ATARI®
DOS 2.5: 1050™
DISK DRIVE

OWNER'S MANUAL

IMPORTANT INFORMATION

Like any electrical appliance, the ATARI 1050 Disk Drive uses and produces radio-frequency energy. If it is not installed and used according to the instructions in this guide, the equipment may cause interference with your radio or television reception.

If you believe that this equipment is causing interference with your radio or television reception, try turning the equipment off and on. If the interference problem stops when the equipment is turned off, then the equipment is probably causing the interference. With the equipment turned on, you may be able to correct the problem by trying one or more of the following measures:

- Reorient the radio or television antenna.
- Reposition the equipment in relation to the radio or television set.
- Move the equipment away from the radio or television.
- Plug the equipment into a different wall outlet so that the equipment and the radio or television are on different branch circuits.

If necessary, consult your ATARI Computer retailer or an experienced radio and television technician for additional suggestions.

Another helpful resource is *How to identify and Resolve Radio-TV Interference Problems,* a booklet prepared by the Federal Communications Commission. This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, Stock No. 004-000-00345-4.

WARNING: This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. These rules are designed to provide reasonable protection against such interference when the equipment is used in a residential setting. However, there is no guarantee that interference will not occur in a particular home or residence. Only peripherals (computer input/output devices, terminals, printers, etc.) that have been certified to comply with the Class B limits may be attached to this equipment.

Operation of noncertified peripherals with this equipment is likely to result in interference with radio and TV reception.

Every effort has been made to ensure the accuracy of the product documentation in this manual. However, because we are constantly improving and updating our computer software and hardware, Atari Corp. is unable to guarantee the accuracy of printed material after the date of publication and disclaims liability for changes, errors, and omissions.

ATARI®
DOS 2.5: 1050™
DISK DRIVE

TABLE OF CONTENTS

YOUR ATARI 1050 DISK DRIVE AND DOS

The ATARI® 1050™ Disk Drive is an extremely efficient, high-speed memory device which greatly enhances your ATARI Personal Computer system. Your ATARI Computer's memory retains the information and instructions you enter through its keyboard. But the computer's memory is limited in size, and without a storage device like the 1050, its contents are erased each time you turn off the computer.

Your ATARI 1050 Disk Drive enables you to store and manage large amounts of information in separate *files* on diskettes. With your ATARI 1050, you can call up your files by name, copy or erase them, and manage them in many other ways.

What DOS Does

To store information on diskettes you need software that allows your computer and disk drive to communicate with each other about your files. That is where the Disk Operating System — DOS for short — fits into the picture. DOS (pronounced "doss") is a program that enables your computer and disk drive to work together in storing, retrieving, and otherwise managing your diskette files. DOS itself is organized in files contained on the Master Diskette.

You must load DOS into your computer before it can work with your disk drive. Some ready-made computer programs already contain a version of DOS, sparing you the trouble of loading it separately. But with other programs, especially those in cartridge form, you have to load DOS along with the program if you plan to use a disk drive with the program. In any case, you need DOS for many essential tasks: to prepare blank diskettes to store your files, for example, and to make backup copies of important files and diskettes.

DOS 2.5 and Your ATARI Personal Computer System

With just a few restrictions, DOS 2.5 is compatible with the earlier ATARI DOS 2.0S. You can also convert DOS 3 files to DOS 2.5 formats. This means that you can use DOS 2.5 to manage files originally stored and managed using the older versions of DOS.

You can use DOS 2.5 with both the ATARI 1050 and the ATARI 810™ Disk Drive. However, your system must include at least one 1050.

DOS 2.5 allows you to format diskettes and store information in either single or enhanced density. With enhanced density you can record about 50 percent more data on each diskette than you can with DOS 2.0S. But you can manage enhanced-density storage *only* if you have an ATARI 1050 Disk Drive; the 810 Disk Drive is not capable of formatting or managing data stored in enhanced density (including the files on your DOS 2.5 Master Diskette — this is why you must have a 1050 Disk Drive to begin working with DOS 2.5). If your computer system includes an 810 Disk Drive that you will often use to access your files, you may want to format all your diskettes in single-density.

For complete details on using a system that includes both a 1050 and an 810 Disk Drive and working with DOS 2.0S files with DOS 2.5, see Appendix H. If you are working with DOS 3 Files, see Appendix L.

With either or both disk drives, use single-sided, double-density diskettes.

DOS 2.5 works with any ready-made cartridge-based program that runs on your ATARI Computer — even programs that predate DOS 2.5, including the AtariWriter™ word processor and ATARI BASIC. With these and other cartridge-based programs, you can always use DOS 2.5 instead of DOS 2.0S to prepare data diskettes and manage files.

Many diskette-based programs designed for use with the earlier DOS 2.0S can also be converted from DOS 2.0S to DOS 2.5, although you may have to continue to use DOS 2.0S with certain *protected* diskette programs (see your program user's manual if you are unsure whether a program is protected).

How to Use This Manual

This manual is designed to serve everyone from the beginning computer user to the advanced programmer. It includes two sections that introduce the 1050 Disk Drive and DOS, a detailed guide to all the capabilities of DOS 2.5, a section of more technical information primarily for programmers, and several appendices. Appendix J, a brief glossary of terms used in talking about DOS, may be particularly helpful to the beginner. As you work with this manual, consult the glossary whenever you are unsure of a term's meaning.

If you just purchased your first disk drive, you should start with Section 1, "Your ATARI 1050 Disk Drive," which provides simple instructions for setting up and using your drive.

Section 2, "Getting Started With DOS 2.5," introduces you to the most frequently used functions of DOS. With step-by-step instructions, it explains how to load DOS into your computer, prepare diskettes to store your files, duplicate diskettes, name and refer to your files, and copy and erase files. Most importantly, it explains how to make a System Diskette, or working copy of DOS. This is a very important procedure that you should not neglect before going on to learn more about DOS. To go through the examples and exercises in this section, you will need at least three blank diskettes, one to make your System Diskette and two to use as practice data diskettes.Since some of the exercises involve the use of ATARI BASIC, you will also need a BASIC cartridge if you have an ATARI 400™, 800™, or 1200XL™ Computer. If you have a 130XE™, 65XE™, or 800XL™, your computer is equipped with built-in BASIC.

Section 3, "Selecting a DOS Menu Option," covers every function on the DOS 2.5 Menu and provides step-by-step examples of how to use each one.

Section 4, "Using BASIC Commands with DOS 2.5," will be of interest primarily to the advanced user or programmer.

The appendices cover a number of both technical and nontechnical topics. You may find the following especially useful:

- See Appendix C if you run into any error messages as you get started with DOS 2.5.
- If you have more than one disk drive, see Appendix G.
- If your computer system includes an ATARI 810 Disk Drive or if you have diskette files created and stored using the earlier ATARI DOS 2.0S, see Appendix H.
- If you have an ATARI 130XE Computer, see Appendix K.
- If you have diskette files created and stored using ATARI DOS 3, see Appendix L.

SECTION 1
YOUR ATARI 1050 DISK DRIVE

When you unpack your ATARI 1050 Disk Drive, be sure you have the following items:

- ATARI 1050 Disk Drive
- Serial I/O Cable
- AC Power Adapter
- DOS 2.5 Master Diskette
- 1050 Owner's Manual
- Warranty/Registration Card

If you are missing any of these items, contact your dealer. You should save the packing materials in case you ever want to transport the disk drive or send it through the mail.

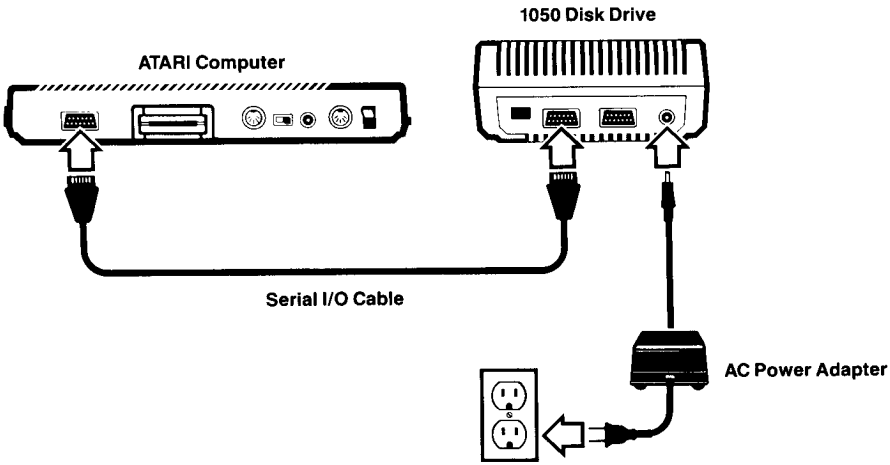
Connecting Your 1050 Disk Drive

Follow these steps to connect your 1050 to your ATARI Personal Computer:

1. Turn off the power to all components of your computer system.

2. Make sure the Power ON/OFF switch of your 1050 (located on the front panel of your disk drive) is in the OFF position.

3. First plug the smaller end of the AC Power Adapter cord into the hole marked POWER IN on the back of the disk drive. Then plug the AC Power Adapter into the wall socket.



4. Plug one end of the Serial I/O Cable into the jack marked PERIPHERAL on the computer console. Then plug the other end of the cable into one of the two jacks marked I/O CONNECTORS located on the back of the disk drive.

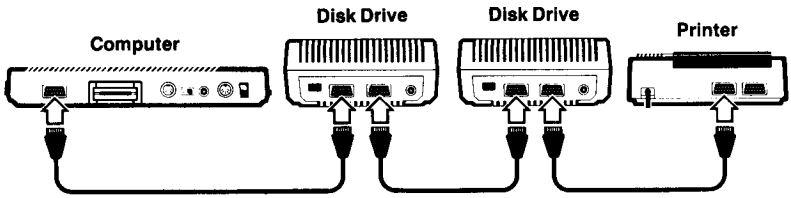
   Warning: Your ATARI 1050 Disk Drive should be placed 12 inches or more from your television. Your TV creates a strong magnetic field that could affect the information recorded on your diskettes.

5. Turn on the disk drive. Both the POWER light (next to the ON/OFF switch) and the BUSY light (above the switch) will go on. When the BUSY light goes off, you are ready to insert a diskette.

Your disk drive is now ready to receive the DOS 2.5 Master Diskette. It is recommended that you read the next section of this manual before proceeding.

Connecting More Than One Disk Drive

You can attach up to four ATARI Disk Drives, in addition to ATARI Printers, Program Recorders or other components, to your ATARI Computer. Multiple disk drives (and other components) are connected to each other in a *daisy chain,* using the Serial I/O Cables supplied with each component.

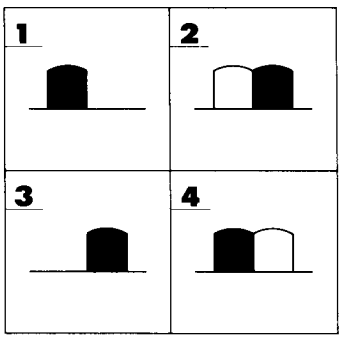


There are two I/O CONNECTOR jacks on the back of each disk drive. To install multiple drives, connect a Serial I/O Cable from one I/O CONNECTOR on the first disk drive to the jack labeled PERIPHERAL on the computer console. Then connect another Serial I/O Cable from the remaining I/O CONNECTOR on the first drive to either I/O CONNECTOR on the second drive. Connect any additional components in the same way.

Setting Drive Select Switches

If your system includes more than one disk drive, you must set the drive codes using two small identifier switches on the back of each drive. These switches tell the computer which drive you're referring to in your programs and commands.

To set the switches, first turn off the power to the disk drives. Then turn the drives around so that you can see the DRIVE SELECT window on the back of each drive. Inside the window is a black switch; behind it, a white switch.



Using a pen or a small screwdriver, set the switches in the window to match the patterns shown here for Drive 1, Drive 2, and so forth. *You must always have one drive set as Drive 1.*

Once you have set the drive switches, be sure to label each disk drive with its number so that you will not mistake one drive for another when using DOS.

Taking Care of Your Diskettes

The surface of a diskette is coated with a sensitive magnetic material that stores your data. To ensure the long life and reliability of your diskettes, you must handle them properly and with care.

Each diskette is permanently enclosed in a black protective envelope and is normally stored in a paper sleeve. Most diskettes have a small *write-protect* notch on one edge of the black protective envelope. By covering this notch with one of the small adhesive rectangular tabs provided by the diskette manufacturer, you can avoid accidentally erasing or writing over any data on a diskette. See Section 2 for more information on write protecting your diskettes.

Remember these rules for the proper care and handling of diskettes:

- Never turn your disk drive on or off with a diskette in the drive, and never leave a diskette in the drive while it's turned off.
- Never wet or wash a diskette. Use a soft brush or compressed air from a spray can to remove any dust from the surface.
- Do not bend your diskettes; they must turn freely in the protective envelope. Handle them with care when loading or unloading.
- Store diskettes in their paper sleeves standing on edge.
- Store your diskettes away from your television set. The strong magnetic fields produced by the television can partially erase the data stored. Keep your diskettes away from electrical devices, including the telephone.
- Do not store your diskettes in direct sunlight. Keep them away from any excessive heat.
- Because a diskette turns inside its envelope, damage to the envelope can result in damage to the diskette.
- Do not write on your diskettes with a pencil or ball-point pen. The sharp point of a pencil or ball-point pen can score the surface of a diskette. Use a felt-tip pen to mark the diskette label or write on the label before you put it on the diskette.
- Do not use erasers on diskette labels. Eraser dust is abrasive and will damage diskettes.
- Do not attach paper clips to your diskettes.
- Never touch a diskette where it is exposed through the diskette envelope. Fingerprints can damage the magnetic medium.

Labeling Your Diskettes

Most diskettes come with labels already affixed to one corner of the black protective envelope or with a set of labels that you can attach to each diskette. Be sure to label every diskette on which you copy or store programs and files.

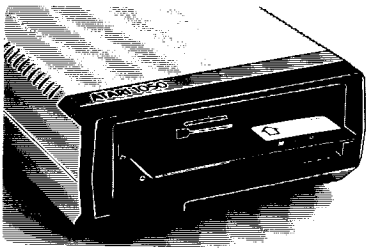
SECTION 2
GETTING STARTED WITH DOS 2.5

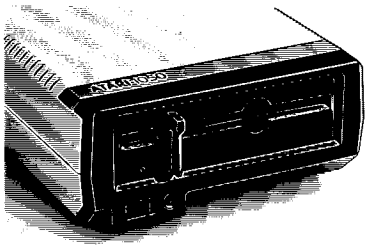
Loading DOS

Follow these steps to load DOS 2.5 with ATARI BASIC (if you do not want to load BASIC, see "DOS With and Without BASIC," on the following page):

1. Make sure that your computer and disk drive are turned off. If you have an ATARI 400, 800, or 1200XL Computer, insert an ATARI BASIC cartridge in your computer's cartridge slot and make sure that there is no diskette in your disk drive. If you have an ATARI 130XE, 65XE, or 800XL with built-in BASIC, just be sure there is no cartridge in your computer, and no diskette in your disk drive.
2. Turn on your disk drive — if you have more than one disk drive in your system you must always use Drive 1 to load DOS. The drive makes a whirring sound when turned on, and the POWER and BUSY lights go on. After a few seconds, the noise stops and the BUSY light goes off.

   Caution: Never insert or remove a diskette in your drive while the BUSY light is on.





3. When the BUSY light goes off, turn the latch on the front of your ATARI 1050 Disk Drive to the open (horizontal) position. Remove your DOS 2.5 Master Diskette from its protective paper sleeve and insert it in your drive, with the label facing up and toward you, until it clicks into place. Then turn the latch to the closed (vertical) position.

4. Turn on your computer. The disk drive's BUSY light goes on again as DOS loads into your computer, and the drive makes a clicking sound. If you turn up the volume on your TV, you can hear it *beep* as DOS loads.
5. When the READY prompt (from ATARI BASIC) appears on your screen, type **DOS** and press [Return].

DOS With and Without BASIC

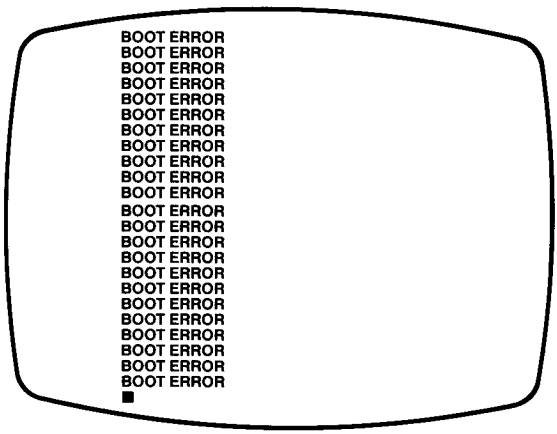
Naturally, you won't always be using DOS 2.5 with ATARI BASIC. To load DOS without BASIC if you have an ATARI 400, 800, or 1200XL Computer, simply follow the steps explained above *without* inserting your BASIC cartridge and omit step 5.

If you have an ATARI 130XE, 65XE, or 800XL, your computer is equipped with built-in BASIC. BASIC is loaded into your computer whenever you turn it on, including when you load DOS — *unless* you first insert a program cartridge in the computer's cartridge slot *or* you hold down the [Option] key on your computer console as you turn it on.

As you've seen, going from BASIC to DOS is easy — just type **DOS** and press [Return]. Going from DOS back to BASIC is just as easy; see "Running a Cartridge From DOS."

Boot Errors

Loading a program into a computer when you first turn it on is called *booting up*. If a problem occurs when booting up your system, the following appears on your screen:



When you start your system, a boot error can occur for the following reasons:

- The inserted diskette does not have DOS on it.
- The diskette was inserted incorrectly.
- The diskette has been scratched, warped, or marred. In this case, use another diskette.
- The diskette is an enhanced-density diskette in an ATARI 810 Disk Drive.

The following conditions will also cause a boot error, but no indication of it will appear on the screen.

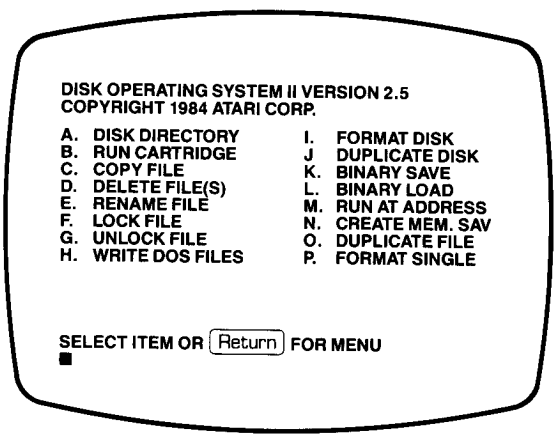
- The disk drive was turned on *after* the computer was turned on.
- The disk drive is not properly connected to the computer console.
- The Power Adapter plug has loosened from its wall socket.
- The Power Adapter plug has loosened from the disk drive POWER IN socket.
- The drive code setting is not correct.

If you have checked and found none of these problems, take the following steps:

1. Insert the DOS 2.5 Master Diskette or System Diskette (working copy of DOS 2.5) in Drive 1 and reboot the system.
2. Remove the DOS diskette.
3. Reinsert the problem diskette and save any accessible files on another diskette using the process for copying files (see C., COPY FILE, in Section 3).
4. With the problem diskette in Drive 1, use the DELETE FILE(S) function to erase all the files.
5. Try using the diskette again. If this fails, the diskette will have to be reformatted (see I., FORMAT DISK, in Section 3).

The DOS Menu

Once you've loaded DOS into your computer and, if necessary, typed **DOS** and pressed [Return], the DOS Menu appears on your TV or monitor screen. The menu presents a list of DOS functions. The prompt below the menu invites you to make a selection. You choose the function you want to use by pressing the letter corresponding to your selection and pressing [Return]. DOS then asks you for the information it needs to proceed (see "Prompts and Responses," later in this section).



Here is a summary of the DOS 2.5 Menu options. Those marked with an * are introduced later in this section; all are explained in detail in Section 3.

*A. DISK DIRECTORY
This option allows you to call up a complete or selective list of the files on a diskette, showing the filenames, extenders (if any), the number of sectors allocated to each file, and the number of free sectors still available on the diskette.

*B. RUN CARTRIDGE
(Can ONLY be used with built-in BASIC or with a cartridge installed in the computer console.) This option allows you to return control of your system to built-in BASIC or to the cartridge inserted in the cartridge slot (the left cartridge slot in the ATARI 800 Computer).

*C. COPY FILE
Use this option when you have two or more disk drives and you want to copy files from one diskette to another. Also use this option to copy a file on the *same* diskette, assigning a second name to the copy.

*D. DELETE FILE(S)
This option lets you erase a file from a diskette, increasing the available space on the diskette.

E. RENAME FILE
Use this option when you want to change the name of a file.

F. LOCK FILE
This option can be used to prevent you — or anyone else — from changing, renaming, or accidentally erasing a file. You will still be able to read the file, but will not be able to write to it. When the directory is displayed, an asterisk is placed in front of the filename to indicate that the file is locked.

G. UNLOCK FILE
This removes the asterisk from in front of the filename and allows you to make changes to the file, rename it, or delete it.

H. WRITE DOS FILES
Use this option to add the DOS files (DOS. SYS and DUP.SYS) on your Master Diskette or System Diskette to a diskette in any disk drive.

*I. FORMAT DISK
This option is used to format a blank diskette, which is necessary before you can record any information on it. Be sure you do not have any files you want to keep on a diskette before formatting it. This option will format a diskette in enhanced density provided you are using a 1050 Disk Drive; otherwise, it will format in single density.

*J. DUPLICATE DISK
This is the option you choose when you want to create an exact duplicate of a diskette. This option will automatically format the destination disk.

K. BINARY SAVE
With this option you can save the contents of specified memory locations on a diskette. (Manipulates assembly language programs.)

L. BINARY LOAD
This option lets you retrieve an object file from a diskette. It is the reverse function of BINARY SAVE. (Manipulates assembly language programs.)

M. RUN AT ADDRESS
With this option you can enter the hexadecimal starting address of an object program after it has been loaded into RAM with a BINARY LOAD. (Executes assembly language programs.)

N. CREATE MEM.SAV
This option allows you to reserve space on a diskette for the program in RAM to be stored while the DUP.SYS file is being used. For some applications like programming, it is a good idea to create a MEM.SAV file on each new diskette you intend to use as a System Diskette. As you become more familiar with DOS, you may find there are cases where a MEM.SAV file serves no useful function. So the inconvenience of waiting for MEM.SAV to load into memory may warrant deleting it from the diskette.

*O. DUPLICATE FILE
This option enables you to copy a file from one diskette to another, even if you have only a single disk drive.

*P. FORMAT SINGLE
Use this option when you want to format a diskette in single density using a 1050 Disk Drive.

Prompts and Responses

The questions and requests that DOS displays on your screen are called *prompts.* The answers you type into your computer are *responses*. DOS always prompts you for the information it needs to carry out your wishes. You will soon become familiar with the most common DOS prompts; since DOS requires the same kind of information for many of its functions, it won't be long before

many of your responses become almost automatic. As you use each function of DOS, the program lists its successive prompts and your responses on your screen as you proceed.

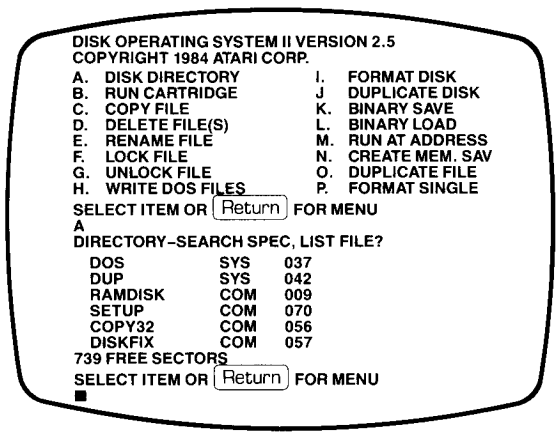
After typing a response into your computer, you must press [Return] to confirm your response. (Pressing [Return] *only* in response to certain prompts tells DOS to supply a preselected, or *default*, response — see "Defaults," later in this section). Many prompts require a simple yes or no answer. To answer yes, type **Y** and press [Return]. To answer no, type **N** and press [Return].

If you make a mistake while typing in a response, press [Delete Bk Sp] to erase the error, then type in the correct information. To delete an entire response before you confirm it, press [Shift] and [Delete Bk Sp] simultaneously.

Looking at a Disk Directory

Each diskette you use to store information has a *disk directory* that keeps track of the files stored on the diskette, how much room they take up, and how much free space is left on the diskette for storing more information. The DISK DIRECTORY selection on the DOS Menu allows you to check what files you have on your diskettes.

Since your DOS Master Diskette itself contains files, you can try out the function by looking at the directory of those files. With the DOS Menu on your screen, type **A**, then press [Return] *twice.*



These are the files that make up the DOS 2.5 program. The three-digit numbers in the right-hand column indicate how many *sectors* each file occupies on the Master Diskette. (See "Formatting a Diskette" for an explanation of sectors.) The line below the index tells you how many sectors remain for storing additional information on the diskette.

DOS.SYS and DUP.SYS are the files that execute the standard DOS functions. For an explanation of RAMDISK.COM, see Appendix K. Explanations of SETUP.COM, COPY32.COM, and DISKFIX.COM appear in Appendix L.

Duplicating a Diskette

With the DUPLICATE DISK option on the DOS 2.5 Menu, you can create an exact replica of a diskette. This function copies everything from your original, or *source,* diskette onto another, or *destination*, diskette. It also formats your destination diskette.

Caution: The DUPLICATE DISK function erases or writes over any information that may already be on a destination diskette. Never use a destination diskette that contains valuable files.

To Duplicate Your DOS Diskette

To learn how the DUPLICATE function works, make a duplicate of your DOS 2.5 Master Diskette. This is also an important safeguard. You should use your duplicate as your working copy of DOS, and keep the Master Diskette itself as a backup copy. Then you can use DOS without worrying about accidental damage to your working diskette.

As your destination diskette, use a new, blank diskette.

If You Have One Disk Drive

1. With the DOS Menu on your screen, type **J** and press Return. The prompt DUP DISK—SOURCE, DEST DRIVES? appears.

2. Type **1,1** and press Return. The prompt — INSERT SOURCE DISK, TYPE RETURN appears.

3. Place the diskette you want to duplicate in the drive — in this case, your DOS 2.5 Master Diskette — and press [Return]. The disk drive begins to "read" the information contained on your source diskette. Then DOS prompts you to INSERT DESTINATION DISKETTE, TYPE RETURN.

4. Remove your source diskette from the disk drive and insert a blank diskette (formatted or unformatted), then press [Return]. DOS writes the information it has read from your source diskette to your destination diskette, first formatting the destination diskette.

How many times DOS prompts you to insert your source and destination diskettes in your disk drive will depend on how much data is recorded on the source diskette and the amount of RAM in your computer system. When the prompt SELECT ITEM OR [Return] FOR MENU appears, the duplication process is complete.

Label your new copy of DOS 2.5 clearly — something like "DOS 2.5 — System Disk." Attach a write-protect tab to it (see "Using Write-Protect Tabs,") and use it as your working copy of DOS from now on. Store your original DOS 2.5 Master Diskette in a safe place.

If You Have Two Disk Drives

1. With the DOS Menu on your screen, type **J** and press [Return]. The prompt DUP DISK — SOURCE, DEST DRIVES? appears.

2. Type **1,2** and press [Return]. The prompt INSERT BOTH DISKS, TYPE RETURN appears.

3. Place your source diskette in Drive 1 — in this case, your original DOS 2.5 Master Diskette — and a blank diskette (formatted or unformatted) in Drive 2, then press [Return].

4. DOS duplicates all the information from your source diskette on your destination diskette, first formatting the destination diskette.

When the prompt SELECT ITEM OR [Return] FOR MENU appears, the duplication process is complete.

Label your new copy of DOS 2.5 clearly — something like "DOS 2.5 — System Disk". Attach a write-protect tab to it (see "Using Write-Protect Tabs," below) and use it as your working copy of DOS from now on. Store your original DOS 2.5 Master Diskette in a safe place.

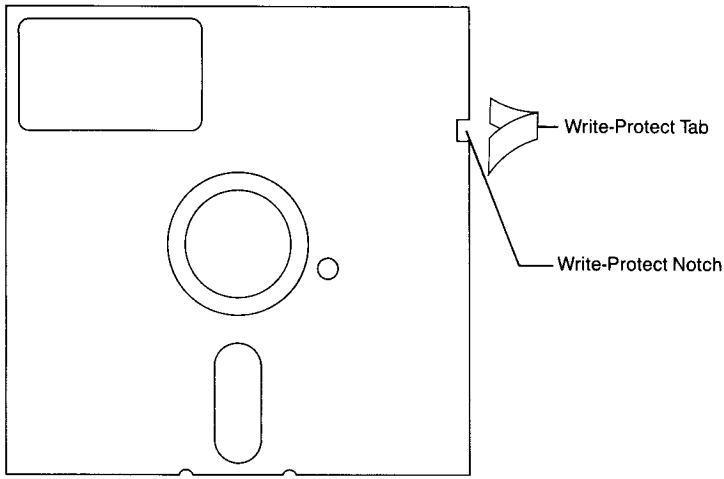
(Another way to make a DOS System Diskette is to use option H., WRITE DOS FILES, to write DOS.SYS and DOS.DUP on a *formatted* diskette. With some applications, such as AtariWriter, it is recommended that you write DOS files on *every* data diskette you plan to use to store files. This way, you can load DOS directly from your data diskettes as you load the application program. See your AtariWriter User's Guide and Section 3 of this manual.)

The DUPLICATE DISK function is sometimes confused with the COPY FILES function of DOS. The COPY FILES function copies only the files you specify from a source diskette. The DUPLICATE DISK function is more efficient when you want to make complete backup copies of data diskettes containing several files.

Using Write-Protect Tabs

Before duplicating a diskette or copying files from one diskette to another, it is a good idea to attach a write-protect tab to your source diskette. (Included with every package of diskettes that you buy, these tabs are adhesive but can easily be removed.) When folded over the notch in the edge of a diskette, a write-protect tab prevents your disk drive from writing information over any files that may already be there.

Particularly when you are using one disk drive to duplicate or copy files, you might mistakenly insert the source diskette when your disk drive is ready to write information onto the destination diskette. A write-protect tab on the source diskette prevents the drive from writing over (and destroying) your original data.



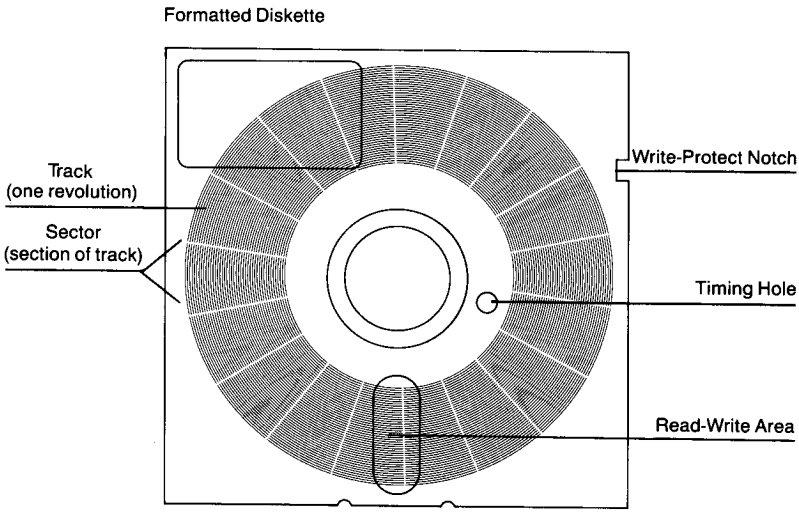
Formatting a Diskette

Unless they're *preformatted*, the diskettes you buy to store your files must be prepared to record information from your computer. This process is called *formatting, or initializing,* a diskette. Formatting organizes the surface of a diskette into *tracks* and *sectors* so that your computer can store and retrieve information on it in an orderly way (see illustration on the next page).

Caution: Formatting a diskette erases any information that may already be recorded on it. Never format your Master Diskette or a data diskette that contains valuable files.

The DOS 2.5 Menu offers you two options for formatting diskettes. When used with a 1050 Disk Drive, Option I., FORMAT DISK, will format the diskette in enhanced density. If DOS finds, when executing this function, that the drive you have specified for formatting is an 810, it will proceed to format the diskette in single density. Option P., FORMAT SINGLE, formats only in single density. It should be used when you want to format a diskette in single density on a 1050 Disk Drive.

If your system includes both an ATARI 1050 Disk Drive and an 810 Disk Drive, or if you have files created and stored using the earlier ATARI DOS 2.0S, see Appendix H. If you have files created with DOS 3, see Appendix L.



To Format a Diskette

To get acquainted with the formatting procedure, format two blank diskettes to use when doing the exercises described in the rest of this section. With the DOS Menu on your screen:

1. To format in enhanced density on a 1050 Disk Drive, type **I** and press Return; or to format in single density, type **P** and press Return. The prompt WHICH DRIVE TO FORMAT? appears.

2. Type the number of the disk drive in your system in which you want to format the diskette. The prompt TYPE "Y" TO FORMAT DISK n (where n is the number of the drive you specified) appears. This gives you a chance to make sure that the specified drive contains a diskette that you want to format — remember, formatting erases any files already contained on a diskette.

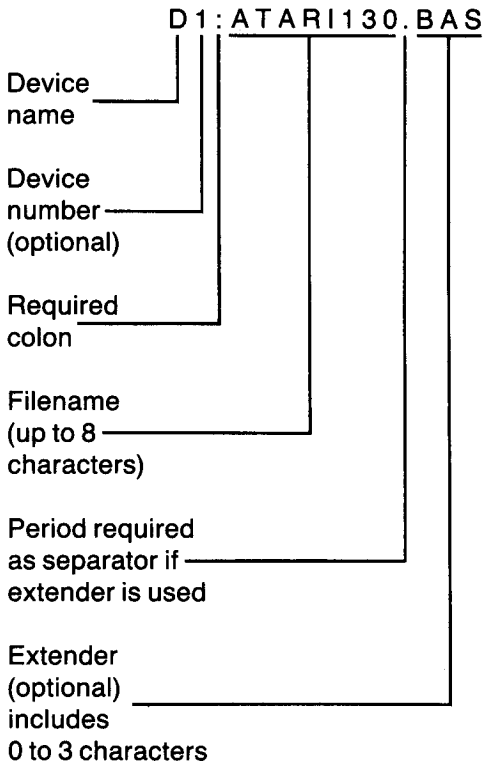
3. Place the diskette to be formatted in the drive you have specified. Type **Y** and press Return. The BUSY light goes on as the drive begins to format the diskette.

When the prompt SELECT ITEM OR [Return] FOR MENU appears, formatting is complete. You can now store files or write DOS Files on the formatted diskette.

Now repeat the procedure with another diskette.

Naming and Referring to Files

To manage your files with DOS, you have to give each file a distinctive *filename*. You also have to use a *device code* to tell your computer what part of your system — for example, Disk Drive 1 — you want to handle the file at any particular time. Taken together, the device code and the filename that you specify make up a *filespec* (short for "file specification"). Here is what a typical filespec looks like:



Device Codes

The **D1:** in the filespec illustration is a device code; there, it represents the part of your system you want DOS to use in carrying out a command. The **D** stands for disk drive, and the **1** specifies the number of the drive in your computer system. The colon (**:**) must always be used with a device code.

There are also codes for other devices that DOS can access or activate. The default display device, for example (see "Defaults," below) is **E:**, which stands for your TV screen or monitor; you could also use **P:**, which stands for a printer. See DISK DIRECTORY in Section 4 for some examples of using different display devices. **C:**, for cassette program recorder, is another device code that you might specify when using DOS, your disk drive, and a program recorder to manage files.

Defaults

For your convenience, *default* responses to several of its own prompts are built in to DOS 2.5.

Since most DOS users have only one disk drive, for example, one of the more convenient defaults in DOS is D1: — Disk Drive 1 of your computer system. You have seen how DOS defaults to this device when you press [Return] in response to a DIRECTORY-SEARCH SPEC, LIST FILE prompt. DOS also defaults to D1: as a source and destination device — provided that you press [Return] in response to the appropriate prompts — when you are using most other DOS functions.

If you have only one disk drive, you do not have to specify a drive number when entering the device code for it. DOS understands **D:** to mean Drive 1.

Filenames

Each file stored on a given diskette must have a unique filename; otherwise, your computer system wouldn't know which file you wanted to work with.

Filenames may be up to eight characters in length, followed if you like by a period and an *extender* of up to three characters.

Except for the period that separates the filename proper from the optional extender, all the characters in a filename must be letters or numbers, not punctuation marks or other symbols. So you could use —

these filenames:	but not these:
PROGRAM.6J	PROG.6J.BAS
ACCT4321	ACCOUNT4321
LETTER1	

If you try to enter an invalid filename, DOS refuses to accept it and displays ERROR 165 on your screen.

Extenders, sometimes called file *types*, can be useful when you're naming related but distinct files that you might want to manage as a group (see "Wild Cards"). For example, you could use BAS as an extender when naming all programs you write in ATARI BASIC—PROGRAM1.BAS, PROGRAM2.BAS, and so on. You might use LETTER.BUS to identify a business letter, and LETTER.SIS for a letter to your sister. If you were writing a book with a word processing program, you might store the various chapters on diskette as CHAPTER.1, CHAPTER.2, and so forth.

Wild Cards

In a game of poker, wild cards are valuable because they stand for any card you choose. Similarly, DOS recognizes special *wild card* symbols that can stand for any character or combination of characters in a filename. A major convenience, wild cards enable you to refer to a group of files rather than to each one individually.

The two wild cards recognized by DOS are the question mark (?), which stands for any single character, and the asterisk (*), which stands for any combination of characters in a filename proper or in an extender. Working with the following files, for example, you could also use PROGRAM?.* to specify *all* the program files, including PROGRAM1.PIL. The file LETTER?.BUS would refer to all the business letter files.

PROGRAM1.BAS	LETTER1.BUS
PROGRAM2.BAS	LETTER2.BUS
PROGRAM3.BAS	LETTER3.BUS
PROGRAM1.PIL	LETTER.SIS

You can now understand the entire default filespec used by DOS in a DISK DIRECTORY procedure. When you press [Return] in response to the DIRECTORY — SEARCH SPEC, LIST FILE prompt, DOS understands your response as D1:*.*. The D1:, as you know, is the default code (Disk Drive 1). The *.* stands for *all* files on the diskette you want to check.

Running a Cartridge From DOS

With ATARI BASIC (whether it's in cartridge form or built in your computer) or any other cartridge-based programming language, you can write your own programs to run on your ATARI Computer. And you can use programming language commands to store and retrieve your programs on diskette. But you need DOS to manage your program files in other ways — for example, to copy or erase them. By enabling you to shift control of your computer from DOS to a programming language cartridge, the RUN CARTRIDGE function on the DOS Menu allows you to use DOS and a programming language at the same time.

The procedures described in this section are based on the assumption that you're using ATARI BASIC (and that you've loaded BASIC along with DOS — see "Loading DOS," earlier in this section). However, the same procedures apply to using DOS with other programming language cartridges.

From BASIC to DOS and Back Again

When you load BASIC and DOS together, as explained earlier in this section, the READY prompt appears. Again, going from BASIC to DOS is easy — just type **DOS**, then press [Return]. The DOS Menu appears on your screen.

To go from DOS to BASIC (or any cartridge-based program), select **B**, RUN CARTRIDGE, from the DOS Menu, then press [Return].

Saving and Loading a BASIC Program

The BASIC computing language includes its own SAVE and LOAD instructions that you use to store and retrieve your programs on diskette. To try saving and loading a BASIC program, first select RUN CARTRIDGE from the DOS Menu. When the READY prompt appears, type the following program exactly as it appears (even the spaces and punctuation are crucial in programming). Press [Return] at the end of each line. If you make a mistake, you can press [Delete Bk Sp] to erase it.

```
10 PRINT "THIS LINE REPEATS ITSELF"
20 GOTO 10
```

You've just written a two-line BASIC program that tells your computer to print — display on your screen — THIS LINE REPEATS ITSELF and then go back to the previous instruction. You can imagine what's going to happen when this program runs.

Try it. Type **RUN**, then press [Return]. When you've seen enough, press [Break] to stop the program from running.

To store this (or any) BASIC program on diskette, you have to enter the appropriate BASIC command — **SAVE** — followed by a filespec that DOS understands. Replace your DOS diskette in Drive 1 with one of the initialized practice diskettes. Then type **SAVE "D:PROGRAM1.BAS"** and press [Return]. As your disk drive goes to work, your program is recorded on diskette.

Though it has now been stored on diskette, your program also remains in your computer's memory. To *see* the program load back into your computer, first type **NEW** and press [Return] to clear it from memory. Then press [Shift] and [Clear] at the same time to clear it from your screen. Next, type **LOAD "D:PROGRAM1.BAS"** and press [Return]. As your disk drive goes to work, the program is loaded back into your computer. Finally, type **LIST** and press [Return] to bring the program back up on your screen.

Note: The BASIC LOAD and SAVE commands are not the same as DOS LOAD and SAVE commands.

Now, though you have loaded it into your computer's memory, the program also remains on your data diskette. Once saved, your files remain on diskette until you use the DELETE FILE(S) function on the DOS Menu to erase them (see "Erasing Files" later in this section).

Copying Files

With the COPY FILE and DUPLICATE FILE options on the DOS Menu, you can copy your files from one diskette to another. You can also make a backup copy of a file on the same diskette as the original, provided you give the copy a different filename.

Note: You cannot copy the files that make up DOS — DOS.SYS and DUP.SYS — using either COPY FILE or DUPLICATE FILE. Instead, use option H., WRITE DOS FILES, to copy these files (see Section 3).

Creating Some Practice Files

When going through the previous section of this manual, you created a short program in BASIC and saved it on a data diskette as PROGRAM1.BAS. To learn how to use the copying functions, create a few more practice files.

Load DOS and BASIC, if necessary; if you have already done so and have the DOS Menu on your screen, select RUN CARTRIDGE. Insert your data diskette (the one containing your PROGRAM1.BAS file) in Drive 1 of your system. When the READY prompt appears, type the three SAVE commands below. Press [Return] after each line and wait while your disk drive saves the file before proceeding. In effect, these files are nothing more than filenames, but they're enough for you to work with as you learn how to copy files.

```
SAVE "D:PROGRAM2.BAS"
SAVE "D:PROGRAM3.BAS"
SAVE "D:PROGRAM1.PIL"
```

To Copy Files

Which of the two copying options you should use depends on how many disk drives you have and whether you want to copy a file from one diskette to another or on the same diskette.

If your system includes both an ATARI 1050 Disk Drive and an 810 Disk Drive, or if you have files created and stored using the earlier ATARI DOS 2.0S, see Appendix H. If you have DOS 3 files, see Appendix L.

From One Diskette to Another With One Disk Drive

Follow these steps to copy a file from one diskette to another using one disk drive:

1. With the DOS Menu on your screen, type **O** for DUPLICATE FILE and press [Return]. The prompt NAME OF FILE TO MOVE? appears.

2. Type the name of one of your practice files — say, **PROGRAM1.PIL** — and press [Return]. The prompt INSERT SOURCE DISK, TYPE RETURN appears.

3. Place the diskette containing the file you want to duplicate in the drive and press [Return]. The disk drive begins to read the specified file on your source diskette. Then DOS prompts you to INSERT DESTINATION DISK, TYPE RETURN.

4. Remove your source diskette from the disk drive and insert a *formatted* diskette, then press [Return]. DOS writes the file it has read from your source diskette to your destination diskette.

How many times DOS prompts you to insert your source and destination diskettes in your disk drive will depend on how large your original file is. When the prompt SELECT ITEM OR [Return] FOR MENU appears, the copying process is complete.

From One Diskette to Another With Two Disk Drives

1. With the DOS Menu on your screen, type **C** for COPY FILE and press [Return]. The prompt COPY — FROM, TO? appears.

2. Type the complete filespec for the file you want to copy, a comma, and the filespec for the copy itself. For your practice file PROGRAM1.PIL, type: **D1:PROGRAM1.PIL,D2:PROGRAM.PIL.**

3. Make sure that the diskette containing your original file is in Drive 1 and the diskette to which you are copying is in Drive 2. Then press Return.

DOS copies the specified file from the diskette in Drive 1 to the diskette in Drive 2. When the prompt SELECT ITEM OR Return FOR MENU appears, the copying process is complete.

Backing Up a File on the Same Diskette

Whether you have one drive or two, you use the COPY FILE option on the DOS Menu to make a backup copy of a file on the same diskette. Follow the steps outlined under "From One Diskette to Another With Two Disk Drives," but *type the same drive code for both the FROM and the TO filespecs and remember to give the file a different name in the TO filespec.* For example, if you are copying your practice file PROGRAM1.PIL on Drive 1, you might type **D1:PROGRAM1.PIL,D1:PROGRAM1. BAK.**

Using Wild Cards to Copy a Group of Files

Suppose you want to make backup copies of all four of your practice files — PROGRAM1.BAS, PROGRAM2.BAS, PROGRAM3. BAS, and PROGRAM1.PIL. You can use wild cards to copy all four at once — a time-saving alternative to copying them one at a time.

To use wild cards to copy all your practice files, follow the same procedure you use to copy one file on your system, but use wild cards when typing your FROM filespec (when using COPY FILE) or your NAME OF FILE TO MOVE (when using DUPLICATE FILE). To copy all four of your practice files, for example, you would type **PROGRAM?.*** as the name of the files to be copied — using the **?** to stand for the numbers in all four filenames and the * to stand for the extenders in all four filenames. If you are using the DUPLICATE FILE option, DOS will tell you as it copies each file.

See Section 3 for detailed examples of using wild cards when copying files with COPY FILE and DUPLICATE FILE.

Erasing Files

You can erase a file from a diskette with the DELETE FILE(S) function on the DOS Menu. Erasing out-of-date files, of course, opens up space on your data diskettes for storing more information. After a file is erased from a diskette, its filename disappears from the directory for that diskette.

Caution: Use the DELETE FILE(S) function with care — it may be permanent. Once you've erased a file, only under certain conditions can you use the DISKFIX.COM utility to get it back. (See the DISKFIX.COM section, Appendix L.)

For practice, try erasing the copy you made of your PROGRAM1.BAS file. Place the data diskette containing the file in your disk drive. Then follow these steps:

1. With the DOS Menu on your screen, type **D** and press Return. The prompt DELETE FILESPEC appears.

2. Type **D1:PROGRAM1.BAS** if your data diskette is in Drive 1, or **D2:PROGRAM1.BAS** if it is in Drive 2, then press Return. The prompt TYPE "Y" TO DELETE... appears — this is a verification prompt, allowing you a chance to change your mind about erasing the file.

3. Type **Y** and press Return to erase the file.

With wild cards in your filespec, you can erase as many files as you wish in one operation. To try this, erase the two remaining copies of files with the BAS extender on your diskette. Follow the same procedure you use to erase one file, but when your computer prompts you to enter the DELETE FILESPEC, type **D1:*.BAS** (or **D2:*.BAS** if you are using Drive 2).

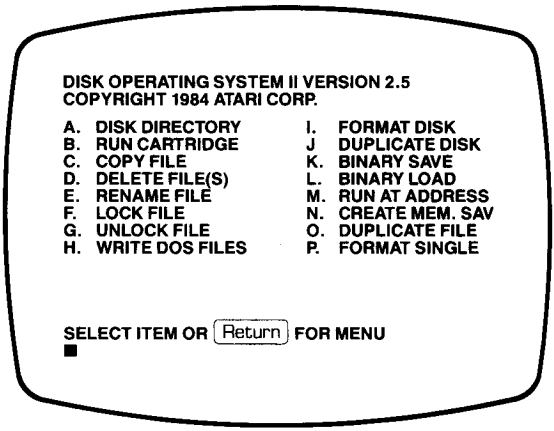
As DOS displays each filename matching your filespec, type **Y** and press Return to delete that file. When you want to erase several but not all of the files in a group that you specify with wild cards, simply type **N** and press Return as the name of each file that you want to preserve appears. This operation takes less time than going through the entire DELETE FILE(S) procedure for several individual files.

SECTION 3
SELECTING A DOS MENU OPTION

To select a DOS Menu option:

1. Load DOS into your computer. (From BASIC, you can go to DOS by typing **DOS** and pressing [Return].)

2. The Menu will appear on the screen, listing the 16 options available.



3. Type the letter corresponding to your selection and press [Return].

4. A prompt will appear, listing the parameters you need to supply before DOS can perform the function you have chosen. A parameter is additional information (sometimes optional) specifying how the command is to operate.

5. The prompt SELECT ITEM OR [Return] FOR MENU appears each time the computer system completes a request. If you choose to select another item, type the letter for the option you need and press [Return]. The bottom half of the screen will scroll upward to make room for the next option's prompts. If you press [Return], the screen will clear and redisplay the DOS Menu.

If your computer system includes both a 1050 and an 810 Disk Drive, or if you have and use files originally created and stored using DOS 2.0S, see Appendix H for detailed information on managing your files and diskettes with DOS 2.5. If you have files formatted with DOS 3, see Appendix L.

A. DISK DIRECTORY

A Disk Directory is a list of the files on a diskette, showing the filename, the extender (if any), and the number of sectors allocated to each file. It will either display a partial list or a complete list, depending on the parameters entered. Wild cards can be used in the parameters.

With the SELECT ITEM OR [Return] FOR MENU prompt on the screen, type **A** and press [Return]. The screen immediately displays this message:

```
DIRECTORY--SEARCH SPEC,LIST FILE?
```

If you press [Return] again, you will see a listing of *all* the files on the diskette, the size (in sectors) of each file, and the number of free sectors remaining on the diskette. The following example shows the files in the directory of your DOS 2.5 System Diskette.

```
SELECT ITEM OR [Return] FOR MENU
A [Return]
DIRECTORY--SEARCH SPEC,LIST FILE? [Return]
```

```
  DOS      SYS 037
  DUP      SYS 042
  RAMDISK  COM 009
  SETUP    COM 070
  COPY32   COM 056
  DISKFIX  COM 057
739 FREE SECTORS

SELECT ITEM OR [Return] FOR MENU
```

In the unlikely event that you use DOS 2.5 to store a file that occupies 1000 or more sectors on a diskette, the Disk Directory for that diskette will show the file as occupying only 999 sectors. Similarly, if there are more than 999 free sectors on a diskette, the directory will show 999 + FREE SECTORS. This is not an error; to maintain compatibility with programs written for DOS 2.0S, DOS 2.5 uses only three-digit numbers to indicate file sizes.

Also, when you store files, DOS 2.5 uses the lowest-numbered sectors on a diskette first. Since the earlier DOS 2.0S understands a maximum of only 719 sectors, any file created using DOS 2.5 that uses sectors numbered 720 or above cannot be properly accessed by DOS 2.0S. When you use option A, DOS 2.5 displays the names of such "extended" files enclosed by angle brackets. For example:

```
  *DOS    SYS  037
  *DUP    SYS  042
   FILE1  DAT  204
   FILE2  DAT  119
  <FILE3  DAT> 350
  <FILE4  DAT> 022
236 FREE SECTORS
```

In this directory listing, the files FILE3.DAT and FILE4.DAT, which use sectors of the diskette numbered 720 and above, are marked with angle brackets. Note that the size of a file does not determine whether it is so marked or not. Also note that files occupying sectors 720 or above will normally, but not necessarily, be the last files in a directory listing.

Parameters for the Disk Directory Option

As you can see from the entry prompt for the Disk Directory, this command has two parameters, SEARCH SPEC and LIST FILE.

At this prompt, you can choose to search for a single file, several files, or all files on the diskette you designate. If you do not indicate a specific disk drive, DOS will assume that you want to see the files on the diskette in Drive 1 (the default drive).

If you do not indicate a specific filespec, DOS will substitute the default values of D1:*.*,E: for the two parameters. The first default parameter, D1:*.*, tells DOS that you want to see a listing of *all* the files on the diskette currently inserted in Drive 1.

The second default parameter, E:, tells DOS that you want all this information to be displayed on the screen. Therefore, if you specify neither parameter and simply press [Return], DOS will list on the screen all filenames and file sizes stored on the diskette inserted in Drive 1.

If you have a printer, you can print a copy of the directory by using a comma (,) for the first parameter and a **P:** for the second parameter. In the example below, the data is printed for only one file, DOS.SYS.

1. Type **A** and press [Return].

2. After the directory entry prompt appears, type **DOS.SYS, P:** and press [Return].

3. If you have a printer and it is on, a partial directory for Drive 1 will be printed on the printer instead of the screen. If you do not have a printer (or it is not turned on), you will see an ERROR-138 displayed on the screen.

On the hard copy from the printer, you will see:

```
  DOS      SYS 037
739 FREE SECTORS
```

Each time the DISK DIRECTORY option completes a task, it displays a SELECT ITEM OR [Return] FOR MENU prompt.

The following examples illustrate several different ways you can use this option.

Note: When filenames are displayed, names and their extenders are not separated by a period. However, when you want to access a file, you *must* use a period between the filename and its extender.

Example 1: Lists all files from Drive 1 diskette with .SYS extender on the screen.

```
SELECT ITEM OR [Return] FOR MENU
A [Return]
DIRECTORY--SEARCH SPEC, LIST FILE?
*.SYS [Return]
```

Example 2: Lists all files on Drive 2 diskette on the line printer.

```
SELECT ITEM OR [Return] FOR MENU
A [Return]
DIRECTORY--SEARCH SPEC, LIST FILE?
D2:,P: [Return]
```

Example 3: Lists all three-letter filenames from the Drive 1 diskette that begin with EO.

```
SELECT ITEM OR [Return] FOR MENU
A [Return]
DIRECTORY--SEARCH SPEC, LIST FILE?
EO?.* [Return]
```

Note 1: When you use the DOS 2.5 Disk Directory option to list the files of a non-DOS disk, you may see a nonsensical listing for the file directory. Commercial adventure games and bootable game disks are examples of non-DOS disks. If this occurs you should turn your computer off, then on again, with your DOS System Disk in Drive 1 before performing any further DOS functions.

Note 2: You cannot use the DOS 2.5 Disk Directory Option to view the directory of a DOS 3 disk directly. See Appendix L, the COPY32.COM section for converting DOS 3 files.

B. RUN CARTRIDGE

Whenever you select B, DOS gives control of your ATARI Computer System to the inserted cartridge or to your computer's built-in BASIC, if any. If the BASIC cartridge is inserted or your computer does have built-in BASIC, the screen displays a READY prompt. If the Assembler Editor cartridge is inserted, the screen displays an EDIT prompt. If you have not inserted a cartridge and your computer does not have built-in BASIC, the message NO CARTRIDGE appears on the screen.

Warning: If you do not have a MEM.SAV file on your System Diskette (in Drive 1) when you entered DOS, you will find that any BASIC or assembly language program in memory before you entered DOS is now gone. Your program cannot be recovered now, unless you previously saved it on a diskette before you called DOS. This loss of your program file happens when using DOS 2.5 because you share the user program area with the Disk Utility Package stored in the DUP.SYS file. The sharing of RAM with DUP.SYS increases the amount of RAM available to the user.

Example:

```
SELECT ITEM OR [Return] FOR MENU
B [Return]
```

If the MEM.SAV file exists on the Drive 1 diskette, your BASIC or assembly language program will automatically be saved to the diskette when you type **DOS** and press [Return] and then reloaded into RAM when you return control to the cartridge. This is assuming that the diskette in Drive 1 is the same diskette that was there before you called DOS and that you did not invalidate MEM.SAV by your use of COPY FILE, DUPLICATE FILE, or DUPLICATE DISK. A prompt will appear to remind you that MEM.SAV can be invalidated if you try to use any of these commands (see N., CREATE MEM.SAV, in this section).

C. COPY FILE

Use this option if you have two or more disk drives and want to copy a file from a diskette in one disk drive to another diskette in a second disk drive.

If your computer system includes both an ATARI 1050 and an 810 Disk Drive, you can copy files from the 1050 to the 810 only if you first format the destination diskette on the 810 (or by using option P., FORMAT SINGLE, on the 1050). Then you can copy as many files as will fit on the single-density formatted diskette in your 810. You will not be able to copy any file that occupies more than 707 sectors, the capacity of a single-density diskette. If your computer system includes both a 1050 and an 810 Disk Drive, or if you have and use files originally created and stored using DOS 2.0S, see Appendix H for detailed information on managing your files and diskettes with DOS 2.5. See Appendix L if you have DOS 3 formatted diskettes.

There are two parameters associated with the COPY FILE command: FROM and TO. The first parameter, FROM, is usually a filespec, which may or may not contain wild cards. The use of wild cards in the first parameter gives you a very convenient way of copying a group of files from one disk drive to another (see Example 6). The **/A** option can be used with the second parameter to allow the FROM file to be appended to the TO file. The second parameter is generally a filespec, but can also be a destination device such as **E:** (screen), **P:** (printer), or **D:** (disk drive) (see Examples 3, 5, and 6).

COPY FILE can also be used to create a backup copy of a particular file on the *same* diskette with the same filename but a different extender, or even a completely different filename. If the file you are copying under a new name is made up of several files that have been appended (a "compound" file), the new version of the file will be compressed; i.e., it will take up fewer sectors than the original file from which it was copied.

If you attempt to copy a file when a MEM.SAV file is on your System Diskette, you will get a new prompt message. You will get the

new message after typing the source drive number (where the information is coming from) and the destination drive number (where the data is going). The message

```
TYPE ''Y'' IF OK TO USE PROGRAM AREA
CAUTION: A ''Y'' INVALIDATES MEM.SAV
```

appears to remind you that DOS can use all of the user program area to speed up the copy file process. A **Y** notifies DOS that you really don't care about your user program area or MEM.SAV file at this time — MEM.SAV will be invalidated. An **N** response tells the DOS that it cannot put anything into the user program area. It can only use a much smaller internal buffer to move your file. In other words, your file will still be copied when you give an **N** response, but it will take much longer.

You can also use this selection to copy the file listing to the screen (**E:**) or the printer (**P:**)

Caution 1: Do not append tokenized BASIC files, i.e., files stored with a SAVE command. Each tokenized file has its own symbol table, and only the first file will be loaded. However, you can merge two BASIC files stored with a LIST command, or two binary files created by the Assembler Editor cartridge or DOS. (Tokenized and untokenized files are explained in Section 4.)

Caution 2: Remember that in merge operations, files stored with a LIST command that have matching line numbers could cause the files to interfere with each other.

Example 1: Copies DOSEX.BAS from D1 to D2.

```
SELECT ITEM OR [Return] FOR MENU
C [Return]
COPY--FROM, TO?
D1:DOSEX.BAS,D2:DOSEX.BAS [Return]
```

Example 2: Creates backup copy of file on same diskette.

```
SELECT ITEM OR [Return] FOR MENU
C [Return]
COPY--FROM, TO?
D1:DOSEX.BAS,D1:DOSEX.BAK
```

Example 3: Displays the program listing on screen.

```
SELECT ITEM OR [Return] FOR MENU
C [Return]
COPY--FROM, TO?
D1:DOSEX.LST,E: [Return]
```

Example 4: Copies any succeeding data into a file named TEMP.DAT. Type data on screen that you want to be stored in TEMP.DAT file. [Control] 3 terminates entry of data.

```
SELECT ITEM OR [Return] FOR MENU
C [Return]
COPY--FROM, TO?
E:,D1:TEMP.DAT [Return]

PETER [Return]
BILL [Return]
RAY [Return]
STEVE [Return]
[Control] 3
```

Example 5: Lists the program listing DISEX.LST on the printer.

```
SELECT ITEM OR [Return] FOR MENU
C [Return]
COPY--FROM, TO?
D1:DISEX.LST,P:
```

Example 6: Copies all files from D1 to D2 except those having .SYS extender.

```
SELECT ITEM OR [Return] FOR MENU
C [Return]
COPY--FROM, TO?
*.*,D2: [Return]
```

Example 7: Appends PROG2 file on D1 to the PROG1 file.

```
SELECT ITEM OR [Return] FOR MENU
C [Return]
COPY--FROM, TO?
D1:PROG2,PROG1/A [Return]
```

D. DELETE FILE(S)

This option allows you to delete one or more files from a diskette and the disk directory. Wild cards can be used in the filespec names.

The verification prompt gives you a chance to change your mind about deleting a file. If you append the **/N** option (No Verification request) to the filespec entry, DOS will eliminate this verification step (see Example 3).

You can also delete all files on a diskette but leave the diskette formatted. Example 4 illustrates the steps for deleting all the existing files on the diskette in Drive 1. Note that the **/N** option is used in this example so the verification request does not need to be answered for each file on the diskette. It you try to delete a locked file, the screen will display ERROR-167 (File Locked).

Example 1: Deletes all files on Drive 2 diskette that begin with REM and that have a .BAS extender, with a verification prompt for each such file.

```
SELECT ITEM OR [Return] FOR MENU
D [Return]
DELETE FILE SPEC
D2:REM*.BAS [Return]
TYPE "Y" TO DELETE...
REM1.BAS?
Y [Return]
REMBAA.BAS?
Y [Return]
```

Example 2: Deletes single file, with a verification prompt.

```
SELECT ITEM OR [Return] FOR MENU
D [Return]
DELETE FILE SPEC
D:TEMP.DAT [Return]
TYPE "Y" TO DELETE...
TEMP.DAT?
Y [Return]
```

Example 3: File will be deleted without requesting verification.

```
SELECT ITEM OR [Return] FOR MENU
D [Return]
DELETE FILE SPEC
DOSEX.BAS/N [Return]
```

Example 4: Deletes all files from the Drive 1 diskette.

```
SELECT ITEM OR [Return] FOR MENU
D [Return]
DELETE FILE SPEC
*.*/N [Return]
```

E. RENAME FILE

This option allows you to change the name of one or more files. There are two parameters, OLD NAME and NEW, for this option. The parameter OLD NAME is always a complete filespec. If you do not specify a device number, the computer assumes D1: (the default). The NEW parameter refers simply to the new filename. The device number is automatically the device specified in the OLD NAME parameter. If there are any illegal characters in the NEW parameter, the name of the renamed file will consist of the characters up to, but not including, the illegal character.

You can use wild cards in both the first and second parameters (see Example 2). However, if you use one or more wild cards in the OLD NAME, they must at least be matched by number and position in the NEW filename. The following examples of legal and illegal filenames entered in response to the GIVE OLD NAME, NEW prompt illustrate this point:

```
LEGAL: TEST,NEW
       TEST.*,NEW.*
       *.DAT,*.BAK
       *.??1,*.??2
       TEST3.DAT,FILE?.*
       TEST3.DAT,FILE*.*
```

```
ILLEGAL: TEST.*,NEW
         TEST?,NEW
         TEST*.??1,TEST3.??2
```

Note that it is legal to use more wild cards in the NEW filename than in the OLD name; in such a case, the filename characters are copied from the OLD name unchanged. Thus the last legal examples shown above would both produce the same result as entering TEST3.DAT,FILE 3.DAT.

Remember that every file on a given diskette should have a unique filename. If you rename a single file without using any wildcards, DOS 2.5 allows you to give it any valid filename, including a name already assigned to a file on the same diskette. Then, if you try to work with one of the two files (for example, to delete it, lock it, etc.) DOS will act on both files. However, DOS 2.5 offers you a solution to this problem. If you find that you have two files with the same filename on one diskette, use the RENAME FILE option without any wild cards in either the OLD or NEW filename — DOS 2.5 will rename only the first file it finds that matches the OLD NAME you have specified. (See also the DISKFIX.COM section, Rename File By #, in Appendix L.)

If you attempt to rename a file on a write-protected diskette, an ERROR-144 (Device Done Error) will appear on the screen. If you try to rename a file that is not on the diskette, an ERROR-170 (File Not Found) appears. If the screen displays ERROR-167, it means that you tried to rename a locked file (see F., LOCK FILE).

Example 1: Changes the file on Drive 2 from TEMP.DAT to NAMES.DAT.

```
SELECT ITEM OR [Return] FOR MENU
E [Return]
RENAME - GIVE OLD NAME, NEW
D2:TEMP.DAT,NAMES.DAT [Return]
```

Example 2: All files on Drive 1 with extender 8KB have their extenders changed to .BAS

```
SELECT ITEM OR [Return] FOR MENU
E [Return]
RENAME - GIVE OLD NAME, NEW
*.8KB,*.BAS [Return]
```

F. LOCK FILE

Use this selection to write-protect a single file. A locked file cannot be written to, appended, renamed, or deleted. An ERROR-167 will result from trying to write to a locked file. You can use wild cards to lock several files at the same time.

A locked file will appear on the Disk Directory with an asterisk (*) preceding its name. *Do not* confuse this asterisk with a wild card.

Warning: If you lock any files on the Disk Directory and then format the diskette, the locked files *will still be obliterated.*

Example 1: Locks the DOS.SYS file on Drive 1.

```
SELECT ITEM OR [Return] FOR MENU
F [Return]
WHAT FILE TO LOCK?
DOS.SYS [Return]
```

Example 2: Locks all files on Drive 1 with an extender of .BAS.

```
SELECT ITEM OR [Return] FOR MENU
F [Return]
WHAT FILE TO LOCK?
D1:*.BAS [Return]
```

Example 3: Locks all files on Drive 1 that begin with T.

```
SELECT ITEM OR [Return] FOR MENU
F [Return]
WHAT FILE TO LOCK?
T*.* [Return]
```

Example 4: Locks all Drive 1 files.

```
SELECT ITEM OR [Return] FOR MENU
F [Return]
WHAT FILE TO LOCK?
*.* [Return]
```

G. UNLOCK FILE

Use this option to unlock a file or files you previously locked using option F. When you complete this option, the asterisk preceding the filename in the Disk Directory (to indicate the file was locked) will no longer appear when you execute a DISK DIRECTORY command (DOS Menu option A.). Wild cards can be used in the filespec names.

Example 1: Unlocks DOSEX.BAS file on Drive 1.

```
SELECT ITEM OR [Return] FOR MENU
G [Return]
WHAT FILE TO UNLOCK?
DOSEX.BAS [Return]
```

Example 2: Unlocks files beginning with the letter T on Drive 1.

```
SELECT ITEM OR [Return] FOR MENU
G [Return]
WHAT FILE TO UNLOCK?
T*.* [Return]
```

Example 3: Unlocks all five-letter files on Drive 1 beginning with PROB and having a .DAT extender.

```
SELECT ITEM OR [Return] FOR MENU
G [Return]
WHAT FILE TO UNLOCK?
PROB?.DAT [Return]
```

H. WRITE DOS FILES

To write DOS 2.5 files onto a diskette, you must have previously formatted the diskette using DOS 2.5 or DOS 2.0S (see I., FORMAT DISK). The diskette on which DOS is to be written can be inserted in the disk drive of your choice.

As soon as the DOS files have been written to the diskette, the screen is cleared and both the menu and the prompt SELECT ITEM OR [Return] FOR MENU are redisplayed.

If you try to write a new DOS file onto a diskette that has been write-protected, you will get an ERROR-144.

```
SELECT ITEM OR [Return] FOR MENU
H [Return]
DRIVE TO WRITE DOS FILES TO?
1 [Return]
TYPE "Y" TO WRITE DOS TO DRIVE 1.
Y [Return]
WRITING NEW DOS FILES
```

I. FORMAT DISK

Ordinarily, this option is used to format a diskette in enhanced density, provided you are using an ATARI 1050 Disk Drive. However, when executing this command DOS 2.5 will automatically switch to single-density formatting if it finds that the drive you are using is an ATARI 810 (which can format diskettes and manage information only in single density).

Option P. on the DOS 2.5 Menu, FORMAT SINGLE, can be used to "force" a 1050 Disk Drive to format a diskette in single density.

The diskette to be formatted can be blank or have files on it that you no longer want. Formatting writes information on the diskette that allows data to be stored and retrieved.

A diskette formatted with this option using a 1050 Disk Drive is capable of storing information in 1023 sectors, while a diskette formatted in single density can store information in 719 sectors. However, the formatting process itself reserves some sectors for the exclusive use of DOS. So a diskette newly formatted in single density will show (in a directory listing) only 707 FREE SECTORS, and in enhanced density 999 + FREE SECTORS. There are actually 1010 sectors available in enhanced density, but DOS 2.5 uses three-digit numbers only for the sake of compatibility with DOS 2.0S.

If your computer system includes both a 1050 and an 810 Disk Drive, or if you have and use files originally created and stored using DOS 2.0S, see Appendix H for detailed information on managing your files and diskettes with DOS 2.5. If you want to use files originally formatted with DOS 3, see Appendix L. Remember to label your diskettes clearly with the version of DOS used to format each one.

The example below illustrates Drive 1 as the drive to be formatted; however, you can specify any drive. If you try to format a diskette containing bad sectors, the screen will display an ERROR-173 (Bad Sectors at Format Time). If DOS gets a message from the disk drive that the diskette has bad sectors, it will keep trying to format the diskette. If this happens, it may take up to 15 minutes trying to format a diskette before returning an ERROR-173.

If a diskette is new and has bad sectors, you should return it to the supplier for exchange.

```
SELECT ITEM OR [Return] FOR MENU
I [Return]
WHICH DRIVE TO FORMAT?
1 [Return]
TYPE "Y" TO FORMAT DISK 1
Y [Return]
```

Warning: Formatting a diskette always destroys all files and format previously existing on the diskette.

J. DUPLICATE DISK

Use this menu option to create an exact duplicate of any diskette. You can use this option with a single disk drive by manually swapping source (diskette with files on it) and destination (diskette on which you are putting files) until the duplication process is complete. You can also use this option with multiple disk drive systems by inserting source and destination diskettes in two separate drives and allowing the duplication process to proceed automatically.

Unlike the same function of the earlier DOS 2.0S, DUPLICATE DISK in DOS 2.5 will format your destination diskette. However, keep in mind that you cannot duplicate an enhanced-density diskette using an ATARI 810 Disk Drive as your destination drive. If your computer system includes both a 1050 and an 810 Disk Drive, or if you have and use files originally created and stored using DOS 2.0S, see Appendix H for detailed information on managing your files and diskettes with DOS 2.5. See Appendix L if you want to use files formatted with DOS 3.

The duplication process is a sector-by-sector copying technique. This means that not only are all your files copied from the source to the destination diskette, but they are also located in the same sector number on both diskettes. The directory of the source diskette is also copied onto the destination diskette. For this reason, any files previously stored on the destination diskette will have been destroyed when the duplication process is complete. So if you use an old diskette for the destination diskette, be sure that none of the files on it are valuable.

You should always save BASIC or assembly language programs that are currently in RAM before attempting to duplicate a diskette. There is no internal buffer for DUPLICATE DISK as there is for the COPY FILE command, and MEM.SAV (if in use) will be invalidated if you give DOS permission to proceed (and to use the program area). The DUPLICATE DISK option always uses the program area (where a RAM-resident BASIC program is stored) as a buffer for moving the files on the source diskette to the destination diskette.

Duplication Using a Single Disk Drive

In a single disk drive system, the source and destination drives are both Drive 1 (see example).

Always write-protect your source diskette as a safety measure. Then, if it is accidentally inserted in place of the destination diskette, the screen will display an ERROR-144, and your source diskette will still be intact.

If you type any character other than Y and press [Return] in response to the TYPE "Y" IF OK TO USE PROGRAM AREA message, the program aborts and the SELECT ITEM OR [Return] FOR MENU prompt appears on the screen.

Here is an example of duplication using a single disk drive:

```
SELECT ITEM OR [Return] FOR MENU
J [Return]
DUP DISK-SOURCE, DEST DRIVES?
1,1 [Return]
INSERT SOURCE DISK, TYPE RETURN
[Return]
INSERT DESTINATION DISK, TYPE RETURN
[Return]
```

Note: The number of times the DUP program prompts you to insert the source and destination diskettes depends on the number and size of the file(s) to be duplicated for a given system and the amount of RAM in the system.

Duplication Using Multiple Disk Drives

If you are using both the ATARI 810 and ATARI 1050 Disk Drives, make sure you distinguish between files stored using the single-density and enhanced-density formats when labeling the diskettes. This will keep you from using them in the wrong disk drive.

For a multiple disk drive system, it is also necessary to save a RAM-resident BASIC program, as the user's program area will be altered and MEM.SAV will be invalidated. Notice that the source diskette is inserted in Drive 1 and the destination diskette in Drive 2.

This process can take several minutes if the source diskette is almost full.

```
SELECT ITEM OR [Return] FOR MENU
J [Return]
DUP DISK-SOURCE, DEST DRIVES
1,2 [Return]
INSERT BOTH DISKS, TYPE RETURN
[Return]
```

K. BINARY SAVE

Note: This option will probably not be used by a beginning ATARI Computer user. Unless you understand hexadecimal numbers and have some knowledge of assembly language, you may not wish to read the information beyond the first example.

Use this Menu selection to save the contents of memory locations in object file (binary) format. Programs written using the Assembler Editor cartridge also have this format. The parameters for this selection—START, END, INIT, RUN—are hexadecimal numbers. The START and END addresses are required parameters for any binary file or program. The INIT (initialize) and RUN addresses are optional parameters that allow you to make any program execute on loading. See Examples 2, 3, and 4 below.

In the example below, a file to be called BINFIL.OBJ with the starting address 3C00 and the ending address 5BFF is saved on a diskette in Drive 1.

Example 1:

```
SELECT ITEM OR [Return] FOR MENU
K [Return]
SAVE-GIVE FILE,START,END(,INIT,RUN)
BINFIL.OBJ,3C00,5BFF [Return]
```

Advanced User Information About Optional Parameters

All binary files, like those you would create with the BINARY SAVE option or with the Assembler Editor cartridge, have a common six-byte header that precedes the file. From the header data shown in the table, you can easily pick out the starting address and ending address that was used in the example above.

The two optional parameters, INIT and RUN, offer the means to make a binary assembly language file execute automatically after loading. A file that makes use of either or both of these address parameters is called a "load-and-go" file. A file that does not contain data for these parameters is called a "load" file,

since it loads into the computer but will not execute until a RUN AT ADDRESS command is given.

Header Byte #	Decimal Number	Hex Number	Description
#1	255	FF	Identification code for
#2	255	FF	binary load file
#3	0	00	Starting address (LSB)
#4	60	3C	(MSB)
#5	255	FF	Ending address (LSB)
#6	91	5B	(MSB)
			File data segment contains 8192 (Dec) bytes of data.

In general, the RUN address parameter defines the point in a program where execution will begin as soon as a whole file is loaded into RAM (i.e., when End of File is reached). For this reason there can only be one effective RUN address even if a file is a compound file. For example, a file could be made up of several small files appended together with each of the original small files having their own RUN address. In this case, only the last RUN address to be loaded would execute.*

If an INIT address is specified, then as soon as the actual address gets loaded into RAM, the code that it points to will be executed. This is true even if the file is made up of several load-and-go files appended together. In such a case, each load-and-go segment that has an INIT address specified will be executed when the INIT address is loaded. Thus, each segment would load and be executed before the next segment would be loaded, etc.** Execution of code pointed to by any INIT address always precedes the execution of any code pointed to by a RUN address.

Files created by the Assembler Editor cartridge using the load-and-go option can be stored in the desired INIT and RUN addresses in your code followed by the code to be controlled. The RUN address is always stored in Locations 2E0 (LOW) and 2E1

*An RTS (RETURN) at the end of a program will always return control to DOS.
**Each code segment must end with an RTS (RETURN) if the next segment is to be loaded or, if desired, returned to DOS control.

(HIGH) Hex. The INIT address is always stored in Locations 2E2 (LOW) and 2E3 (HIGH) Hex. Remember, the INIT address is executed as soon as it is loaded, so the code that it points to must have been previously loaded.

Note: IOCB #1 is open during the execution of code pointed to by any INIT address. For this reason it is not available and must not be tampered with by the user program being executed.

Using Binary Save With Optional Parameters

The example below illustrates an assembly language program that uses a data area that must be initialized before the main program can use it. Suppose the initialization code resides from address 4000 (Hex) to 41FF (Hex) and the main program resides between 4200 (Hex) and 4FFF (Hex). For purposes of illustration, assume that both the initialization code and main program contain executable code and the initialization code ends with an RTS (RETURN).

In the following example, it is assumed that the program LAGPRG.OBJ is already in memory.

Example 2:

```
SELECT ITEM OR [Return] FOR MENU
K [Return]
SAVE-GIVE FILE,START,END(,INIT,RUN)
LAGPRG.OBJ,4000,4FFF,4000,4200[Return]
```

The following events will occur on loading this file into memory:

1. Memory from 4000 to 4FFF will be filled with the program.

2. The INIT address 4000 (Hex) is stored in Memory Locations 2E2 and 2E3 (Hex).

3. Initialization program from 4000 to 41FF will execute.

4. The RUN address 4200 (Hex) is stored in Memory Locations 2E0 and 2E1 (Hex).

5. Main program from 4200 to 4FFF begins to execute and will continue to do so until a RETURN (RTS) is executed, or a [System Reset] or [Break] occurs.

In the case of compound files, the result is more complicated, depending on how the now appended files were created. The next section illustrates several cases where files have been appended.

Structure of a Compound Binary File

Before considering the next example, look at the structure of a compound file. A compound file is constructed of various binary files that have been appended together. You can create compound files in one of two ways. One way is to use the COPY FILE option with its append option. A compound file created with this command is not compatible with the Assembler Editor loader, although it can be loaded using the BINARY LOAD option of DOS. If compatibility with the Assembler Editor cartridge is desired, an alternate way to create a compound file is to use the BINARY SAVE option. (To do this, you must tell DOS the name of the file you are appending to, followed by the **/A** option—see Example 5.) The two types of files are illustrated in Appendix I. The only real difference is that the FFFF (Hex) identification code is included with every segment when a compound file is created using COPY FILE.

When BINARY SAVE is used, the additional identification codes for each segment (after the first one) are *not* included in the final file. This is the only form of compound file that is compatible with the LOAD command of the Assembler Editor cartridge. The BINARY LOAD option of DOS, however, is compatible with both types of compound files.

Now consider what happens when a compound file like this is loaded—supposing various INIT and RUN addresses were specified for each of these files before they were appended. (It will help you to think of the INIT and RUN addresses as being part of the data in each segment, which they essentially are.)

Example 3:

Suppose you have three files, each of which has a RUN address but no INIT address included in its data. This example shows one way a file of this type might be created.

```
SELECT ITEM OR [Return] FOR MENU
K [Return]
SAVE-GIVE FILE,START,END(,INIT,RUN)
PART1.OBJ,2000,21FF,,2000 [Return]

SELECT ITEM OR [Return] FOR MENU
K [Return]
SAVE-GIVE FILE,START,END(,INIT,RUN)
PART2.OBJ,2200,23FF,,2200 [Return]
```

The other two files, PART2.OBJ and PART3.OBJ, that are created the same way as PART1.OBJ can then be merged into WHOLE.OBJ by using the BINARY SAVE or COPY FILE option with the append option. When this new file is loaded—

1. PART1.OBJ loads, but does not execute (no INIT).

2. RUN address for PART1.OBJ is stored in 2E0 and 2E1.

3. PART2.OBJ loads, but does not execute (no INIT).

4. RUN address for PART2.OBJ is stored in 2E0 and 2E1, which overwrites PART1.OBJ RUN address.

5. PART3.OBJ loads, but does not execute (no INIT).

6. RUN address for PART3.OBJ is stored in 2E0 and 2E1, which overwrites PART2.OBJ RUN address.

7. Execution begins at RUN address of PART3.OBJ, since you are now at the end of the file.

Example 4:

For another example of a compound file, consider a three-segment file, BIGFILE.OBJ. Suppose each segment loads into a different area of memory and that—

SEG1.OBJ has an INIT address, but no RUN address;
SEG2.OBJ has no INIT or RUN address;
SEG3.OBJ has an INIT address, and a RUN address for SEG2.OBJ and, in addition, is loaded on top of SEG1.OBJ.

When BIGFILE.OBJ is loaded, the following events occur:

1. SEG1.OBJ is loaded.

2. SEG1.OBJ executes starting at its INIT address.

3. SEG2.OBJ is loaded.

4. SEG3.OBJ is loaded on top of SEG1.OBJ.

5. SEG3.OBJ executes starting at its INIT address.

6. SEG2.OBJ executes starting at the RUN address specified in SEG3.OBJ.

Clearly, this option gives you great power and flexibility for creating large files that load and execute immediately.

Example 5:

To convert an existing load-only file to a load-and-go file, you can load the file into memory and then save it under a new filename using the BINARY SAVE Menu option. This poses some problems, as you can sometimes forget the final address the file occupies, or the file could be compounded with the segments not necessarily consecutive in memory. Therefore, the new file would take up more space on the diskette than the old, etc. You can avoid these problems by using the procedure shown in the following example. This example illustrates a load file with a run address of 4000 Hex that is changed to a load-and-go file.

In the example, a one-byte file located at FF00 (in the OSROM) is appended to the end of your file LOADFIL.OBJ. Since this file's run address is the same as the address at which your load file normally runs, your load file begins execution as soon as the entire appended file is loaded into RAM.

```
SELECT ITEM OR [Return] FOR MENU
K [Return]
SAVE-GIVE FILE,START,END(,INIT,RUN)
LOADFIL.OBJ/A,FF00,FF00,,4000
```

L. BINARY LOAD

Note: This instruction will probably not be used by a beginning ATARI Computer user.

Use this selection to load into RAM an assembly language (binary) file that was previously saved with menu option K. or created by the Assembler Editor cartridge. If the RUN address or INIT address was appended to the file in Locations 2E0 and 2E1 or 2E2 and 2E3, the file will automatically run after being entered. In a load-and-go file, INIT and RUN addresses are ignored when you type **/N** after the filename (see Example 1). The file can then be run using the RUN AT ADDRESS menu option.

To execute a file that has no appended RUN or INIT address, see the next menu option, M., RUN AT ADDRESS.

Example 1:

```
SELECT ITEM OR [Return] FOR MENU
L [Return]
LOAD FROM WHAT FILE?
MYFILE.OBJ/N [Return]
```

The use of this option without the **/N** option is shown in Example 2. Since this file had the starting address in Locations 2E0 and 2E1 appended to it (see Example 1 for K. BINARY SAVE), the file will begin executing as soon as the load is complete.

Example 2:

```
SELECT ITEM OR [Return] FOR MENU
L [Return]
LOAD FROM WHAT FILE?
BINFIL.OBJ [Return]
```

Example 3 illustrates a file called MACHL.OBJ that does not have a RUN address or an INIT address. In this case, the SELECT ITEM OR [Return] FOR MENU prompt will display on the screen as soon as the file finishes loading.

Example 3:

```
SELECT ITEM OR [Return] FOR MENU
L [Return]
LOAD FROM WHAT FILE?
MACHL.OBJ [Return]
```

M. RUN AT ADDRESS

Note: This instruction will probably not be used by a beginning ATARI Computer user.

Use this selection to enter the hexadecimal starting address of an object file program after you have loaded it into RAM with the BINARY LOAD selection. This selection is used when the starting address has not been appended to the object file.

In the example below, the instructions at hexadecimal Location 3000 will begin executing. Be very careful when entering these hexadecimal address locations. If you enter an address that does not contain executable code, it will create problems. For example, you could lock up the system, making it necessary for you to reboot.

```
SELECT ITEM OR [Return] FOR MENU
M [Return]
RUN FROM WHAT ADDRESS?
3000 [Return]
```

N. CREATE MEM.SAV

This option allows you to create a file on diskette called MEM.SAV into which the contents of lower user memory are saved whenever you call DOS. When you type **DOS** [Return], the computer saves the contents of lower user memory, including the RAM-resident user program (if any), in the MEM.SAV file before it brings the diskette file DUP.SYS into RAM. When you have finished using the DOS options, you simply return control to the cartridge by typing **B** [Return], and MEM.SAV will automatically reload the portion of your program that was replaced by DUP.SYS into RAM. If you are not using a cartridge, typing **B** has no effect. You will have to respond to the SELECT ITEM OR [Return] FOR MENU prompt.

You must be careful not to allow DOS to use all of user memory when you want the COPY FILE, DUPLICATE FILE, or DUPLICATE DISK options for saving the existing data. DOS does not know if *all* or only part of your program has been saved in MEM.SAV. When DOS utilizes all of user memory, it automatically invalidates the MEM.SAV file. If this occurs, your program will not be reloaded when control is returned to the cartridge.

Here are the steps for creating a MEM.SAV file on a diskette inserted in Drive 1. Note that MEM.SAV files can only by created on a diskette in Drive 1.

```
SELECT ITEM OR [Return] FOR MENU
N [Return]
TYPE "Y" TO CREATE MEM.SAV
Y [Return]
```

If you attempt to use this option to create a MEM.SAV file on a diskette that already has a MEM.SAV file, the screen will display the message MEM.SAV FILE ALREADY EXISTS and follow it with the prompt SELECT ITEM OR [Return] FOR MENU.

Why Have a MEM.SAV File?

This special file allows you to save your RAM-resident program temporarily in a special file on diskette. To be effective, MEM.SAV (which requires 45 sectors) must be on the diskette

inserted in Drive 1. This diskette must not be write-protected if MEM.SAV is to work. Once MEM.SAV exists on your diskette, then the area of user memory to be overwritten by DUP.SYS will be stored in MEM.SAV every time DOS is called. Essentially, you are performing a "swap-contents" operation, thereby "expanding" your user program area. When you return control of the computer system to the cartridge, the DUP.SYS file is in turn overwritten as the contents of MEM.SAV are loaded back into RAM automatically.

If you are working on a BASIC program and need to return to DOS for some reason, you can do so using MEM.SAV without having to save your program to diskette and reenter it. When you finish using DOS and return control of the computer system to BASIC, the MEM.SAV file is automatically reloaded into memory and your BASIC program is restored into user program memory.

Note: MEM.SAV is most time efficient when used with the ATARI 130XE and the RAMDISK option (see Appendix K). In other cases, it may by faster to SAVE your program before calling up DOS, then LOAD it upon returning to BASIC.

Here is an example of MEM.SAV usage:

1. Type **LOAD "D:MYPROG. BAS"** Return.

2. Edit your program and then type **RUN** Return. It works, and you want to RENAME the original file to keep as a backup copy.

3. Type **SAVE "D:MYPROG.NEW"**, then press Return.

4. Type **DOS** Return.

5. Next type **E** (for RENAME FILE) **MYPROG.BAS, MYPROG. OLD,** and press Return.

6. Type **E** and press Return. Then type **MYPROG.NEW, MY PROG.BAS** and press Return again.

7. To return to BASIC, type **B** (for RUN CARTRIDGE).

Using MEM.SAV to Write Assembly Language Programs

The MEM.SAV file also allows you to write assembly language programs (or load in binary data) that share the user program area with DUP.SYS. This means you are free to write programs or load data in the area from LOMEM (which fluctuates with the number of drives in the system and the number of files that can be open concurrently) to HIMEM (which fluctuates depending on which graphics mode you are in). See Appendix D, Memory Map.

Example:

Suppose you have a binary file you want to execute automatically as soon as it is loaded. This type of file is called a load-and-go file. The run address is already programmed into such a file and you will not need to select the RUN AT ADDRESS option. In this case, it is not necessary to have a MEM.SAV file on your diskette. Since the file is load-and-go, it will simply load and then begin to execute. The safest way to get back to DOS is to reboot your computer. If you have not overwritten the DUP.SYS program during the execution of your binary file, then you can recover by simply executing a RETURN (RTS) in your program. If a binary file overwrites DUP.SYS during the time it is loading, DOS will keep track of this fact and will automatically reload and execute DUP.SYS after the RETURN in your program is executed.

Warning: If the execution of your load-and-go file writes into any areas below LOMEM used by DOS.SYS or DUP.SYS or a RAM area used by the Operating System, the RETURN (RTS) from your program may leave the computer in an undefined state. Should this occur, you may have to power up the computer again to recover.

Using MEM.SAV to Load Binary Files

This section deals with loading a binary file that is not to be executed at the same time it is loaded, or loading a file that contains data for another program. If your LOAD file does not overlay any part of the DUP.SYS area, then a MEM.SAV file is not required.

If your LOAD file overlays any part of the DUP.SYS file, you must have a MEM.SAV file on the diskette in Drive 1 if the load is to be successful. If you do have MEM.SAV, the following actions take place after you execute LOAD BINARY FILE option:

1. You use the LOAD BINARY FILE selection to load your file.
2. Your original MEM.SAV is loaded from disk into memory, overlaying and invalidating DUP.SYS.
3. Your file is loaded on top of the original MEM.SAV, modifying part or all of the original MEM.SAV file.
4. Your new MEM.SAV file in RAM is saved in the MEM.SAV area on the diskette.
5. DUP.SYS is reloaded from diskette into memory.
6. You remain in DOS until you choose to:

RUN CARTRIDGE	at which time your file is loaded into memory from MEM.SAV and you come up under the control of your BASIC or Assembly Language cartridge;
RUN AT ADDRESS	at which time your file is loaded into memory from MEM.SAV and you begin execution of whatever code is at the address you specified; or
LOAD BINARY FILE	where you wish to load a load-and-go file. In this instance, if the new file also overlays a part of DUP.SYS, but not the original file, then both MEM.SAV and your new file will now be in memory when the load is complete. If the new file does not overlay DUP.SYS at all, then the load will complete with only the new file loaded into RAM. Since the new file is a load-and-go and loaded whether DUP.SYS is overlaid or not, you will come up under the control of this file until a RETURN (RTS) is executed.

Note: If you wish to have two files in memory simultaneously, one of which resides wholly or in part in the DUP.SYS area and the other of which resides wholly outside of the DUP.SYS area, simply merge the two files into one file, then load the newly merged file.

O. DUPLICATE FILE

This option (shown in Example 1) is used if you have only one disk drive and want to copy a file from one diskette to another. Remember that a single disk drive must always be set up as Drive 1. Since there is only one disk drive, you must manually insert and remove the source and destination diskettes. If a file is very long, you may have to alternate the source and destination diskettes several times before the duplication process is complete.

Example 1:

```
SELECT ITEM OR [Return] FOR MENU
O [Return]
NAME OF FILE TO MOVE?
DOSEX.BAS [Return]
INSERT SOURCE DISK,TYPE RETURN
[Return]
INSERT DESTINATION DISK,TYPE RETURN
[Return]
```

Wild cards are available with this option. In Example 2 you will notice that even when using wild cards, your files are still copied one at a time. You will have to alternate diskettes at least once for each file that you want to copy.

Example 2 illustrates using a wild card to copy files having five-letter filenames beginning with TEST from one diskette to another. It is assumed that the source diskette has only two files with names that satisfy TEST?.

Example 2:

```
SELECT ITEM OR [Return] FOR MENU
0 [Return]
NAME OF FILE TO MOVE?
TEST? [Return]
INSERT SOURCE DISK, TYPE RETURN
[Return]

  COPYING---D1:TEST1
INSERT DESTINATION DISK, TYPE RETURN
[Return]

INSERT SOURCE DISK, TYPE RETURN
[Return]

  COPYING---D1:TEST2
INSERT DESTINATION DISK, TYPE RETURN
[Return]

INSERT SOURCE DISK, TYPE RETURN
[Return]
```

In Example 3 both the filenames and extenders have been replaced with wild cards. DOS will therefore copy all files except those that have an extender of .SYS. It is assumed that only three files are to be copied: MEM.SAV, TEST1, and TEST2.

Note: Creating MEM.SAV on the new disk is faster than copying it.

Example 3:

```
SELECT ITEM OR [Return] FOR MENU
0 [Return]
NAME OF FILE TO MOVE?
*.* [Return]
INSERT SOURCE DISK, TYPE RETURN
[Return]
```

```
  COPYING-D1:MEM.SAV
INSERT DESTINATION DISK,TYPE RETURN
Return

INSERT SOURCE DISK,TYPE RETURN
Return
  COPYING---D1:TEST1
INSERT DESTINATION DISK,TYPE RETURN
Return

INSERT SOURCE DISK,TYPE RETURN
Return

  COPYING---D1:TEST2
INSERT DESTINATION DISK,TYPE RETURN
Return

INSERT SOURCE DISK,TYPE RETURN
Return
```

P. FORMAT SINGLE

Use this option to format a diskette in single density. If you have only an ATARI 1050 Disk Drive, you *must* use this option, rather than option I., FORMAT DISK, on the DOS Menu to format a diskette in single density.

Remember that formatting will erase any files or other data already stored on a diskette.

The procedure for using FORMAT SINGLE is just like that for using FORMAT DISK:

```
SELECT ITEM OR Return FOR MENU
P Return
WHICH DRIVE TO FORMAT?
1 Return
TYPE "Y" TO FORMAT DISK 1
Y Return
```

SECTION 4
USING BASIC COMMANDS WITH DOS 2.5

BASIC Commands Used With DOS

Before learning about the BASIC commands used with DOS 2.5, you need to know how the commands will act on programs being stored and retrieved. The following paragraphs explain the two types of files that can contain BASIC programs.

Tokenized and Untokenized Files

The first type of file, called "untokenized," contains standard ATASCII text characters, so it looks like a printout of a BASIC program. These programs do not retain their symbol tables each time they are loaded and saved. The symbol table associates the variable name with the memory location where the values for that variable are stored. To store and retrieve a file in its untokenized form, you use the LIST and ENTER commands.

A tokenized file is a condensed version of a BASIC program. It has one-byte tokens instead of the ATASCII characters to represent the BASIC commands. You can move tokenized programs back and forth between the disk drive and computer memory using the SAVE and LOAD commands. Tokenized versions of a file are generally shorter and load faster than untokenized versions. For this reason, many programmers prefer to store their programs in the tokenized form.

Usually during program development the symbol table becomes cluttered with unused variable names. If you use a variable in a program line and then change the variable name or delete the line, the original variable name remains in the symbol table. Use the following procedure to clear the symbol of all unused names.

1. LOAD your program (see LOAD below).

2. LIST it to disk (see LIST below).

3. Type **NEW** [Return].

4. ENTER your program from disk (see ENTER). The symbol table now contains only those variable names present in the program.

LOAD (LO.)

Format: LOAD filespec
Example: `LOAD "D1:DOSEX.BAS"` [Return]

This command is used to load a file from a particular diskette in a disk drive into the user program RAM area. Before you can use this command to load a file called DOSEX.BAS, the file (DOSEX-.BAS) must have been previously saved using the BASIC command SAVE. This command loads only a tokenized version of a program.

This command can also be used in "chaining" programs. If a program is too big to run in your available RAM, you can use the LOAD command to spread the program across two files. Simply type the LOAD statement as the last line of the first program file, as shown in the example following this paragraph. When the program encounters the LOAD statement, it will automatically read in the next part of the program from the diskette. The second program file must be able to stand alone without depending on any variables or data in RAM from the first program file. The loaded program will not execute until you type **RUN** and press [Return], at which time the previous program and any variables will be cleared (see RUN for another example).

```
100 REM CHAIN PROGRAM
110 LOAD "D1:CHAIN.BAS"
```

SAVE (S.)

Format: SAVE filespec
Example: `SAVE "D1:EXAMP2.BAS"` [Return]

This command causes the computer system to save a program on diskette with the filespec name designated in the command. SAVE is the complement of LOAD and stores programs in tokenized form.

LIST (L.)

Formats: LIST filespec ,lineno ,lineno
device

Examples:
```
LIST "D:DATFIL.LST"
LIST "P:"
LIST "P:",10,100
```

One use of the LIST command in BASIC is very similar to the SAVE command: it can take a program from user program RAM and store it on a particular drive with any name you want to assign it (illustrated by the first example). However, the program is stored in standard ATASCII text and not as tokens. Differences in the formatting of data storage also allow LIST to be much more flexible than SAVE. As shown in the above format examples, you can specify a single device (e.g., P:, E:, C:, D:, D2:, etc.), or you can specify line numbers to be listed to a designated device (e.g., "P:", 100, 200).

ENTER (E.)

Format: ENTER filespec
Example:
```
ENTER "D:LIST2.LST"
```

This command causes the computer to move a file on diskette with the referenced filespec into RAM. The program is entered in untokenized form and is interpreted as the data is received. ENTER, unlike LOAD, will not destroy a RAM-resident BASIC program, but will merge the RAM-resident program and the disk file being loaded. If there are duplicate line numbers in the two programs, the line in the program being entered will replace the same line in the RAM-resident program.

RUN

Format: RUN filespec
Example: `RUN "D2:MYFILE.BAS"`

This command causes the computer to LOAD and RUN the designated filespec. It is a combination of the two commands LOAD and RUN. However, the RUN command can be used only with tokenized files. Therefore, you cannot execute a RUN "D2: LIST.LST" command.

To chain programs and cause a second segment of a file to load and run automatically, you can use a RUN "D: filespec" as the last line of the first segment. However, the second program must be able to stand alone without depending on any variables or data in RAM from the first program. Before running the first segment, make sure that you have saved it on a diskette, because the RUN statement will wipe out your RAM-resident first segment when the second segment is loaded.

Input/Output Control Blocks

An I/O (input/output) operation is controlled by an I/O Control Block (IOCB). An IOCB is a specification of the I/O operation, consisting of the type of I/O, the buffer length, the buffer address, and two more auxiliary control variables of which the second is usually 0. ATARI BASIC sets up eight IOCBs and dedicates three as follows:

IOCB #0 is used by BASIC for I/O to E:
IOCB #6 is used by BASIC for I/O to S:
IOCB #7 is used by BASIC for LPRINT, CLOAD, and SAVE commands.

IOCBs #1 through #5 can be used freely, but the dedicated IOCBs should be avoided unless a program does not make use of one of the dedicated uses above. IOCB #0 can never be opened or closed from a BASIC program.

Each I/O command must have an IOCB associated with it. The I/O commands that can be used in connection with DOS 2.5 are:

OPEN/CLOSE
INPUT/PRINT
PUT/GET
STATUS
XIO

Using the OPEN/CLOSE Commands

OPEN (O.)

Format: OPEN #iocb, aexp1, aexp2, filespec
Example:
```
100 OPEN#2,8,0,"D1:
    ATARI800.BAS"
```

The OPEN statement links a specific IOCB to the appropriate device handler, initializes any CIO-related control variables (see Glossary), and passes any device-specific options to the device handler. The parameters in this statement are defined as follows:

#	Mandatory character entered by user.
iocb	A number between 1 and 7 that refers to a device or file.
aexp1	Number that determines the type of operation to be performed.
	Code 4 = input operation; positions file pointer to start of file.
	6 = disk directory input operation, DOS 2.0S compatible.
	7 = disk directory input, with DOS 2.5 information.
	8 = output operation; positions file pointer to start of file.
	9 = end-of-file append operation; positions file pointer to end of file. Code 9 allows program input from screen editor without user pressing [Return].
	12 = input and output operation; positions file pointer to start of file.
aexp2	Device-dependent auxiliary code.
filespec	Specific file designation (see Section 3 for filespec definition).

In the example OPEN #2, 8, 0, "D1: ATARI800.BAS", IOCB #2 is opened for output to a file in Drive 1 designated as ATARI800. BAS. If there is no file by that name in Drive 1, DOS creates one. If a file by that name already exists, the OPEN statement destroys that file and creates a new one. If the IOCB has already been opened, the screen displays an ERROR-129 (File Already Opened).

CLOSE (CL.)

Format: CLOSE #iocb
Example: `300 CLOSE #2`

The CLOSE command releases the IOCB that had been previously opened for read/write operations. The number following the mandatory # must be the same as the IOCB reference number used in the OPEN statement (see example below). The same IOCB cannot be used for more than one device at a time. You will not get an error message if you close a file that has already been closed.

```
10 OPEN #1,8,0,"D:FIL.BAS"
20 CLOSE #1
```

Note: The END command will close all open files (except IOCB # 0).

Using the INPUT/PRINT Commands

INPUT (I.)

Format: INPUT [#iocb {,/;}] {avar/svar} [, {avar/svar} ...]
Examples:
```
100 INPUT #2;X,Y
100 INPUT #2;N$
```

This command is used to request data (either numerical or string) from a specified device. INPUT is the complement of PRINT. When it is used without a #iocb, the data is assumed to be from the default device (E:). INPUT uses record I/O (see PRINT).

In the sample INPUT/PRINT program listed below, Line 35 allows the user to type in data on the keyboard (default device). In Line 70, the INPUT statement reads the contents of the string from the opened file.

```
5 REM **CREATE DATA FILE**
7 REM **OPEN WITH 8 CREATES DATA FILE**
10 OPEN #1,8,0,"D:WRITE.DAT"
20 DIM WRT$(60)
30 ? "ENTER A SENTENCE NOT MORE THAN
60 CHARACTERS."
35 INPUT WRT$
38 REM **WRITE DATA TO DISKETTE**
40 PRINT #1;WRT$
45 REM **CLOSE DATA FILE**
50 CLOSE #1
55 REM **OPEN DATA FILE FOR READ**
58 REM **OPEN WITH 4 IS A READ ONLY**
60 OPEN #1,4,0,"D:WRITE.DAT"
65 REM **READ DATA FROM DISKETTE**
70 INPUT #1;WRT$
75 REM **PRINT DATA**
80 PRINT WRT$
85 REM **CLOSE DATA FILE**
90 CLOSE #1
```

PRINT (PR. or ?)

Format: PRINT { # iocb / [exp] } [{ ; / , } [exp]...]

Examples:
```
100 PRINT #2;X,Y
100 PRINT #2;A$
100 ? C$
100 PRINT "X = ",X
```

This command writes an expression (whether string or arithmetic) to the opened device with the same IOCB reference number.

If no IOCB number is specified, the system writes the expression to the screen, which is the default device. If the information is directed to a device that is not open, ERROR-133 displays on the screen.

PRINT performs what is called record I/O. Records are sets of bytes separated by end-of-line characters (9B Hex). The size of a record is arbitrary. Record size can be determined by the length of a string printed to a diskette file or the format of an arithmetic variable. It can also be the length of a string of characters entered from the keyboard and terminated by [Return].

The INPUT statement cannot generally read a record that is longer than 255 characters in length. If you PRINT a record to the disk that you might later want to INPUT, it is best to limit the size of the PRINTed records to 255 characters or less.

Direct Accessing With the NOTE/POINT Commands

NOTE (NO.)

Format: NOTE #iocb, avar, avar
Example: `NOTE #2,A,B`

Files are created sequentially and are normally accessed from beginning to end. If you want to access the records in a file in a nonsequential manner (directly), you can either read the file sequentially and stop at the record you want, or use a special method of addressing the record you want.

Because the former is very time-consuming for large files, DOS 2.5 incorporates NOTE and POINT to give you the capability of accessing a file randomly. To get to a record without going through every record that precedes it, you need to let the computer know what record you want. This requires a "note" of the file's sector, so you use a NOTE command before each write and save the returned value in a table.

This command gets the value of the current file pointer for the file using the specified IOCB. The file pointer specifies the exact position in the file where the next byte is to be read or written. This command stores the absolute disk sector number in the first arithmetic variable and the current byte number in the second. Sector numbers range from 1 to 719 in single density and 1 to 1023 in enhanced density; byte numbers range from 0 to 124.

The following program listing and sample run illustrate one way of using NOTE to store keyboard input into a specified file location.

Following is a sample program that uses NOTE to store information entered from the keyboard in a Random-Access disk file. Type in the program, enter some numbers, and press [Return] to end. The sample run uses numbers, but you can type any string for A$ up to 40 characters.

```
1 REM NOTEST - NOTE STATEMENT DEMO
2 REM THIS PROGRAM READS LINE OF DATA
3 REM FROM THE KEYBOARD AND STORES
4 REM THEM ON DISK IN FILE D:DATFIL.DAT.
5 REM POINTERS ARE STORED IN D:POINTS.DAT.
20 DIM A$(40)
25 OPEN #1,8,0,"D:DATFIL.DAT"
27 OPEN #2,8,0,"D:POINTS.DAT"
30 REM READ LINE OF DATA FROM K:
40 INPUT A$
41 PRINT A$
42 REM IF RETURN ONLY, THEN STOP
45 IF LEN(A$)=0 THEN 100
50 NOTE #1,X,Y
55 REM STORE LINE OF DATA.
60 PRINT #1;A$
61 REM STORE POINTER TO BEGINNING OF
62 REM LINE OF DATA.
65 PRINT #2;X;",";Y
70 PRINT "SECTOR # = ";X,"BYTE # = ";Y
90 GOTO 40
95 REM INDICATE END OF FILE
100 PRINT #2;0;",";0
110 END
```

This sample program was run on a diskette that contained the DOS.SYS, DUP.SYS, and MEM.SAV files. Your sector and byte numbers may be different. Sample entries were 45, 55, 75, 80, 90, 100, and 110.

```
45
SECTOR # = 145          BYTE # = 9
55
SECTOR # = 145          BYTE # = 12
75
SECTOR # = 145          BYTE # = 15
80
SECTOR # = 145          BYTE # = 18
90
SECTOR # = 145          BYTE # = 21
100
SECTOR # = 145          BYTE # = 24
110
SECTOR # = 145          BYTE # = 28
```

POINT (P.)

Format: POINT #iocb, avar, avar
Example: `100 POINT #2,A,B`

POINT is the complement of NOTE. This command sets the file pointer to an arbitrary value determined by the arithmetic variables. POINT is used when reading specified file locations (sector and byte) into RAM. The first arithmetic variable specifies (points to) the sector number, and the second arithmetic variable specifies the byte number into which the next byte will be read or written. As with the NOTE command, the sector number ranges from 1 to 719 in single density and 1 to 1023 in enhanced density; byte range is between 0 and 124. If you point out of an opened file, you will get a File Number Mismatch error message. The program listing below contains an example of the POINT command to read data created by the program shown as the example for the NOTE command.

When run, this program prints the keyboard input by sector and byte in reverse order from the way it was written to the diskette.

After you type the sample NOTE and POINT programs, run the NOTE program, then run the POINT program without changing disks.

```
1 REM POINTEST - POINT STATEMENT DEMO
2 REM THIS PROGRAM TAKES THE FILE
3 REM CREATED BY NOTEST AND PRINTS
4 REM THE LINES IN REVERSE ORDER.
10 DIM B(20,1)
20 DIM A$(40)
25 REM OPEN DATA FILE
30 OPEN #1,4,0,"D:DATFIL.DAT"
35 REM OPEN POINTER FILE
40 OPEN #2,4,0,"D:POINTS.DAT"
45 REM READ POINTERS INTO AN ARRAY
50 FOR I=0 TO 20
60 INPUT #2;X,Y
70 B(I,0)=X:B(I,1)=Y
80 IF X=0 AND Y=0 THEN LAST=I:I=20
90 NEXT I
95 REM PRINT FILE IN REVERSE ORDER
100 FOR I=LAST-1 TO 0 STEP -1
110 X=B(I,0):Y=B(I,1)
120 POINT #1,X,Y
130 PRINT "SECTOR # = ";X,"BYTE # =
";Y
140 INPUT #1;A$
150 PRINT A$
160 NEXT I
```

Here is the sample run:

```
SECTOR # = 145          BYTE # = 28
110
SECTOR # = 145          BYTE # = 24
100
SECTOR # = 145          BYTE # = 21
90
SECTOR # = 145          BYTE # = 18
80
SECTOR # = 145          BYTE # = 15
75
SECTOR # = 145          BYTE # = 12
55
SECTOR # = 145          BYTE # = 9
45
```

The PUT/GET Commands

PUT (PU.)

Format: PUT #iocb, aexp
Example: `100 PUT #6,ASC("A")`

The PUT command writes a single byte (value from 0 to 255) to the device specified by the IOCB reference number. In the sample program below, the PUT command is used to write each number you type into an array dimensioned as A(50). You can enter up to 50 numbers, each of which must be less than 256 but greater than or equal to zero. This command is used to create data files or to append data to an existing file.

GET (GE.)

Format: GET #iocb, avar
Example: `100 GET #2,X`

This command reads a single byte from the device specified by the IOCB reference number into the specified variable. GET allows you to retrieve each byte stored by the PUT command.

The sample program shown below demonstrates PUT and GET. In the GET part of the program (lines 130 to 230), rather than using a TRAP statement to sense an end-of-file error, a zero byte (entered by you) has been used to determine the end of data.

```
10 REM**PUT/GET DEMO**
20 DIM A(50),A$(10)
30 ? "PUT AND GET TO DISK PROGRAM
EXAMPLE":?
40 ? "IS THIS TO BE A READ OR A
WRITE?":INPUT A$:?
50 IF A$="READ" THEN 170
60 IF A$<>"WRITE" THEN PRINT "?":
GOTO 40
70 REM WRITE ROUTINE
80 OPEN #1,8,0,"D1:EXAMPL1.DAT"
90 ? "ENTER A NUMBER LESS THAN
256":PRINT "(0 TO END)": INPUT X
```

```
95 REM WRITE NUMBER TO FILE
100 PUT #1,X
110 IF X=0 THEN CLOSE #1:GOTO 130
120 GOTO 90
130 GRAPHICS 0: ?: ? "READ DATA IN
FILE NOW?" :INPUT A$: ?
140 IF A$="NO" THEN END
150 IF A$<>"YES" THEN 130
160 REM READ OUT ROUTINE
170 OPEN #2,4,0,"D:EXAMPL1.DAT"
180 FOR E=1 TO 50
185 REM READ NUMBER(S) FROM FILE
190 GET #2,G:A(E)=G
200 IF G=0 THEN GOTO 230
210 PRINT "BYTE#";E;"=";G
220 NEXT E
230 CLOSE #2
```

Note that INPUT/PRINT and PUT/GET are incompatible types of Input/Output commands. PRINT inserts end-of-line (EOL) characters between records and INPUT uses them to determine a record. GET and PUT merely write single bytes to a file without separating them with EOL. A file created using PUT statements will look like one large record unless you place an EOL (9B hex) character into the file. After you enter the sample PUT/GET program, type **RUN** and press [Return]. Use the number entries 2, 5, 67, 54, and 68. When you run the program, it prints the numbers entered from the keyboard together with the byte location where each is stored. For example:

BYTE #1 = 2
BYTE #2 = 5
BYTE #3 = 67
BYTE #5 = 68

Using the STATUS Command

STATUS (ST.)

Format: STATUS #iocb, avar
Example: `100 STATUS #5,ERROR`

The STATUS command is used to determine the condition (state) of a file. This command is a CIO command and checks for several ways an error might occur. The first set of possible errors it checks for is as follows:

Sector buffer available?	If no, then ERROR-161
Legal device number?	If no, then ERROR-20
Legal filename?	If no, then ERROR-170
File on diskette?	If no, then ERROR-170
File locked?	If yes, then ERROR-167

You can also identify all I/O serial bus errors with a STATUS command. These are as follows:

Device timeout	ERROR-138
Device not acknowledged	ERROR-139
Serial bus error	ERROR-140
Serial bus data frame overrun	ERROR-141
Serial bus checksum error	ERROR-142
Device done	ERROR-144

Before you can issue a STATUS command, you must open the file. It is advisable that you use the XIO command form for this command since it is more reliable and allows you to associate a specific filename with the error you are trying to find.

Substituting the XIO Command for DOS Menu Options

XIO (X.)

Format: XIO cmdno, #iocb, aexp1, aexp2, filespec
Example: `100 XIO 3,#6,4,0,"D:TEST.BAS"`

The XIO command is a general INPUT/OUTPUT statement used for special operations. It is used when you want to perform some of the functions that would otherwise be performed using the DOS Menu selections. These XIO commands are used to open a file, read or write a record or character, close a file, reference a location in a file for reading or writing, or to rename, delete, lock, or unlock a file. Note that XIO calls require filespecs.

CMDNO (command number) is used to designate just which of the operations is to be performed.

CMDNO	OPERATION	EXAMPLE
13	STATUS Request	XIO 13, #1, 0, 0, "D: TEST.BAS"
32	RENAME	XIO 32, #1, 0, 0, "D: OLD, NEW"
33	DELETE	XIO 33, #1, 0, 0, "D: TEMP BAS"
35	LOCK FILE	XIO 35, #1, 0, 0, "D: ATARI. BAS"
36	UNLOCK FILE	XIO 36, #1, 0, 0, "D:DOSEX. BAS"
253	FORMAT DISK	XIO 253, #1, 0, 0, "D1:" Format Single Density
254	FORMAT DISK	XIO 254, #1, 0, 0, "D1:" Format Enhanced Density

Note: Do not use the *device name* twice when renaming a file, i.e., do not use "D: OLD, D: NEW."

Following is a mini-DOS program that lets you manipulate your disk files from BASIC. Normally if you want to rename, delete, lock (protect) or unlock (unprotect) a file, you need to exit BASIC by typing DOS to access the DOS menu, which usually erases any BASIC program in memory. By using the XIO command you can perform these functions without exiting to DOS.

```
10 REM MINI-DOS PROGRAM TO ACCESS DOS
20 REM FUNCTIONS FROM BASIC
30 DIM FILE$(40),NAME2$(20)
40 REM GET DISK DIRECTORY
50 GRAPHICS 0
60 TRAP 100:OPEN #1,6,0,"D:*.*"
70 FOR I=1 TO 64
80 INPUT #1,FILE$
90 ? FILE$: NEXT I
100 CLOSE #1
110 ? "DO YOU WANT TO: (1) RENAME A
FILE"
120 ? :? "(2) DELETE A FILE":? :? "(3)
LOCK A FILE"
130 ? :? "(4) UNLOCK A FILE":? :? "(5)
QUIT"
140 ? :? "ENTER A NUMBER (1-5)";:INPUT
N
150 IF N<1 OR N>5 THEN 140
160 ON N GOSUB 200,300,400,500,170:
GOTO 50
170 END
198 REM SUBROUTINE TO RENAME A FILE
199 REM USE XIO 32
200 ? "ENTER FILENAME TO CHANGE IN
THIS FORM: D:FILENAME.EXT"
210 INPUT FILE$
220 ? "ENTER NEW NAME FOR ";FILE$;" IN
THIS FORM: FILENAME.EXT"
230 INPUT NAME2$
235 FILE$(LEN(FILE$)+1)=","
237 FILE$(LEN(FILE$)+1)=NAME2$
240 XIO 32,#1,0,0,FILE$
250 RETURN
298 REM SUBROUTINE TO DELETE A FILE
299 REM USE XIO 33
300 ? :? "ENTER FILENAME TO DELETE IN
THIS FORM: D:FILENAME.EXT"
310 INPUT FILE$
320 ? :? "ARE YOU SURE YOU WANT TO DE-
LETE ";FILE$;" (Y/N)";
330 INPUT NAME2$
```

```
340 IF NAME2$(1,1)<>"Y" THEN 360
350 XIO 33,#1,0,0,FILE$
360 RETURN
398 REM LOCK FILE WITH XIO 35
400 ? "ENTER FILENAME TO LOCK IN THIS
FORM: D:FILENAME.EXT"
410 INPUT FILE$
420 XIO 35,#1,0,0,FILE$
430 RETURN
498 REM UNLOCK FILE WITH XIO 36
500 ? "ENTER FILENAME TO UNLOCK IN
THIS FORM: D:FILENAME.EXT"
510 INPUT FILE$
520 XIO 36,#1,0,0,FILE$
530 RETURN
```

Accessing Damaged Files

Files can be damaged in two ways. The disk directory entry (filename, directory pointer, and the number of sectors in the file) can be damaged; or the file itself can be damaged. (For more information on accessing and repairing damaged files, see the DISKFIX.COM section of Appendix L.)

Should the disk directory entry be damaged, there is no way to access the file. If the disk directory entry was accidentally deleted, an ERROR 170 (File Not Found) will appear on the screen. If the number of sectors indicated in the disk directory entry are shorter than the actual number of sectors in the file, an ERROR 164 (File Number Mismatch) will appear on the screen. In the latter case, you may be able to retrieve that portion of the file that falls within the sector range by initiating the Get Byte program shown below, where File 1 = the damaged file and File 2 = the recovery file.

```
10 OPEN #1,4,0,"D:FILE.1"
20 OPEN #2,8,0,"D:FILE.2"
25 TRAP 50
30 GET #1,A
40 PUT #2,B
45 GOTO 30
50 CLOSE #1
60 CLOSE #2
```

Note: You can read only the sectors that fall BEFORE the damaged sector(s). All other sectors after the damage cannot be accessed. As a result, it would be best to COPY the good files on the damaged diskette to a new diskette to avoid any further problems.

If the file itself is damaged, you can also use the Get Byte program to transfer each good sector from the damaged file into a recovery file.

The AUTORUN.SYS File

When an AUTORUN.SYS file exists on the diskette in Drive 1, that file will automatically be loaded into RAM and executed (if appropriate) every time you boot the system. This entire process is completed before control of the system is returned to you. The AUTORUN.SYS file can be data; it can also be object code that is loaded but not executed; or it can be object code that is loaded and then executed as soon as the load is complete.

The following sample program shows the use of AUTORUN.SYS to boot up directly to DOS even if built-in BASIC or a cartridge is present. After execution, AUTORUN.SYS normally returns to the DOS initialization routine. If it does not return during your application, or if you use [Reset] before the return, the system initialization must be completed before proceeding. You can do this by modifying two operating system storage locations: COLDST at address 244 (hex) and BOOT at address 9 (hex). COLDST should be cleared to 00 and BOOT set to 01.

The sample program sets these two locations to the proper value and then jumps indirectly to the start DOS vector.

If you do not have an Assembler Editor cartridge, you can create the equivalent file using BASIC POKE statements and then saving the binary file in DOS. Enter and RUN the following:

```
10 FOR J=0 TO 10
20 READ CODE
30 POKE 15000+J, CODE
40 NEXT J
50 DATA 162,0,142,68,2,232,134,9
60 DATA 108,10,0
```

After running the program, go to DOS and save the file using Option K, Binary Save.

Example:

```
SAVE-GIVE FILE,START,END(,INIT,RUN)
AUTORUN.SYS,3A98,3AA2,3A98
```

Note: There is no number entered for the INIT parameter. If you turn off your computer and then turn it back on, you should boot up directly into DOS. To enter BASIC, use the RUN CARTRIDGE function or press [Reset].

Here is an assembly version of the program:

Hex Code	Assembly Language
	; Autorun Program
	;
	; Run DOS without going to cartridge
	;
	COLDST = $244
	BOOT = $09
	DOSVEC = $0A
	* = 3A98
A2 00	DOSGO LDX #0
8E 44 02	STX COLDST
E8	INX
86 09	STX BOOT
6C 0A00	JMP (DOSVEC)
	* = $2E0
	; run address at 2E0
98 3A	.WORD DOSGO
	.END

APPENDIX A ALPHABETICAL DIRECTORY OF BASIC RESERVED WORDS USED WITH DISK OPERATIONS

This is a list of BASIC reserved words and their abbreviations, with a brief summary of the BASIC statement made by each one.

Note: The period is mandatory after all abbreviated keywords.

CLOSE	CL.	I/O statement used to close a disk file at the conclusion of I/O operations.
DOS	DO.	This command causes the DOS Menu to appear. The Menu contains all DOS utility selections. Passes control from cartridge to DOS utilities.
END		Stops program execution, closes files, and turns off sounds. Program may be restarted using CONT. (**Note:** END may be used more than once in a program.)
ENTER	E.	I/O command used to retrieve a listed program in untokenized (textual) form. If a program or lines are entered when a program is resident in RAM, ENTER will merge the two programs. If you don't want programs merged, type NEW before using ENTER to load a program into RAM.

GET	GE.	Used with disk operation to enter a single byte of data into a specified variable from a specified device.
INPUT	I.	This command requests data from a specified device. The default device is E:(Screen Editor).
LIST	L.	This command outputs the untokenized version of a program to a specified device.
LOAD	LO.	I/O command used to retrieve a saved program in tokenized form from a specified device.
NOTE	NO.	This command stores the absolute disk sector number and the current byte number of the file pointer in its two arithmetic variables.
OPEN	O.	Opens the specified file for input or output operations. Determines the type of operations allowed on a file.
POINT	P.	This command is used in setting the file pointer to a specified location (sector and byte) on the diskette.
PRINT	PR. or ?	I/O command that causes output from the computer to a specified output device in record format.
PUT	PU.	Causes output of a single byte of data—i.e., a character—from the computer to a specified device.

RUN	R.	Both loads and starts execution of designated filespec.
SAVE	S.	I/O statement used to record a tokenized version of a program in a specified file on a specified device.
STATUS	ST.	Calls status routine for specified device.
TRAP	T.	Directs execution to a specified line number in case of a program error, allowing you to maintain control of the program and recover from errors.
XIO	X.	General I/O statement used in a program to perform DOS Menu selections and specified I/O commands.

APPENDIX B
NOTATIONS AND TERMINOLOGY USED WITH DOS 2.5

SYSTEM RESET	Press the [Reset] key on the keyboard.
RETURN	Press the [Return] key on the keyboard.
[ ]	Brackets enclose optional items.
. . .	Ellipsis. An ellipsis following an item in brackets indicates that you can repeat the optional item any number of times, but are not required to do so.
{ }	Braces. Items stacked vertically in braces indicate that you have a choice as to which item you want to insert. Select only one to put in your statement or command.
CAPITAL LETTERS	Capital letters are used (usually in **BOLDFACE**) to indicate commands, statements, and other functions you must type exactly as they appear.
, . / : ; "	Punctuation marks. These punctuation marks must be typed as shown in the format of a command or statement. However, do not type brackets or braces.
cmdno	Command number. Used in XIO commands.
exp	Expression. In this manual, expressions are divided into three types: arithmetic, logical, and string expressions.

aexp	Arithmetic expression. Generally composed of a variable, function, constant, or two arithmetic expressions separated by an arithmetic operator(aop).
aexp1	Arithmetic expression 1. This arithmetic expression represents the first auxiliary I/O control byte when used in commands such as OPEN.
aexp2	Arithmetic expression 2. This arithmetic expression represents the second auxiliary I/O control byte when used in commands such as OPEN. Usually it is set to 0.
filespec	File specification. Usually a string expression that refers to a file and the device where it is located, e.g., "D1:MYPROG.BAS" for a file on Drive 1.
IOCB	Input/Output Control Block (IOCB). An arithmetic expression that evaluates to a number from 1 to 7. The IOCB is used to refer to a device or file. IOCB 0 is reserved in BASIC for the Screen Editor and should only be used if the Screen Editor is not to be used.
lineno	Line number. A constant that identifies a particular program line in a deferred-mode BASIC program. A line number can be any integer from 0 through 32767. Line numbering determines the order of program execution.
var	Variable. Any variable. In this manual, variables are classified as arithmetic variables (avar), matrix variables (mvar), or string variables (svar).

avar	Arithmetic variable. A location where a numeric value is stored. Variable names can be from 1 to 120 alphanumeric characters, but must start with an unreversed, uppercase alphabetic character.
svar	String variable. A location where a string of characters may be stored.

APPENDIX C
ERROR MESSAGES AND HOW TO RECOVER

Note: Error messages 2 through 21 should only occur when running a BASIC program.

Error No.	Error Name	Cause and Recovery
2	Insufficient Memory	Your computer system does not have enough memory to store the statement, or to dimension a new string variable. Delete any unused variable names or add more memory. (See your *BASIC Reference Manual* for tips on memory conservation.)
3	Value Error	Either the expected positive integer was negative or the value was not within the expected range.
4	Too Many Variables	You have exceeded the maximum number (128) of variable names and must delete any that are no longer applicable. (See your *BASIC Reference Manual.*)
5	String Length Error	You have attempted to read from or write into a location past the dimensioned string size, or you have used zero as a reference index. Enlarge DIM size. Do not use zero as an index.
6	Out of Data Error	You do not have enough data in your DATA statements for the READ statements.
7	Line Number Greater Than 32767	Check line number references in statements such as GOTO and RESTORE.

Error No.	Error Name	Cause and Recovery
8	Input Statement Error	You have attempted to enter a non-numeric value into a numeric variable. Check your variable types and/or input data.
9	Array or String DIM Error	The DIM size exceeds 5460 for numeric arrays or 32767 for strings; an array or string was redimensioned; reference was made to an undimensioned array or string.
11	Floating Point Overflow/ Underflow	You have attempted to divide by zero or to refer to a number with an absolute value less than 1E-99 or greater than or equal to 1E + 98.
12	Line Not Found	A GOSUB, GOTO, or THEN statement referenced a non-existent line number.
13	No Matching FOR	A NEXT statement was encountered without a matching FOR.
14	Line Too Long Error	You have exceeded the BASIC line-processing buffer length.
15	GOSUB or FOR Line Deleted	A NEXT or RETURN statement was encountered and the corresponding FOR or GOSUB was deleted since the last time the program was run.
16	RETURN Error	Check your program for a missing GOSUB statement.
17	Syntax Error	The computer encountered a line with improper syntax. Fix the line.
18	VAL Function Error	The string in a VAL statement is not a number string.
19	LOAD Program Too Long	Your computer system does not have enough memory to load your program.

Error No.	Error Name	Cause and Recovery
20	Device Number Error	You entered a device number that was not between 1 and 7.
21	LOAD File Error	You attempted to load a nonload file, not a BASIC tokenized file. Tokenized files are created with the SAVE command.

Note: The following are input/output errors that result during the use of disk drives, printers, or other accessory devices. Further information is often provided in the auxiliary hardware manual.

128	BREAK Abort	You have pressed the [Break] key during I/O operation, stopping execution.
129	IOCB* Already Open	An OPEN statement within a program loop or IOCB is already in use for another file or device.
130	Nonexistent Device	You have tried to access a device not in the handler table, i.e., the device is undefined. This error can occur when trying to access an ATARI direct-connect modem without running the "T:" device AUTORUN.SYS file. Another common cause of this error is specifying a filename without a device, i.e., "MYFILE" instead of "D:MYFILE." Check your I/O command for the correct device. Then load and initialize the correct handler.
131	IOCB Write-Only	You have attempted to read from a file opened for write-only. Open the file for read or update (read/write).

*IOCB refers to Input/Output Control Block.

Error No.	Error Name	Cause and Recovery
132	Illegal Handler Command	This is a CIO error code. The command code passed to the device handler is illegal. The command is either less than or equal to 2 or is a special command to a handler that hasn't implemented any special commands. Check your XIO or IOCB command code for an illegal command code.
133	Device/File Not Open	You have not opened this file or device. Check your OPEN statement or file I/O statement for a wrong file specification.
134	Bad IOCB Number	You have tried to use an illegal IOCB index. For BASIC the range is 1–7, as BASIC does not allow use of IOCB 0. The Assembler Editor cartridge requires the IOCB index to be a multiple of 16 and less than 128.
135	IOCB Read-Only	You have tried to write to a device or file that is open for read-only. Open the file for write or update (read/write).
136	End of File	You have read all the data from the file.
137	Truncated Record	This error typically occurs when the record you are reading is larger than the maximum record size specified in the call to CIO. (BASIC's maximum record size is 255 bytes.) Trying to use an INPUT (record-oriented) type of command on a file that was created with PUT (byte-oriented) commands can result in this problem.

Error No.	Error Name	Cause and Recovery
138	Device Timeout	When you sent a command over the serial bus, the device did not respond within the period set by the Operating System for that device command. Either the device number is wrong or you specified the wrong device; the device is not there (wrong spec); it is unable to respond within the proper period; or it is not connected. If the device is a cassette, the tape baud rate may have been mismeasured or the tape improperly positioned. Examine all connections to make sure they are secure and check the disk drive to make sure it is turned on and set for the correct drive number. Check your command for the correct drive number. Retry the command. If this error recurs, have the disk drive checked.
139	Device NAK	The device cannot respond because of bad parameters such as an unaddressable sector. The device might also have received a garbled or illegal command or received improper data from the computer. Check your I/O command for illegal parameters and retry the command. Also check your I/O cables. This is a device-specific error, so refer to the documentation for that device.

Error No.	Error Name	Cause and Recovery
140	Serial Frame Error	Bit 7 of SKSTAT in the POKEY chip is set. This means that communication from the device to the computer is garbled. This is a very rare error and it is fatal. If it occurs more than once, have your device or computer checked. You can also remove the peripherals one at a time to isolate the problem. For cassettes, try the recovery suggested in Error 138.
141	Cursor Out of Range	Your cursor is out of range for the particular graphics mode you chose. Change the cursor position parameters.
142	Serial Bus Overrun	Bit 5 of SKSTAT in POKEY is set. The computer did not respond fast enough to a serial bus input interrupt, or POKEY received a second 8-bit word on the serial bus before the computer could process the previous word. This is rare error. If it occurs more than once, have your computer serviced.
143	Checksum Error	The communications on the serial bus are garbled. The checksum sent by the device is not the same as that calculated for the frame received by the computer. There is no standard recovery procedure because it could be either a hardware or software problem.

Error No.	Error Name	Cause and Recovery
144	Device Done Error	The device is unable to execute a valid command. You have either tried to write to a write-protected diskette or device, or the disk drive is unable to read/write to the requested sector. Remove the write-protect tab. This error is also caused by trying to read an enhanced density diskette in an 810 drive or by variations in drive motor speed. If you suspect motor speed, have your drive checked. See specific manuals for other devices.
145	Illegal Screen Mode	You have tried to open the Screen Editor with an illegal graphics mode number. Check your graphics mode call or the aux2 byte in the IOCB.
146	Function Not Implemented	The handler does not contain the function—e.g., trying to PUT to the keyboard or issuing special commands to the keyboard. Check your I/O command for the right command and the correct device.
147	Insufficient RAM	Not enough RAM for the graphics mode you selected. Add more memory or use a graphics mode that doesn't require as much memory.
160	Drive Number Error	You specified a drive number that was not 1–8, you did not allocate a buffer for the drive, or your drive was not powered up at boot time. Refer to Sections 1 and 2 of this manual. Check your filespec or byte 1802 for the number of drive buffers allocated.

Error No.	Error Name	Cause and Recovery
161	Too Many OPEN Files	You don't have any free sector buffers to use on another file. Check Location 1801 for the number of allocated sector buffers. Also make sure that no files are open that should not be open.
162	Disk Full	You don't have any more free sectors on this diskette. Use a different diskette that has free sectors.
163	Unrecoverable System I/O Error	This error means that the File Manager has a bug in it. Your DOS or the diskette may be bad. Try using another DOS.
164	File Number Mismatch	The structure of the file is damaged, or POINT values are wrong. One of the file links points to a sector allocated to another file. Turn the system off and retry program execution. If this fails, you have lost the file. Try to recover the other files on the diskette, then reformat the diskette.
165	File Name Error	Your file specification has illegal characters in it. Check the filespec and remove the illegal characters.
166	POINT Data Length Error	The byte count in the POINT call was greater than 125 (single-density). Check the parameters in your POINT statement.
167	File Locked	You have tried to access a locked file for purposes other than reading it. Use DOS Menu option G to unlock the file and retry your command.

Error No.	Error Name	Cause and Recovery
168	Device Command Invalid	You issued an illegal command to the device software interface. Check the documentation for that device and retry the command.
169	Directory Full	You have used all the space allocated for the Directory (64 files).
170	File Not Found	You have tried to access a file that doesn't exist in the diskette's Directory. Use DOS Menu option A to check the correct spelling of the filename and to be sure it is on the diskette you are using.
171	POINT Invalid	You have tried to POINT to a byte in a file not opened for UPDATE. Check the parameters of your OPEN statement or aux1 byte of the IOCB used to open the file.
172	Illegal Append	You have tried to open a DOS 1 file for append using DOS 2.5. DOS 2.5 cannot append to DOS 1 files. COPY the DOS 1 file to a DOS 2.5 diskette using DOS 2.5.
173	Bad Sectors at Format Time	The disk drive has found bad sectors while formatting a diskette. Use another diskette, as you cannot format a diskette with bad sectors. If this error occurs with several diskettes, your disk drive may need repair.

APPENDIX D
DOS 2.5 MEMORY MAP FOR 64K RAM SYSTEM

ADDRESS		CONTENTS
Decimal	Hexadecimal	
65535	FFFF	
		OPERATING SYSTEM
49152	C000	
49151	BFFF BFFF	
	or or NONE	16K, 8K, or NO CARTRIDGE
32768	8000 A000	
32767	7FFF or 9FFF or BFFF	
		SCREEN DISPLAY AREA
varies		
varies		HIMEM*
		USER PROGRAM AREA
varies		
	3305	DISK UTILITY PROGRAMS (DUP.SYS)
	1D7C	LOMEM**
		Sector Buffer 4 BUFFER
		Sector Buffer 3 AREA
		Sector Buffer 2 RESERVED
		Sector Buffer 1 FOR DOS 2.5
		Drive 8 (RAMDISK) Buffer
		Drive 2 Buffer
6781	19CC	Drive 1 Buffer
6780	19CB	
		FILE MANAGEMENT
		SUBSYSTEM (DOS.SYS)
1792	0700	
	06FF	
		VARIES — most often used by languages
		(ATARI BASIC does *not* use $680 – $6FF).
	480	
	47F	
		OPERATING SYSTEM
		(including cassette buffer)
	0***	

* Depends on which graphics mode is currently in use.
** Varies with the number of drive and sector buffers reserved.
*** Portions of PAGE 0 ($00 – $FF) may be available with some languages. The Operating System alone uses only $00 – $7F.

Note 1: For given Drive Buffer allocation and Sector Buffer allocation, LOMEM can be determined by PEEKing locations 2E7 (LOW) and 2E8 (HIGH) Hex or 743 (LOW) and 744 (HIGH) Decimal.

Note 2: To determine the amount of User Program Area available or HIMEM, you can either make use of the BASIC FRE(0) instruction or PEEKing Locations 2E5 (LOW) and 2E6 (HIGH) Hex or 741 (LOW) and 742 (HIGH) Decimal.

APPENDIX E
HEXADECIMAL TO DECIMAL CONVERSION TABLE

FOUR HEX DIGITS

4		3		2		1	
HEX	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	206	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

Use this table to convert up to four hex digits.

For example, to convert the hex number 1234 to decimal, add the entries from each of the four columns in the table. For 1 use the column number 4, and so on.

```
1234 hex. =      4096
               +  512
               +   48
               +    4
               ------
                 4660  dec.
```

Other examples:

```
EEDD hex. =     57344
              +  3584
              +   208
              +    13
              ------
                61149  dec.

AB hex. =         160
                +  11
                ------
                  171  dec.
```

APPENDIX F
HOW TO SPEED UP DATA TRANSFERS TO DISK DRIVE

DOS 2.5 has the ability to Write with Read Verify, a safety technique that should be used whenever improved reliability is more important than rapid data transmissions. This is the way your DOS 2.5 Master Diskette is shipped to you. To save time, however, the information can be written to the diskette without a Read Verify. Memory Location 1913 (decimal) contains the data that determines whether the File Management Subsystem will use Write with Read Verify (57 hex, 87 decimal) or Write without Read Verify (50 hex, 80 decimal). Write without Read Verify is of course faster, but may not be as reliable. To customize your version of DOS 2.5 from BASIC, you need to

```
POKE 1913,80
```

for fast Write (Write without Read Verify). If you would rather have the Write with Read Verify,

```
POKE 1913,87
```

To alter the version of DOS stored on diskette so that your custom version will always boot in, simply type **DOS** and then use an H command (WRITE DOS FILES) from the DOS Menu to store the new version of DOS from RAM onto your diskette.

(See also Appendix L, the SETUP.COM section, for how you can do this with the "Change System Configuration" option.)

Caution: POKEing location 1913 with any values other than 80 or 87 will cause loss of data, a destroyed diskette, and probably a system lock-up as well.

APPENDIX G
HOW TO TELL DOS HOW MANY DISK DRIVES YOU HAVE

This appendix explains how to change the RAM location number to reflect the number of drives attached to your ATARI Computer system. (See also Appendix L, the SETUP.COM section.) If you are using more than two drives (the default maximum), you will need to poke the correct drive number code into RAM location 1802 (decimal). The following table gives the correct entries for the number of ATARI 810 or 1050 Disk Drives. Note that a maximum of four disk drives can be used with DOS 2.5 since the switches on each drive can only be set from 1 to 4. Note also that in the Binary Drive Code, there is a 1 corresponding to each drive in the system.

CODES FOR NUMBER OF DISK DRIVES ATTATCHED

Drives Allocated	Decimal Drive Code	Binary Drive Code
Drive 1	01	00000001
Drive 2	02	00000010
Drive 3	04	00000100
Drive 4	08	00001000
Drives 1 + 2	03	00000011 (default)
Drives 1 + 2 + 3	07	00000111
Drives 1 + 2 + 3 + 4	15	00001111

So to tell DOS that you have three disk drives, for example, you would use POKE 1802, 7.

To alter the version of DOS stored on diskette so that your custom version will always boot in, simply type **DOS** and press [Return]; then use the H. option (WRITE DOS FILES) from the DOS Menu to store your new version of DOS on your diskette.

Note to ATARI 130XE Computer owners: Keep in mind that if DOS 2.5 is using the additional 64K of memory in a 130XE as a RamDisk, the RamDisk is always Drive 8 (D8:). See Appendix K and your 130XE *Owner's Manual* for more details.

Technical Notes

In theory, either DOS 2.0S or DOS 2.5 can support up to eight disk drives and up to eight files open at the same time. In practice, there are several limitations on these numbers: (1) at a minimum, most programs must reserve IOCB number zero for the keyboard and screen (the **E:** device); (2) ATARI does not make a disk drive that can be assigned a device number greater than 4; (3) because DUP.SYS loads into memory at a particular location under DOS 2.0S, and because DOS 2.5 keeps DUP.SYS at that same location for compatibility reasons, there is a real limit to the amount of buffer room available when DUP.SYS is active.

The practical result of these limitations is that you are limited to five disk drives (including the RamDisk if you have an ATARI 130XE Computer) and seven files open at one time when using BASIC and other languages and programs that pay attention to the system LOMEM pointer (at location $02E7). If, however, you wish to be able to use the DOS Menu, you are limited to a maximum of four drives (again including the RamDisk). Because the DOS Menu never uses more than two files at a time, you must provide for at least that many concurrently open files, but you may use more if you wish.

Another practical limitation on how many drives and buffers you may use is that many programs currently available *assume* both a particular value for LOMEM (the contents of location $0E27) and a minimal configuration of two drives and three concurrently open files. Since LOMEM's value depends on the number of buffers you have chosen, the following comparison chart lists buffer allocation information for both DOS 2.0S and DOS 2.5.

	DOS 2.0S	DOS 2.5
Start of Buffers	$1A7C	$19CC
Bytes per Drive Buffer	$80 (128)	$90 (144)
Bytes per File Buffer	$80 (128)	$90 (144)
LOMEM Value, 2 Drives, 3 Files	$1CFC	$1C6C
LOMEM Value, 3 Drives, 3 Files	$1D7C	$1CFC
LOMEM Value 4 Drives, 2 Files	$1D7C	$1D0C
Bottom of DUP.SYS	$1D7C	$1D7C

As you can see from this chart, the controlling location is $1CFC, which DOS 2.0S uses for LOMEM when DOS is configured for two drives and three files. DOS 2.5 ensures that identical value for three drives and three files because it is expected that a normal ATARI system configuration might consist of two disk drives connected to an Atari 130XE Computer, which allows the use of a RamDisk. Since even a RamDisk requires a drive buffer, it has to be accounted for in configuring for the "magic" value of LOMEM.

APPENDIX H
USING DOS 2.5 WITH AN ATARI 810 DISK DRIVE OR WITH DOS 2.0S FILES

If your computer system includes an ATARI 810 Disk Drive or if you have diskette files created and stored under the earlier ATARI Disk Operating System DOS 2.0S, there are several points to keep in mind as you use DOS 2.5 with your computer system.

If You Have an ATARI 810 Disk Drive

- An 810 Disk Drive cannot read a diskette formatted in enhanced density using DOS 2.5 and an ATARI 1050 Disk Drive. Be sure to mark each diskette you format so that you will know whether it is formatted in single or enhanced density.

- Because your DOS 2.5 Master Diskette is itself formatted in enhanced density, you cannot use it (or any enhanced-density duplicates of it) to load DOS 2.5 from an 810. If you want a version of DOS 2.5 that you can load from an 810, first use DOS to format a diskette in single density. Then use DOS Menu option H., WRITE DOS FILES, to write DOS.SYS and DOS.DUP from your DOS 2.5 Master Diskette onto the diskette. If you like, you can also use option C., COPY FILE, to copy RAMDISK.COM, SETUP.COM, COPY32.COM, and DISKFIX.COM onto your single-density version of DOS (see Appendices K and L).

- You cannot duplicate an enhanced-density diskette from a 1050 Disk Drive to an 810. You *can* duplicate a *single-density* diskette from any ATARI Disk Drive to any other. So if your system includes both a 1050 and an 810, you may find it convenient to format most of your diskettes in single density so that they will be interchangeable between your drives.

- You can usually use DOS Menu option C., COPY FILE, to copy files from an enhanced-density diskette in a 1050 to a single-density diskette in an 810, with these limitations:

  — You cannot of course copy more information than will fit on the diskette formatted in single density, which has a smaller capacity than a diskette formatted in enhanced density.

  — You cannot copy any single file that occupies more than 707 sectors.

  — You may be unable to copy files created with certain data manipulation programs such as mailing lists, file managers, and so forth; some such files can be copied only through diskette duplication. In some cases you may be able to copy a file but the copy may not work with the program's file manager. Check the manual that came with your programs to see if this condition applies to any of your files.

If You Have DOS 2.0S Diskette Files

Diskettes formatted and written to using DOS 2.0S are completely compatible with DOS 2.5. Diskettes formatted in *single density* and written to using DOS 2.5 are completely compatible with DOS 2.0S. In short, there is no difference in format, directory structure, data storage capacity, file structure, and so forth between a DOS 2.0S diskette and a single-density DOS 2.5 diskette.

So problems of incompatibility between DOS 2.5 and DOS 2.0S can arise in only one kind of situation: when DOS 2.0S tries to read a diskette formatted in enhanced density and written to using DOS 2.5. The potential problems discussed below are based on the assumption that you have loaded DOS 2.0S from a 1050 Disk Drive and that you are using the 1050 to manage files on a diskette formatted in enhanced density by DOS 2.5. In this situation DOS 2.0S can read from and write to the enhanced-density diskette with just a few restrictions.

- DOS 2.0S knows nothing about sectors numbered from 720 up. So it will not attempt to write any data to the extended storage capacity of the enhanced-density diskette.

By the same token, any extended files stored under DOS 2.5 (i.e., files occupying sectors numbered 720 and up) will be "invisible" to DOS 2.0S. You will not be able to work with any such files under DOS 2.0S, nor will they show up in a disk directory called up under DOS 2.0S. Consider the following directory listing, as viewed under DOS 2.5:

```
*DOS        SYS 037
*DUP        SYS 042
 FILE1      DAT 204
 FILE2      DAT 119
<FILE3      DAT>350
<FILE4      DAT>022
236 FREE SECTORS
```

If you called up a directory of the same diskette under DOS 2.0S, you would see a somewhat different listing:

```
*DOS        SYS 037
*DUP        SYS 042
 FILE1      DAT 204
 FILE2      DAT 119
000 FREE SECTORS
```

The extended files, marked by DOS 2.5 with brackets, are not recognized by DOS 2.0S. Note also that DOS 2.0S shows *no* free sectors left on the diskette, because in fact the only free sectors are numbered 720 and up. The dropping of FILE-3.DAT, which uses sectors numbered both below and above 720, from the DOS 2.0S directory illustrates yet another point: if *any* sector of a file is numbered 720 or up, the *entire file* is invisible to DOS 2.0S.

On the other hand, those files visible to DOS 2.0S are completely accessible to it—they may be read, locked, unlocked, and so forth using DOS 2.0S. And after you work with such files using DOS 2.0S, the diskette will still be wholly compatible with DOS 2.5, which will again recognize the extended files invisible to DOS 2.0S.

Caution: Never use any DOS 2.0S "disk fix" program with a diskette formatted in enhanced density using DOS 2.5. The program may decide that any "invisible" files are bad, and attempt to erase them. The program may also destroy the

information on the diskette that allows DOS 2.5 to recognize it as an enhanced-density diskette.

- Because DOS 2.5 extended files are invisible to DOS 2.0S and will not show up on a directory listing called up under DOS 2.0S, it is possible accidentally to use DOS 2.0S to write a file to an enhanced-density diskette that already contains an invisible file with the same name. For example, working with the diskette discussed in the preceding paragraphs, you might delete FILE1.DAT and write a new file called FILE3.DAT in its place. If you then called up a directory of the diskette under DOS 2.5, you would discover *two* files named FILE3.DAT. However, DOS 2.5 allows you to rename just one of a pair of files bearing the same name—see option E., RENAME FILE, in Section 3.

Technical Notes

Although a single-density diskette has 720 formatted sectors, both DOS 2.0S and DOS 2.5 utilize only 719 of them. (This is because DOS marks sector zero — which does not appear on 810 and 1050 drives — as unavailable, as if it were part of the boot sectors.)

Several programs are available which count on DOS's *not* using sector 720. Generally, these programs write special information (often including copy protection methods) directly to sector 720.

On a dual-density diskette, DOS 2.5 manages sectors numbered from 1 to 1023. Even though DOS 2.5 could easily use sector 720 for a file, it purposely avoids the sector, marking it as already in use even on a newly formatted diskette. This is done simply to ensure compatibility with any program which uses sector 720 for its own purposes.

In the same vein, a dual-density diskette actually consists of 40 tracks of 26 sectors each, for a total of 1040 sectors. However, due to the inner workings of DOS, it is limited to using sectors 1 through 1023 for files. (This is because DOS uses a 10-bit sector address, and the highest number that 10 bits can represent is 1023.) In addition, DOS 2.5 uses sector 1024 for the *Extended* Volume Table of Contents (also known as the "volume direc-

tory''), where it keeps track of the sectors that are invisible to DOS 2.0S. In no way, though, does DOS 2.5 utilize sector numbers 1025 through 1040, so they are available to the user in much the same way that sector 720 is available to the DOS 2.0S user.

APPENDIX I
STRUCTURE OF A COMPOUND BINARY FILE

Compound File Structure Using C. COPY FILE With Append

Byte No.	Decimal No.	Hex No.	Description	Hex Address
1	255	FF	Identification Code (PART 1)	
2	255	FF		
3	0	00	Starting Address (PART 1)	$5000
4	80	50		
5	31	1F	Ending Address (PART 1)	$501F
6	80	50		
.	.	.	DATA (PART 1)	
.	.	.	32 Bytes	
.	.	.		
38	255	FF	Identification Code (PART 2)	
39	255	FF		
40	32	20	Starting Address (PART 2)	$5020
41	80	50		
42	143	8F	Ending Address (PART 2)	$508F
43	80	50		
.	.	.	DATA (PART 2)	
.	.	.	112 BYTES	
.	.	.		

Compound File Structure Using K. BINARY SAVE with Append

Byte No.	Decimal No.	Hex No.	Description	Hex Address
1	255	FF	Identifier Code	
2	255	FF		
3	00	00	Starting Address (Part 1)	$5000
4	80	50		
5	31	1F	Ending Address (PART 1)	$501F
6	80	50		
.	.	.	DATA (PART 1)	
.	.	.	32 Bytes	
.	.	.		
38	32	20	Starting Address (PART 2)	$5020
39	80	50		
40	143	8F	Ending Address (PART 2)	$508F
41	80	50		
.	.	.	DATA (PART 2)	
.	.	.	112 BYTES	
.	.	.		

APPENDIX J
GLOSSARY OF TERMS

Address: A location in memory, usually specified by a two-byte number in hexadecimal or decimal format. (Maximum range is 0—FFFF hexadecimal.)

Alphanumeric: The capital letters A–Z and the numbers 0–9, and/or combinations of letters and numbers. Specifically excludes graphics symbols, punctuation marks, and other special characters.

Array: A one- or two-dimensional set of elements that can be referenced one at a time or as a complete list by using the array variable name and one or two subscripts. Thus the array B, element number 10 would be referred to as B(10). Note that string arrays are not supported by BASIC, but you can pick up each element within a string—for example, A$(10,10). All arrays must be dimensioned before use. A matrix is a two-dimensional array.

ATASCII: The method of coding used to store text data. In ATASCII (which is a modified version of ASCII, the American Standard Code for Information Interchange), each character and graphics symbol, as well as most of the control keys, has a number assigned to represent it. The number is a one- or two-byte code (decimal 0—255). See the *ATARI BASIC Reference Manual* for table.

AUTORUN.SYS: Filename reserved by the Disk Operating System.

Baud Rate: Signaling speed or speed of information interchange in bits per second.

Binary: Also known as base 2. A number base system using only the digits 0 and 1.

Binary Load: Loading a binary machine-language object file into the computer memory.

Binary Save: Saving a binary machine-language object file onto a disk drive or program recorder.

Bit: Abbreviation of "binary digit." The smallest unit of information, represented by the value 0 or 1.

Boot: This is the initialization program that "sets up" the computer when it is powered up. At conclusion of the boot (or after "booting up"), the computer is capable of loading and executing higher-level programs.

Break: To interrupt execution of a program. Pressing the [Break] key causes a break in execution.

Buffer: A temporary storage area in RAM used to hold data for further processing, input/output operations, and the like.

Byte: Eight bits. A byte can represent one character. A byte has a range of 0 through 255 (decimal).

CIO: Central Input/Output Subsystem. The part of the operating system (OS) that handles input/output.

CLOSE: To terminate access to a disk file. Before you can handle further access to the file, it must be opened again. See OPEN.

Data: Information of any kind, usually a set of bytes.

Debug: To isolate and eliminate errors from a program.

Decimal: Also known as base 10. A number base system using the digits 0 through 9. Decimal numbers are stored in binary-coded decimal format in the computer. See Bit, Hexadecimal, and Octal.

Default: A condition or value that exists or is caused to exist by the computer until it is told to do something else. For example, in BASIC the computer defaults to GRAPHICS 0 until another graphics mode is entered.

Delimiter: A character that marks the start or finish of a data item but is not part of the data. For example, BASIC uses quotation marks (") to delimit strings.

Density: The closeness of space distribution on a storage medium, i.e., the number of sectors per track. Both single and enhanced density record 128 bytes per sector.

Destination: The device or address that receives data during an exchange of information (especially an I/O exchange). See Source.

Directory: A summary of files contained on a diskette listed by filename and file size.

Diskette: Often called simply a disk. A record/playback medium like tape, but made in the shape of a flat disk and placed in an envelope for protection. The access time for a diskette is much faster than for tape.

DOS: Abbreviation for Disk Operating System. The software or programs that facilitate use of a disk drive system.

DOS.SYS: Filename reserved by the Disk Operating System.

Drive Number: An integer from 1 to 8 that specifies the drive to be used.

Drive Specification or Drivespec: Part of the filespec that tells the computer which disk drive to access. If this is omitted, the computer will assume Drive 1.

End of File: A marker that tells the computer that the end of a certain file on disk has been reached.

Entry Point: The address where execution of a machine-language program or routine is to begin. Also called the transfer address.

File: An organized collection of related data. A file is the largest grouping of information that can be addressed with a single name. For example, a BASIC program is stored on diskette as a particular file, and may be addressed by the statements SAVE or LOAD (among others).

Filename: The alphanumeric characters used to identify a file. A total of eight numbers and/or letters may be used, optionally followed by a period and an extender or extension of up to three characters.

Filename Extender or Extension: Up to three additional characters used following a period (required if the extender is used) after the filename. For example, in the filename PHONLIST.BAS, the letters "BAS" comprise the extender.

File Pointer: A pointer to a location in a file. Each file has its own pointer.

Filespec: Abbreviation for file specification. A sequence of characters which specifies a particular device and filename.

Format: To organize a new or magnetically (bulk) erased diskette onto tracks and sectors. When formatted in single density, each diskette contains 40 circular tracks, with 18 sectors per track, and in enhanced density, 40 tracks with 26 sectors per track. Each sector can store up to 128 bytes of data.

Hexadecimal or Hex: Also known as base 16. Number base system using 16 alphanumeric characters: 0,1,2,3,4,5,6,7,8,9,A, B,C,D,E, and F.

I/O: See Input.

Indexed Addressing: See Random Accessing.

INPUT: A BASIC command used to request either numeric or string data from a specified device.

Input: To transfer data from outside the computer (say, from a diskette file) into RAM. Output is the opposite, and the two words are often used together to describe data transfer operations: Input/Output or just I/O. Note that the reference point is always the computer. Output always means from the computer, while Input means into the computer.

IOCB: Input/Output Control Block. A section of RAM reserved for addressing an input or output device and processing data received from it or for addressing and transferring data to an output device.

iocb: An arithmetic expression that evaluates to a number between 1 and 7. This number is used to refer to a device or file.

Kilobyte or K: 1024 bytes. 16K RAM is actually 16 times 1024 or 16,384 bytes.

Least Significant Byte: The byte in the rightmost or low order position in a number or a word.

Machine Language: The instruction set for the particular microprocessor chip used in a computer.

Most Significant Byte: The byte in the leftmost or high order position in a number or a word.

Null String: A string consisting of no characters. For example, A$ = "" stores the null string as A$.

Object Code: Machine language derived from "source code," typically from assembly language.

Octal: Also known as base 8. The octal numbering system uses the digits 0 through 7. Address and byte values are sometimes given in octal form.

OPEN: To prepare a file for access by specifying whether an input or output operation will be conducted, along with the file-spec.

Output: See Input.

Parameter: Variables in a command or function.

Peripheral: An I/O device.

POKEY: A custom I/O chip that manages communication on the serial bus and generates sound on the TV speaker.

Random Accessing: The method of reading data from a diskette directly from the byte and sector where it was stored without having to read the entire file sequentially.

Record: A block of data, delimited by EOL (End-of-Line, 9B Hex) characters.

Sector: A sector is the smallest block of data that can be written to a disk file or read from a file. Each single-density sector can store 128 bytes of data.

Sequential Accessing: The method of reading each byte from a diskette file in order, starting from the first byte in the file.

Source: The device or address that contains the data to be sent to a destination. See Destination.

Source Code: A series of instructions, written in a language other than machine language, which requires translation in order to be executed.

String: A sequence of letters and/or characters usually delimited with quotation marks ("").

System Diskette: An exact copy of an original Master Diskette. Always use backup copies of your Master Diskette instead of the original. Keep backup copies of all important data and program diskettes.

Tokenizing: The process of interpreting textual BASIC source code and converting it to the internal format used by the BASIC interpreter.

Track: A circle on a diskette used for magnetic storage of data. Each track has 18 sectors in single density and 26 in enhanced density, each with 128-byte storage capability. There are a total of 40 tracks on each diskette.

Variable: A variable may be thought of as a box in which a value may be stored. Such values are typically numbers and strings.

Write-Protect: A method of preventing a disk drive from writing on a diskette. Many diskettes are write-protected by covering a notch on the diskette cover with a small sticker.

APPENDIX K
DOS 2.5 AND THE ATARI 130XE RAMDISK

The ATARI 130XE Computer is equipped with 131,072 bytes — 128K — of Random Access Memory (RAM), twice the maximum 64K available with earlier model ATARI Computers. The additional 64K RAM can be useful for many purposes: fast exchange of screen images for animation, additional storage for large data bases, and so forth.

You can also use the 130XE's extra RAM as a very fast "virtual" disk drive. Set up as a "RamDisk" — recognized by DOS 2.5 as Drive 8 in your system — it can accommodate up to the equivalent of 499 sectors on a diskette. That is about half what you can store on a diskette formatted in enhanced density.

Of course, the "storage" capacity offered by the RamDisk is *volatile* memory, which means that information stored in it will be lost when you turn off your computer system. So before turning off your system, you should always be sure that any data currently in the RamDisk that you want to save permanently is recorded on an actual diskette.

However, the RamDisk can be a very convenient tool. It allows you to switch almost instantaneously between BASIC (or any other programming language) and DOS, and back again. You can also use it to work with files "stored" on Drive 8 — a technique that might prove especially useful when you are transferring large amounts of data between two programs that are chained together (that is, when one program RUNs the other).

To Activate the RamDisk

Your DOS 2.5 Master Diskette contains a file called RAMDISK.COM that automatically sets up the 130XE's extra 64K

RAM as a RamDisk. (If you do not want to activate the RamDisk in your 130XE with DOS 2.5, see "If You Do Not Want to Use the RamDisk," below.)

When you boot your 130XE system with a DOS 2.5 Master or System Diskette containing RAMDISK.COM, DOS will —

- Display a message that it is initializing the RamDisk;
- Set up your computer's extra 64K of memory to act very much as a disk drive, telling DOS to regard it as Drive 8; and
- Copy the DOS file DUP.SYS and establish MEM.SAV (see Section 3) on the RamDisk, and proceed when necessary to use the DUP.SYS and MEM.SAV files on the RamDisk rather than the files of the same name on the Master or System Diskette.

If you wish to expand the usable capacity of your RamDisk, you may recover the memory used by DUP.SYS and MEM.SAV by —

- Changing the contents of location 5439 ($153F) to ATASCII 1 — for example, POKE 5439,ASC("1"); and
- Deleting the files DUP.SYS and MEM.SAV from the "diskette" in Drive 8 — that is, the RamDisk. Use option D., DELETE FILE(S), on the DOS Menu and enter **D8:*.*** in response to the DELETE FILESPEC prompt.

  Note: Booting a disk which doesn't contain DUP.SYS will cause RAMDISK.COM to initialize the RamDisk, but DUP.SYS and MEM.SAV will not be moved to the RamDisk.

Using DOS With the RamDisk

Because of the size of the RamDisk, you may not use DOS Menu option J., DUPLICATE DISK, to copy either a single-density or enhanced-density diskette to the RamDisk. Instead, you must copy individual files, taking care that they do not exceed in size the capacity of the RamDisk. You *can* ask DOS to duplicate the contents of the RamDisk on an actual diskette. From then on,

however, that diskette will be capable under DOS of accessing only 499 sectors' worth of data — though you can always duplicate its contents back to the RamDisk.

If You Do Not Want to Use the RamDisk

If you do not want to activate your ATARI 130XE's RamDisk, you can either delete or rename the RAMDISK.COM file on your DOS 2.5 Master or System Diskette. You may then use the 130XE's extra RAM for other purposes.

If you have applications for which you do not wish to use the RamDisk, it is recommended that you leave the RAMDISK.COM file intact on your DOS 2.5 Master Diskette. You might wish to make one working copy of DOS (System Diskette) that contains RAMDISK.COM, and one that does not. Or you can simply rename the RAMDISK.COM file on your System Diskette, then rename it back to RAMDISK.COM when you wish to use it.

APPENDIX L
THE DOS 2.5 DISK UTILITIES

Your DOS 2.5 Master Diskette contains three new utility programs in addition to the standard disk utilities handled by the DUP.SYS file. These three programs, each of which appears on the disk directory with a .COM extender, function as follows:

COPY32.COM allows you to copy files from diskettes formatted and written to under ATARI DOS 3 to DOS 2.5 diskettes, converting the files in the process from DOS 3 to DOS 2.5.

DISKFIX.COM allows you to correct some problems that may occur with files on DOS 2.5 and DOS 2.0S diskettes. Under certain conditions, you can also use this utility to recover deleted files.

SETUP.COM allows you to change certain DOS parameters. You can also use it to create an AUTORUN.SYS file that will automatically load and run a BASIC program when you boot your system.

Note: RAMDISK.COM is not a disk utility per se. It is used only to set up the RamDisk on a 130XE. See Appendix K, for details.

Selecting and Loading a Utility

All three utilities are binary files that are loaded and run using option L, BINARY LOAD, from the DOS 2.5 Menu. For example, to begin using the COPY32.COM program, with the DOS 2.5 Menu on your screen, you would type L and press [Return], then type **COPY32.COM** as the name of the file to load and press [Return] again.

Specific instructions for using each utility follow.

COPY32.COM

Using this utility is much like using the COPY FILE function on the DOS Menu. After you load the COPY32.COM program, you are prompted to specify which drive will hold your DOS 3 (source) disk and which drive will hold your DOS 2.5 (destination) disk. If you have only one drive, type **1** in response to both prompts. In this case, you will have to swap your DOS 3 and DOS 2.5 diskettes during the copying process. If you have more than one disk drive, you may select one to hold your DOS 3 diskette and another to hold your DOS 2.5 diskette. Here is a sample menu:

```
        COPY 3 to 2.x

Copy files from a DOS 3 disk to
   a DOS 2.5 (or DOS 2.0S) disk.
(Hit [Return] for drive # to quit.)
On which drive (1-4) is DOS 3 disk? 1
On which drive (1-4) is DOS 2.x disk? 1

Place the DOS 3 disk in drive 1

CAUTION: You will be swapping disks.

         Put a write protect tab
         on your DOS 3 disk!

Push [Start] when ready.
```

At this point, if you have only one drive, the utility prompts you to insert your DOS 3 disk in Drive 1; for safety, place a write-protect tab on your DOS 3 disk so that you will not erase valuable data if you make an error while swapping diskettes.

If you specified two different drives, the utility prompts you to insert both your DOS 3 and DOS 2.5 disks.

After you insert the diskette or diskettes, press [Start]. The COPY-32.COM program reads the directory of the DOS 3 diskette and displays the files it contains, sixteen at a time, by number. Press [Return] to see the next sixteen files; when all the files on the diskette have been listed, you have the options to Restart, return to DOS, or view the files again.

To convert a file, enter the number of the file you wish to convert. The utility prompts you to confirm your choice by pressing [Start].

When you press [Start], the program begins the conversion process by reading the specified file from the DOS 3 diskette. After COPY32.COM reads the entire file (or as much data as it can accomodate in its memory buffer), it asks you to swap disks if you specified the same drive for your DOS 3 and DOS 2.5 disks; with very large files, you may have to swap diskettes several times. If you are using two drives, the program copies and converts the file in a single operation.

After the file has been copied and converted, press [Start] to return to the listing of files on your DOS 3 diskette, from which you may choose another file to convert.

If an error occurs during the copy process, COPY32.COM displays an error number (see Appendix C) and prompts you to press [Start] to restart, or [Select] to return to the DOS 2.5 menu.

Note: Unless you have two disk drives, you will be unable to convert files of more than 124,700 bytes (300 bytes less than the maximum file length possible under DOS 2.5).

DISKFIX.COM

This program begins by showing you the current drive number, which is always 1 when the program is first loaded, and a menu with these five options:

1. Change Drive #

2. Unerase File

3. Verify Disk

4. Rename File by #

5. Quit to DOS

Type the number of the function you wish to use but *do not* press [Return] after typing your choice.

Instructions for using each of the DISKFIX options follow.

Change Drive

This option lets you choose the drive on which the DISKFIX options 2, 3, and 4 will act. When you select this option, the utility prompts you to specify which disk drive you wish to make current. Type a number from 1 through 8 (if you press any other key, it will not be accepted).

Unerase File

Under certain conditions, Unerase File enables you to recover files that have been deleted. When you delete a file, it is not actually removed from the disk — DOS just marks its directory entry as erased, bars the file from normal access, and marks the space it occupies as available for new files.

Unerase File may also enable you to recover files that were opened for output (for example, with the command OPEN #1, 8, 0, "D:MYFILE.DAT") but not properly closed for some reason (for example, if you pressed [Reset] before closing the file). Such a file may occupy a part of the disk to which you cannot normally regain access. Unerase File can be used to access the file.

Note for advanced users: You can recover a file open for output by unerasing it only if there are no bad links in it.

Your chances of using Unerase File for a successful recovery of a deleted or open file are best if the diskette containing the file has not been written to at all since the file was deleted or opened for output. Otherwise, part or all of the file may have been overwritten by new data.

After selecting Unerase File by typing **2**, insert the disk containing the file you wish to recover in the current drive and press [Start]. The utility displays a list of the first 32 filenames on the disk, each marked with a number from 0 to 31. Each entry has a "file type" code before it. A blank space means the file is in use and is good. A "W" means that the file is in use but was left open for output. A "D" means that the file has been deleted. Blank directory entries appear as "(unused)." Press [Return] for a list of the remaining files on the diskette — up to 32 more will be displayed, numbered 32 to 63.

Enter the number of any filename preceded by a W or D file type code. If you press [Return] or enter a number higher than 63, the message **You didn't choose anything!** appears. If you select an unused entry, the message **That file is unused.** appears.

If the file you specify has not been erased, a message to that effect appears and you must press [Start] to resume. Otherwise, you are shown a numeric entry type code for the file, the filename, the file length, and the starting sector in both hexadecimal and decimal notation. Type code 80 indicates a deleted file, and 43 means open for output. The utility also prompts you to confirm your choice by pressing **Y**. Type **Y** to recover the specified file.

Before actually recovering ("unerasing") a file, the program first verifies all files on the diskette (see the next subsection for details on this process). Unless problems are encountered, the recovered file will also be verified and the directory sector containing its entry will be written back to the diskette with a valid file type code.

After the last file has been verified, the Volume Table Of Contents (VTOC) is written to the diskette and the utility prompts you to press [Start] to return to the DISKFIX.COM menu.

Verify Disk

This function verifies the soundness of every file on a disk. When you select it from the DISKFIX.COM menu, the utility prompts you to insert the disk you wish to verify in the current drive and press Start. Verify Disk tests the validity of each file, and keeps track of each sector. Finally, a new Volume Table of Contents (see below) is written back to the disk.

How the Verification Process Works

DOS 2.5 uses a small portion of every disk, called the Volume Table of Contents (VTOC), to keep track of available sectors on the disk for file allocation purposes. The VTOC is written to each time a file is created, erased, or changed in length. The verification process compares the information contained in the disk directory with actual disk file allocation, makes changes (or displays error messages) as necessary, and then writes a new VTOC to the disk. This process not only checks the validity of files, it also frees up sectors that may previously have been unavailable. Also, since a VTOC is written with each disk verification, you can use the Verify Disk function to repair a disk with no existing VTOC.

During disk verification, any files left open for output (preceded by the W type code in the Unerase File directory) are deleted and their type code changed to D, and a message to this effect appears onscreen. If the file length is zero, as in the case of a file that has the same start sector as a file written later, you will not be able to unerase it — instead, you will see the message **Bad file – deleting.** Otherwise, you should be able to recover at least part of the file. Each sector of every file on the disk is checked to ensure that it belongs to a proper file. If a bad file link in a file with a good directory entry is encountered, the message **Bad link in file – truncating** appears, and only part of the file will be recovered. If an I/O error occurs during reading or writing of the disk, the appropriate error number appears — see Appendix C for an explanation and possible solutions for errors.

Rename File by

This function is handy for renaming the second of two identically named files on a diskette, since you choose the file to be re-

named by number, not by name. It can be used to rename a file whether or not the file is locked. To rename the first of two identically named files on a diskette, simply use DOS Menu selection E., RENAME FILE.

After selecting the Rename File by # function, place the diskette containing your identically named files in the current drive and press [Start]. A numbered directory (as described in the Unerase File subsection) appears. Next, enter a filename for the file to be renamed — the directory information for that file (also described in the Unerase File subsection) appears. In addition, you are prompted to enter the **New file name.** Type in the new filename, following normal DOS conventions — you may use up to eight characters, optionally followed by a period and an extender of up to three characters — and press [Return].

When the file has been successfully renamed, the utility prompts you to press [Start] to return to the DISKFIX.COM menu.

Quit To DOS

After you select this option, the prompt **Return to DOS 2.5 (Y/N)** appears. Press **Y** to return to DOS 2.5 or **N** to return to the DISKFIX.COM menu.

SETUP.COM

When you load the SETUP.COM utility, this menu appears:

Choose an option:

1. Change current drive number

2. Change system configuration

3. Set up an AUTORUN for Boot

0. Quit — Return to DOS

Your choice (0, 1, 2, OR 3) ?

Menu selections 1 and 0 are used for "housekeeping" purposes. The two main functions of this utility are menu selections 2 and 3. Press the number key that corresponds to the function you wish to use.

Change Current Drive Number

Options 2 and 3 on the SETUP menu use and change data and files on the diskette in the current drive, which is always Drive 1 unless you change it using option 1. If you have only one disk drive, you cannot change the current drive number. With multi-drive systems, the current drive number can be 1, 2, 3, or 4, depending on how many drives you have.

Change System Configuration

When you select this option from the SETUP menu, you are presented with this menu:

Change System Configuration

Current System Configuration:

Active Drives: 1 2

Up to 3 files open simultaneously.

Disk writes occur with verify.

Do you want to change any part of that configuration (Y/N) ?

If you press **Y**, you will be presented with the opportunity to change each of the options displayed.

Since under DOS 2.5 each "active" drive uses 144 bytes of memory and each possible simultaneously open file uses a 128-byte buffer, you may be able to save memory with this option by changing the number of drives and files in use. Alternatively, you may have a program that uses more than the "normal" two drives and three files. By enlarging the capacity of DOS, you lose a little memory but gain a lot of disk power. See Appendix G for further information on this topic.

Keep in mind, however, that if you choose four active drives and you are using an Atari 130XE Computer with the RamDisk, or if you choose only one open file buffer, you will not be able to use certain DOS Menu functions:

C. COPY FILE

D. DELETE FILE

O. DUPLICATE FILE

When DOS 2.5 writes to a diskette, it normally reads back (verifies) each sector as it is written. For faster disk output — although perhaps with slightly less reliability — use this option to omit the verifying process. See Appendix F for further information.

Making Changes Permanent

After you make changes to your system configuration, a New System Configuration screen such as this one appears:

New System Configuration

Current System Configuration:

Active Drives: 1

Up to 3 files open simultaneously.

Disk writes occur with verify.

Are you sure this configuration is what you want (Y/N)?

Current system configuration has been changed. Do you want to make these changes to the disk currently in drive 1 (Y/N) ?

The first prompt on this screen allows you to confirm your new configuration. The second prompt allows you to make the changes "permanent." If you type **Y**, the SETUP.COM utility writes your configuration changes to the DOS.SYS file on the diskette in the current drive. Otherwise, the changes are made only to the DOS currently in memory.

Note: If DOS.SYS is not on the diskette in the current drive, if DOS.SYS is locked, or if the diskette is write-protected, you will not be able to make permanent changes to the DOS System or Master Diskette.

Set Up An AUTORUN for Boot

When you boot your system with DOS 2.5, the first thing that happens is that the DOS.SYS file is loaded into memory and run. Before loading DUP.SYS or transferring control to BASIC or a cartridge, if present, DOS determines whether a file named AUTORUN.SYS exists on the boot disk. If so, the file is loaded into memory; if it was saved with a run vector it is also run (see DOS Menu selections K and L, BINARY SAVE and BINARY LOAD, in Section 3).

You can use option 3 on the SETUP utility menu to create an AUTORUN.SYS file on the diskette in Drive 1 that will perform either or both of these two functions:

- Load the RS232 handler from an ATARI 850 Interface Module (if the module is connected and turned on when you boot your system), and/or—
- Load and RUN a BASIC program from the boot disk.

When you choose option 3 from the SETUP menu, this screen appears:

```
Make an AUTORUN.SYS program file

When the disk currently in drive
     number 1 is next booted, what
     do you want to happen?

1. The RS232 (R:) drivers for the
   ATARI 850 Interface Module are
   loaded and made active.

2. A BASIC program will automatically
   load and RUN.

3. Both actions (1. and 2. above)
   will occur.

0. None — quit to main menu.

     Your choice (0, 1, 2, OR 3) ?
```

Press the number key corresponding to the function or functions you want your AUTORUN.SYS file to perform.

If you choose 1, the AUTORUN.SYS file is automatically written to the diskette in the current drive. If you choose 2 or 3, this screen appears:

```
Please enter the name of the BASIC
     Program that you wish to have
     automatically RUN when this disk
     is next booted.

Do NOT enter the drive specifier
     (i.e., do not use D:, D1:, etc.)
     but DO use the proper extension
     (e.g., .BAS, .SAV, etc.) if you
     SAVEd it with an extension.
```

REMEMBER: The BASIC program that you wish to 'AUTORUN' in this way MUST be SAVEd on the same disk which receives this AUTORUN.SYS program file!

Now enter your BASIC program's name here>>

As the screen reminds you, your BASIC program must be SAVEd on the diskette that will contain your AUTORUN.SYS file; otherwise, you will get an error message when you try to write your AUTORUN.SYS file to the diskette (though the SETUP utility will give you the opportunity to proceed anyway or abort).

Enter the filename of your BASIC program, but *do not* include the device spec, and press [Return]. The AUTORUN.SYS file is automatically written to the diskette in the current drive when you press [Return].

If the BASIC program isn't on the AUTORUN.SYS disk when it is booted (DOS.SYS must also be on the disk), or if it has not been SAVEd, you will get an error message.

APPENDIX M
ATARI 1050 DISK DRIVE SPECIFICATIONS

Control Logic	6507 Microprocessor
Diskette Format	Dual-density, single-sided, 5 1/4 inch diskette
Data Storage	127 K
Dimensions	
Height	3.50 Inches
Width	7.40 Inches
Length	12.00 Inches
Weight	6 pounds
Input Voltage	8.52 Volts AC ±12% @ 60 ±3Hz
Power Consumption	
Standby	15 Watts
Operation	30 Watts
Start Up	50 Watts
Temperature	
Operating	75.6-129.6F
Storage	21.6-165.6F
Relative Humidity	
Operating	20%-80%
Storage	5%-95%
Altitude	0-9842.5 feet

	Dual Density	**Single Density**	**RamDisk**
Number of Tracks:	40	40	N/A
Number of Sectors per Track	26	18	N/A
Total Number of Sectors	1,040	720	512
Number of Sectors available to DOS	1,023	719	511
DOS Overhead, in Sectors	13	12	12
Numbers of Sectors Usable for File Storage	1,010	707	499
Number of Bytes per Physical Sector	128	128	128
Number of Bytes of Overhead per Logical Sector	3	3	3
Number of Usable Bytes of File Storage per Drive	126,250	88,375	62,375

Encoding Method	MFM	FM
Transfer Rate	250,000BPS	125,000 BPS
Access Time		
Track to Track (maximum)	40 MS	40 MS
Motor Start (maximum)	1000 MS	1000 MS

INDEX

CUSTOMER SUPPORT

Atari Corp. welcomes any questions you might have about your 1050 Disk Drive or about any other ATARI Computer product. Write to:

Atari Customer Relations
P.O. Box 61657
Sunnyvale, CA 94088

Please write the subject of your letter on the outside of the envelope.

Or contact your local Atari User Group. They are an outstanding source of information on how to get the most from your ATARI Computer. To receive a list of User Groups in your area, send a self-addressed stamped envelope to:

ATARI User Group List
P.O. Box 61657
Sunnyvale, CA 94088

ERROR CODE NO	ERROR CODE MESSAGE
2	Insufficient Memory
3	Value Error
4	Too Many Variables
5	String Length Error
6	Out of Data Error
7	Line Number Greater Than 32767
8	Input Statement Error
9	Array or String DIM Error
11	Floating Point Overflow/Underflow Error
12	Line Not Found
13	No Matching FOR Statement
14	Line Too Long Error
15	GOSUB or FOR Line Deleted
16	RETURN Error
17	Syntax Error
18	VAL Function Error
19	LOAD Program Too Long
20	Device Number Larger Than 7 or Equal to 0
21	LOAD File Error
128	BREAK Abort
129	IOCB* Already Open
130	Nonexistent Device Specified
131	IOCB Write-Only
132	Illegal Handler Command
133	Device or File Not Open
134	Bad IOCB Number
135	IOCB Read-Only Error
136	End of File
137	Truncated Record
138	Device Timeout
139	Device NAK (not acknowledged)
140	Serial Frame Error
141	Cursor Out of Range for Particular Graphics Mode
142	Serial Bus Data Frame Overrun
143	Serial Bus Data Frame Checksum Error
144	Device Done Error (invalid "done" byte)
145	Illegal Screen Mode
146	Function Not Implemented in Handler
147	Insufficient RAM for Selected Graphics Mode
160	Drive Number Error
161	Too Many OPEN Files (no sector buffer available)
162	Disk Full (no free sectors)
163	Unrecoverable System Data I/O Error
164	File Number Mismatch
165	File Name Error
166	POINT Data Length Error
167	File Locked
168	Command Invalid (special operation code)
169	Directory Full (64 files)
170	File Not Found
171	POINT Invalid
172	Attempt to Append to DOS 1 File Using DOS 2.5
173	Bad Sectors at Format Time

*IOCB refers to Input/Output Control Block.

Sunnyvale, CA 94086

Printed in U.S.A.
CO72033-001 REV. A