you specify part of a name followed by =, for instance,

```
CUSTOMER =
```

FID will copy all of the files that start with CUSTOMER regardless of the remaining characters in the name. If you specify =, followed by part of the name, FID will copy all of the files whose names end with the specified character. Using the "wildcard" character makes copying multiple files much easier.

If you specify the wildcard character, FID will ask you if you want prompting. If you answewr Y, FID will display each file name that meets the wildcard parameters and wait for the operator to verify that file name before copying the file. If you answer N to prompting, FID will automatically copy all of the files that meet the wildcard parameters.

A second very useful DOS utility is the COPY program. This program is also available on the DOS 3.3 System Master. The purpose of this program is to make a complete copy of diskette. There are actually two COPY programs: COPY is an Integer Basic program and COPYA is an Applesoft program. To use one of these programs type:

```
RUN COPY
```

with Integer machines, or

```
RUN COPYA
```

with Applesoft. The COPY or COPYA programs will copy the entire diskette with all of the files, regardless of the file types.

DOS will load the program and begin execution. The COPY program will ask for a source slot and drive number and a destination slot and drive number. If you specify the same slot and drive for both the source and destination drives, the program will prompt you to change the diskette from the original to the new copy as required. This will usually involve swapping diskettes four times. If you specify two separate drives, the program will make a copy without swapping diskettes.

7. A2 UTILITY DISKETTE

The diskette you received with your A2 does not contain a DOS 3.3 system. This diskette is in DOS 3.3 format and has a diagnostics program to check out your disk drives on the Boot tracks. Section 5 explains this diagnostic program. This diskette also contains a standard DOS directory.

If you Boot DOS 3.3 and CATALOG this diskette, you will see the diskette also contains a Binary program called Speed. This program is described in Section 7.1.

7.1 SPEED TEST

The MICRO-SCI A2 Utility Diskette contains a binary program called Speed. To use this program, you must Boot DOS 3.3, put the A2 Utility diskette into the drive, and type:

```
BRUN SPEED
```

Once it is loaded, the Speed program will display operating instructions. Following these instructions, you should select the desired drive and start the speed test. The proper speed is 200.00 ± 2.50 milliseconds. It is normal for the displayed speed to vary by as much as .50 milliseconds from one rotation to the next. If the displayed speed stays within 197.50 to 202.50 milliseconds, we say the drive is within an acceptable speed tolerance. If the speed varies outside this range, we suggest that the speed should be adjusted to stay within the proper range.

Using this program, a user can adjust the drive speed. This adjustment is quite simple and requires no special tools, only a Phillips screwdriver and a small common screwdriver. If, however, you feel uncomfortable "tinkering" with electronic equipment, we suggest you take the drive to your dealer for adjustment or contact MICRO-SCI.

To adjust the drive speed, you will need to remove the cover on the drive unit. First, turn off the computer power. There are four Phillips screws in the sides of the drive. Remove these screws and slide the cover off the drive. There is a small electronics board vertically mounted to the back of the drive chassis. This board is the speed control board. Mounted on this board, about 1/2-inch above the bottom of the drive, is a rectangular potentiometer. It is the only component in the location described above that has a visible adjustment screw. This is the speed control adjustment. (There is another potentiometer inside the drive unit, but it is on the large electronics board on top of the drive. This potentiometer is the write current adjustment and cannot be adjusted without the proper equipment.)

After you have removed the drive cover and located the speed control adjustment, you can re-Boot the DOS. Then put in the A2 Utility Diskette and BRUN SPEED. Select the proper drive and start the speed check. While the program is displaying the rotational speed of the drive, turn the speed control potentiometer until the display indicates 200.00 ± 1.0 milliseconds. Once you have adjusted the speed, terminate the Speed program, turn off the Apple power switch, and put the cover back on the A2 drive.

7.2 UTILITY DISKETTE COPIER

You can make back-up copies of the A2 Utility Diskette; however, the standard COPY programs COPY and COPYA will encounter an I/O ERROR when they attempt to copy the track formatted for the speed check. This track is not in the standard DOS 3.3 format. To overcome this problem, we have included on the A2 Utility Diskette the EXEC file, Utility Copier. This file is used to modify the COPY and COPYA programs so that they do not try to copy the speed track and, therefore, will not encounter an I/O ERROR when copying the Utility Diskette.

To copy the Utility Diskette, you must Boot DOS, put the Utility Diskette into Drive 1, and type:

```
EXEC UTILITY COPIER
```

The DOS will load the EXEC file and prompt you to put a diskette containing COPY or COPYA into Drive 1. Remove the Utility Diskette and insert a diskette containing the appropriate copy program. The EXEC file will load COPY, modify it, and then execute it. The modified COPY program will operate just like the unmodified COPY program; however, it will not attempt to copy the speed track. Once the COPY program finishes, you will have a back-up copy of the A2 Utility Diskette. This new Utility Diskette must have the speed track formatted by running the Speed Program. Section 7.1 explains how to run the Speed Program.

GLOSSARY

DOS	Disk Operating System
FP	Floating Point (Applesoft)
INT	Integer
I/O	Input/Output
IOB	Input/Output Block
PCB	Printed Circuit Board
RAM	Random Access Memory
ROM	Read Only Memory
RWTS	Read/Write Subsystem
VTOC	Volume Table of Contents

INDEX

INDEX (continued)

INDEX (continued)