

\$19.95

INSIDE ATARI® DOS

ATARI 8

**From The Editor's of COMPUTE! Magazine and
Optimized Systems Software, Inc.**

INSIDE ATARI[®] DOS

Compiled by Bill Wilkinson,
Optimized Systems Software, Inc.

Published by **COMPUTE! Books,**
A Division of Small System Services, Inc.,
Greensboro, North Carolina

ATARI is a registered trademark of Atari, Inc.



Preface

This book contains the only complete and official listings for the disk File Manager System (FMS) commonly known as "Atari DOS 2.0S." You will note that we have clearly stated that the purchase of this book does *not* entitle you to make, sell, give, or otherwise distribute copies of either the original Atari DOS 2.0S or any modified version you may produce as a result of using this book.

By way of information, should you desire to produce and distribute a modified version of this product (e.g., to support a new disk drive), you *must* sign a contract and licensing agreement with the party who owns the rights to grant such licenses for non-exclusive uses. Currently, Optimized Systems Software is the only entity able to grant such licenses.

Some of you may find it strange that the publishers of **COMPUTE!** magazine are publishing this book. You might wonder why Atari, Inc., hasn't released this information before. Why can you only obtain distribution rights from Optimized Systems Software? For the answers to these and other questions we present the following Introduction, an historical perspective on the development of the systems software for the Atari Home Computers.

All reasonable care has been taken in the writing, testing, and correcting of the text and of the software within this book. There is, however, no expressed or implied warranty of any kind from the authors or publishers with respect to the text or software herein contained. In the event of any damages resulting from the use of the text or the software in this book, the authors or publishers shall be in no sense liable. Please review the important cautions noted in Appendix A regarding the use of this book.

Copyright © 1982 text, Small System Services, Inc.

Copyright © 1978, 1979, 1980, 1982 program listings, Optimized Systems Software, Inc.

All rights reserved. Reproduction or translation of any part of this work beyond that permitted by sections 107 and 108 of the United States Copyright Act without the permission of the copyright owner is unlawful.

Printed in the United States of America

ISBN 0-942386-02-7

10 9 8 7 6 5 4 3 2

Table of Contents

Preface	Page ii
Introduction: Being a History of Two Births: "Coleen" and "Candy"	Page iv
Chapter One: Atari DOS Overview	Page 1
Chapter Two: Disk Organization	Page 10
Chapter Three: FMS File Control Blocks (FCB)	Page 15
Chapter Four: FMS Initialization	Page 17
Chapter Five: FMS Entry	Page 22
Chapter Six: FMS Exit	Page 23
Chapter Seven: Device Dependent Commands	Page 25
Chapter Eight: FMS Open Routines	Page 31
Chapter Nine: FMS Close Routines	Page 34
Chapter Ten: The GET BYTE Routine	Page 36
Chapter Eleven: The PUT BYTE Routine	Page 37
Chapter Twelve: Burst I/O	Page 38
Chapter Thirteen: Reading the Directory as a File	Page 40
Chapter Fourteen: Sector I/O Routines	Page 42
Chapter Fifteen: File Name Decode Routine	Page 46
Chapter Sixteen: Directory Searching	Page 48
Chapter Seventeen: Write Next Sector	Page 50
Chapter Eighteen: Read Next Sector	Page 51
Chapter Nineteen: Get and Free Sector Routines	Page 53
Chapter Twenty: The Boot Process	Page 55
Chapter Twenty-One: Maintaining the Boot Record	Page 57
Atari DOS 2.0S	Page 59
Appendix A: An Intermediate User's Guide To This Book	Page 102

COMPUTE! Books is a division of Small System Services, Inc.,
Publishers of **COMPUTE!** Magazine
Editorial Offices are located at:
625 Fulton Street, Greensboro, NC 27403 USA. (919)275-9809

Optimized Systems Software, Inc., is located at:
10379 Lansdale Avenue, Cupertino, CA 95014 USA. (408)446-3099.

Introduction

BEING A HISTORY OF TWO BIRTHS “COLEEN” AND “CANDY”

I don't know exactly when the concept of the Atari Computer was developed within the corporate mind of Atari, Inc., nor do I know all of the people responsible for nursing that concept into reality. The following history covers the relationship with Atari, Inc., during the evolution of the system software.

Sometime in early 1978, when the Atari 800 and 400 were still called “Coleen” and “Candy” and were still in the breadboard stages, Atari bought a copy of the source for Microsoft 8K BASIC. This version of BASIC was fundamentally the same product that was implemented by Commodore in the early PETs, was used by OSI, and was a close ancestor of Applesoft. Six months and many, many Atari man-hours later, that 8K BASIC was *almost* functioning properly on the Atari prototypes. But buying source for a program buys you just that: source. Generally, you also receive little documentation, sometimes obscure code, no guide to modification, and no real support. What to do? The products were due to be shown in early January, 1979, at the Consumer Electronics Show (CES) in Las Vegas, Nevada.

Enter Shepardson Microsystems, Inc. (SMI), my employer at that time. Though little known by the microcomputer public, SMI had already produced some very successful, private labeled microcomputer software. Among our better-known efforts were the original Apple DOS, Cromemco 16K Extended BASIC, and

Cromemco 32K Structured BASIC (just being completed at that time). Also, we had done some work for Atari on a custom game processor. (Which used a 12-bit ROM and 5-bit RAM configuration and was well received at Atari, but never produced.)

Coincidentally, about that same time SMI had *also* purchased source for Microsoft 6502 BASIC. After producing Apple's DOS, we had the bright idea of mating the Apple II peripheral bus with the KIM/SYM/AIM system bus (and it still seems like a good idea to us, but ...). The idea was to provide a disk system (Apple's) to the Single Board Computer market. Needing a BASIC to sell with the system, we plunked down a few grand and purchased Microsoft's. Though it looked to us like it would be difficult to modify, we were intending to resell it with a minimum of changes, so it seemed appropriate.

A New BASIC?

Re-enter Atari, some time in the late summer of 1978, asking if SMI could help them. With Microsoft BASIC? Well ... we really didn't want to, but ... Could we propose a new BASIC? We talked. And had meetings, and a study contract, and more meetings, and finally we wrote a specification for a 10K, ROM-based BASIC. (I still have a copy of that spec, and it's amazing how little the final version deviated from that original.)

Of course, in the middle of all these discussions, Atari naturally divulged how their (truly superb) ROM-based Operating System would interface both with BASIC and with various devices. Somewhere in here, my memory of the sequence of events and discussions becomes a little unclear, but suffice it to say that we found ourselves making a bid on producing not only a BASIC for Atari, but also the File Manager (disk device driver) which would change Atari OS to Atari DOS.

Sometime in late September, 1978, the final proposal was made to Atari, and it was accepted by them shortly thereafter. In mid-October, 1978, we received the go-ahead. The project leader was Paul Laughton, author of Apple DOS. The bulk of the work ended up being done by Paul and Kathleen O'Brien. Though I was still involved in the finishing touches on Cromemco BASIC, I take credit for designing the floating point scheme used in Atari BASIC. Paul Krasno implemented the math library routines following guidelines supplied to us by Fred Ruckdeschel (author of the acclaimed text, *BASIC Scientific Subroutines*). And, of course, much credit must go to Mike Peters, our combination keypuncher/computer operator/junior programmer/troubleshooter.

Since we obviously couldn't have the Atari machines to work on (they hadn't been built yet), the first step was to bring up an emulator for Atari's CIO ("Central Input-Output," the true heart of Atari's OS) on our Apple II systems. With Paul Laughton leading the way (and doing a lion's share of the work), the pieces fell together quickly. "Little" things had to be overcome: the cross-assembler was modified to handle the syntax table pseudo-ops, the 256-byte Apple disk sectors had to be made to look like 128-byte Atari sectors, the BASIC interpreter seemed to function, but was waiting for the floating point routines. And there are funny things to tell of, also. Like our cross-assembler, running on an IMP-16P (a 1973 vintage, 16-bit, bit-sliced PMOS microprocessor) that used keypunched cards for input, a floppy disk (with no DOS) as temporary storage, and a paper tape punch as output.

Somehow, Kathleen and Paul guided the two programs unerringly toward completion. On December 28, 1978, Atari's purchasing department at last delivered a signed copy of the final purchase order. It called for delivery of both products by April 6, 1979. There was a clause which provided for a \$1,000 per week incentive (if we finished early) and penalty (if we finished late). What is especially humorous about that December 28th date is that the first working versions of *both* BASIC and FMS had *already* been delivered to Atari over a week before! That is *fast* work.

Fortunately, then, Atari took their new Atari BASIC to CES. Unfortunately, there was a limit on the amount of incentive money collectible. Oh, well.

In the months that followed, SMI fixed bugs, proofread manuals, and worked on other projects (including the Atari Assembler/Editor, which was mostly Kathleen's effort). The nastiest bugs in BASIC were fixed by December, 1979, but it was too late: Atari had already ordered tens of thousands of BASIC ROMs. The FMS bugs were easier to get fixed, since DOS is distributed on disk.

In mid-1980, Paul Laughton once again tore into FMS. This time, he modified it to handle the ill-fated 815 double-density disk drive and added "burst I/O" (and there will be much more about both these subjects in the technical discussion that follows).

In late 1980, and early 1981, Bob Shepardson, owner of Shepardson Microsystems, Inc., decided that the pain and trouble of having employees wasn't justified by the amount of extra income (if any) that he derived. Though we still occasionally function in a loose, cooperative arrangement, the halcyon days of SMI seem to be over.

A New Beginning

I negotiated with Bob Shepardson for his rights to the Atari products (FMS, BASIC, and the Assembler/Editor) and their Apple II counterparts. Thankfully, Atari had purchased from SMI only a non-exclusive right to distribute these products. SMI had retained the rights to license other users on a similar non-exclusive basis (and, indeed, SMI sold a version for the Apple II during most of 1980).

So now it was frantic time again: this was February 25, 1981, and the West Coast Computer Faire was April 3rd. But our brand new company, Optimized Systems Software, arrived on time, bringing with it BASIC A+, OS/A+ and EASMD. All three were enhanced, disk-based versions of the original Atari programs (and, in fact, derived some of their enhancements from the previous OSS Apple II products).

The products have been well received by the Atari user community, in part due to the fact that they are truly compatible, yet enhanced, versions of standard Atari software.

Why This Book?

The decision to publish these listings was not an easy one to make; and it is, in its own way, an historic occasion. After all, have you ever seen anyone offering source or listings of CP/M, the most popular of all computer operating systems? Since Atari, to their credit, has honored the original agreement with SMI and not released either source or listings without permission, the responsibility for doing so seemed to rest with OSS.

But Atari has set a powerful precedent by publishing the listings of DUP (their portion of DOS 2.0S) and the OS ROMs. The clamor from Atari users for the source for FMS finally even reached us, so we have bowed to the inevitable, and honored the same commitment that Atari has made: to release as much information and aid as possible to the user community.

We hope that the users will appreciate these efforts and, in turn, respect our rights and Copyrights. As long as there is a mutual respect and benefit, you, the user, can expect continued support.

About This Book

With the release of this book, the dedicated Atari enthusiast can examine all the inner workings of Atari DOS and modify his (or her) system to his heart's delight. Rather than simply publish listings, we have chosen also to provide a complete guide to the workings of FMS.

Although the listing itself is relatively clear and commented, all

but the most expert would have trouble plowing through some of the tortuous logic necessary in such a program. The guide included here describes all aspects of the FMS, including the external view, the charts and tables, the various interfaces, and (in copious detail) the functions of the individual subroutines (including complete entry and exit parameters).

There is much of value here even for the person who never intends to modify Atari DOS. We feel that FMS is a fairly well-structured, relatively sophisticated, system level assembly language program. We hope that most users will gain by the insights presented here.

We would welcome any notes you would care to send pointing out errors either in the DOS or in this book.

Bill Wilkinson
Optimized Systems Software
Cupertino, California
February, 1982

Chapter One

ATARI DOS OVERVIEW

The standard Atari Disk Operating System, DOS 2.0S, consists of four separate elements, ranked as follows in order of their “visibility” to the average DOS user.

1. DUP – Disk Utility Package
2. CIO – Central Input/Output
3. FMS – File Management System
4. SIO – Serial Input/Output

It is helpful to understand the entire Input/Output (I/O) process. While this book is intended to give detailed information on the workings of FMS, this overview will attempt to at least show how the four elements of DOS are connected. To this end, we would first call your attention to Figure 1. This figure is, itself, an overview of the entire Atari I/O system, including indications as to how and where data and control flows between the various elements thereof. Figures 1-1 through 1-4 show “close-ups” of portions of this diagram as they relate to the four elements of DOS.

In these figures, the rectangular boxes represent system elements, and are appropriately labeled. The wide, lettered arrows represent the flow of data (via buffers, control blocks, or even registers) between the various elements. The narrow, numbered arrows show how and where control, and control information, is transferred.

1-1. Disk Utility Package

DUP (which shows as “DUP.SYS” in a disk directory listing) is the most obvious and visible element of Atari DOS. DUP’s function is to provide the user with keyboard access to the various file management functions in FMS. It does so via the menu which is displayed when, for example, the user keys “DOS” from BASIC. Actually, the menu offers several options which are not directly a part of the FMS (e.g., copy and duplicate files). Refer to the *Atari Disk Operating System II*

CHAPTER ONE

Reference Manual (part number C016347) for more information.

DUP is *not* an integral part of FMS. DUP may be relatively easily replaced with a program of the user's choice. In fact, our own OS/A+ does exactly that: instead of a menu, the user is given a command-driven keyboard interface to the other elements of DOS.

DUP is not even a privileged portion of DOS (excepting, perhaps, for needing to know a little of the internals of FMS when it performs a Duplicate Disk function). Any user application program (and that includes Atari BASIC, BASIC A+, EASMD, and many, many more) interacts the same way DUP does. Figure 1-1 shows the "proper" flow of control in DOS. Note that DUP transfers control only to CIO, which, in turn, transfers control to FMS and thence to SIO. An application program which maintains this protocol should be able to perform correctly in any Atari system, regardless of the revision of the OS ROMs and/or FMS.

Of course, control is not the only thing which DUP must transfer. It must also tell CIO where its data is and what to do with it. Refer to Figure 1-2 for a diagram of the complete application/CIO interface (again, it is labeled in this way because DUP is just another application program as far as the rest of DOS is concerned). CIO always expects an Input/Output Control Block (IOCB) and usually (i.e., for all but the simplest operations) needs a buffer into or out of which it may perform its operations.

1-2. Central Input/Output

CIO is actually the heart of the entire Atari Computer. It is less than 800 bytes long and yet serves to handle virtually all the input and output which takes place in the computer. CIO is a part of the Atari "OS ROMs," the 10K byte package which also houses the floating point routines, the default character set, the interrupt handlers, and several device drivers.

The entire set of operations summarized in Figure 1-2 is covered in detail in the Atari OS Manual (C01655) and will be covered only briefly here. Readers of **COMPUTE!** will also find some helpful material on this subject in issues #18 through #21 (November, 1981, through February, 1982) in the "INSIGHT: ATARI" columns.

In order to allow easy control and data flow, CIO is written to expect and provide for eight Input/Output Control Blocks (IOCBs) which are used to pass the information needed to process the various kinds of I/O requests. An application places the necessary command and control information in an IOCB which it selects (data path A). If a buffer is required, the application must provide one (data path C)

and place its address into the IOCB. When ready to execute the I/O command, the application places the IOCB number (times 16) in the 6502's X-register (data path C) and executes a JSR call to CIO (control path 1). Note that a few command variations may pass data via the 6502's A-register, but we may consider that simply a special case location of the user's buffer.

When CIO receives control, it examines the information in the IOCB (and, for some operations, in the user buffer) to determine what actions it is to perform. Generally, this action requires the execution of a device handler routine.

A device handler (interchangeably known as a *device driver*) is a system routine that performs I/O operations for a specific device (or class of devices). Examples of device handlers include the "P:" driver (the printer) and the "E:" driver (the screen/keyboard editor). Figure 1-3 illustrates the interface between CIO and the various device handlers. Note that FMS is simply another device handler as far as CIO is concerned, having been given the name "D:".

All device drivers are required to contain a table of address pointers (known as the Device Vector Table) to various specific routines within themselves, including a device OPEN routine, GET CHARACTER routine, etc. The name of a device and the address of this table is placed in CIO's Device Handler Table. When an application program makes an I/O request to CIO for a specific device, CIO searches the Device Handler Table for the given name and corresponding Device Vector Table address. With the thus-located vector table, CIO can then call the appropriate device handler routine (via a JSR, along control path two of Figure 1-3).

1-3. File Management System

As stated above, FMS is actually simply another device driver as far as CIO is concerned. The control and data flows shown in Figure 1-3 are equally valid for all device drivers in the Atari system. Note that many of the drivers in the default ("as-shipped") system reside entirely within the so-called OS ROMs. Although it resides in RAM, what is somewhat unique about FMS is that the Atari system initialization code contains a segment of "boot" code which loads FMS into memory upon power-on.

FMS is the system device handler for all I/O operations that specify the device name "D" (including "D1:", "D2:", etc.). In order to perform its functions, FMS examines the data in the specified IOCB (data path F). It may also examine, read, or write data to or from the user-supplied buffer (data path I). Data path H is used to pass

the IOCB-designator (again, via the X-register) and single-byte transfer data (via the A-register).

FMS is called upon to perform a variety of tasks, including all disk I/O, file renaming, protecting, deleting, etc. Since the rest of this book consists of a listing of FMS along with detailed explanations of all sections thereof, we will not now dwell on the inner workings of FMS.

However, we do need to note that, in order to perform its work, FMS must transfer data to and from the disk. FMS accesses the disk drive via SIO, the fourth element of DOS.

1-4. Serial Input/Output

SIO is the name given to the component of DOS which drives and controls the Atari serial I/O bus and the various peripherals (disk, printer, modem, etc.) which are placed on that bus. Figure 1-4 illustrates the interface between FMS and SIO, but it could just as well serve to show (for example) how the printer driver talks to the various Atari printers.

The SIO is primarily driven by a request placed in SIO's Device Control Block (DCB) by the device handler (data path K) followed by a transfer of control (control path three) via a JSR. SIO uses the information in the DCB (data path M) to determine what it needs to do. If the DCB specifies a serial bus data transfer (as opposed to, for example, a status request), then the address of the data buffer must also be passed (via a field in the DCB). For example, the FMS buffer shown is accessed via data paths J (from FMS) and L (from SIO).

Although SIO only understands the single system DCB, the buffer specified may be located anywhere in memory. FMS takes advantage of this to implement "burst I/O" (discussed in section 12), which has SIO transferring data directly to or from the user's buffer (data path E).

Since the actual disk data transfer occurs in fact within the 810 disk drive and, since SIO communicates to the drive via data path N, one might reasonably argue that the disk drive constitutes a fifth component of DOS. However, because the disk drive functions are preprogrammed in ROM, and because SIO implements the only method of accessing the disk (as well as most other peripherals), then, for all practical purposes, even machine language software may treat SIO as the last link in the I/O chain on the Atari Computers.

Once again, we remind you to study Figure 1. In the following dissertation and dissection of FMS, we shall refer to this chart often.

Figure 1 (entire)
DOS Data And Control Flow

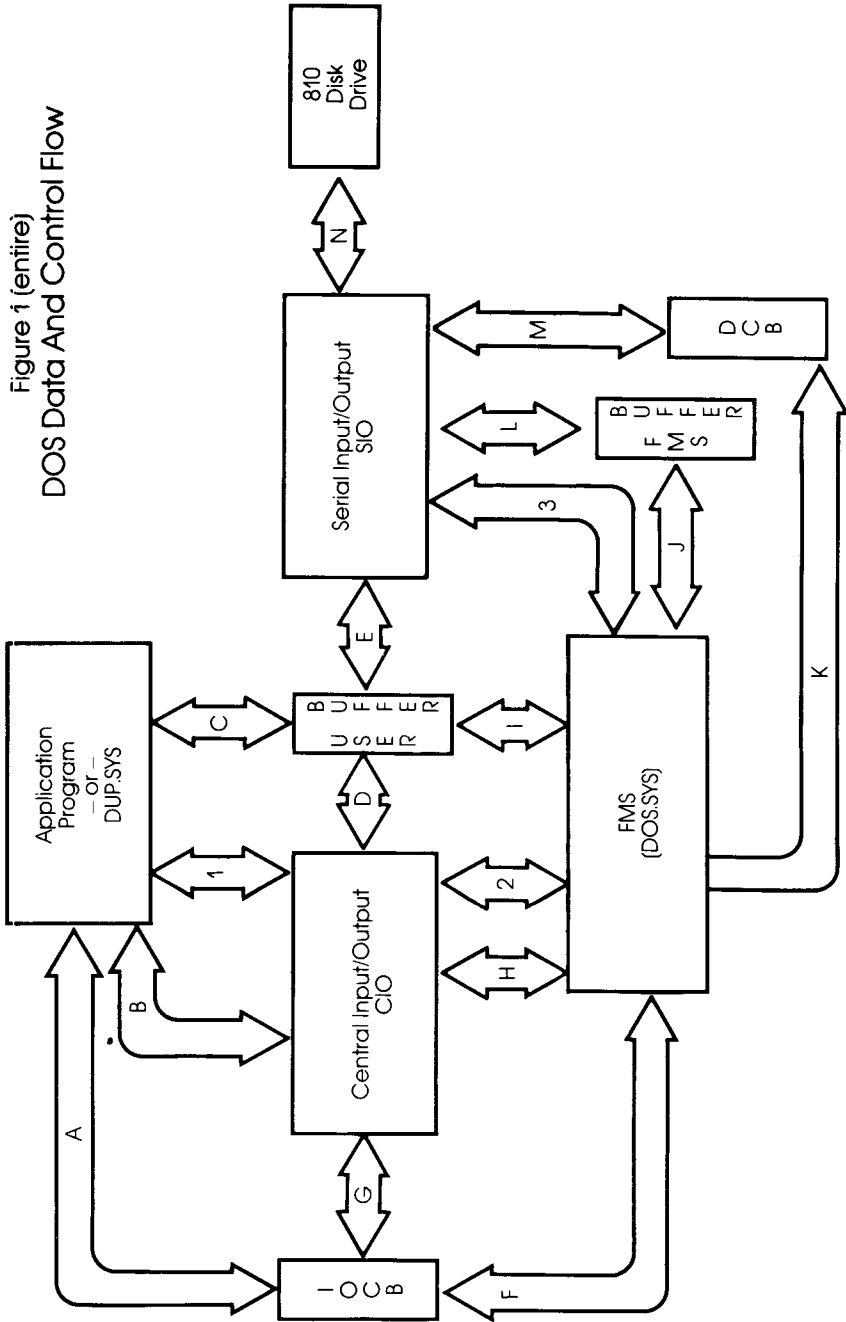


Figure 1-1
DOS Control Flow

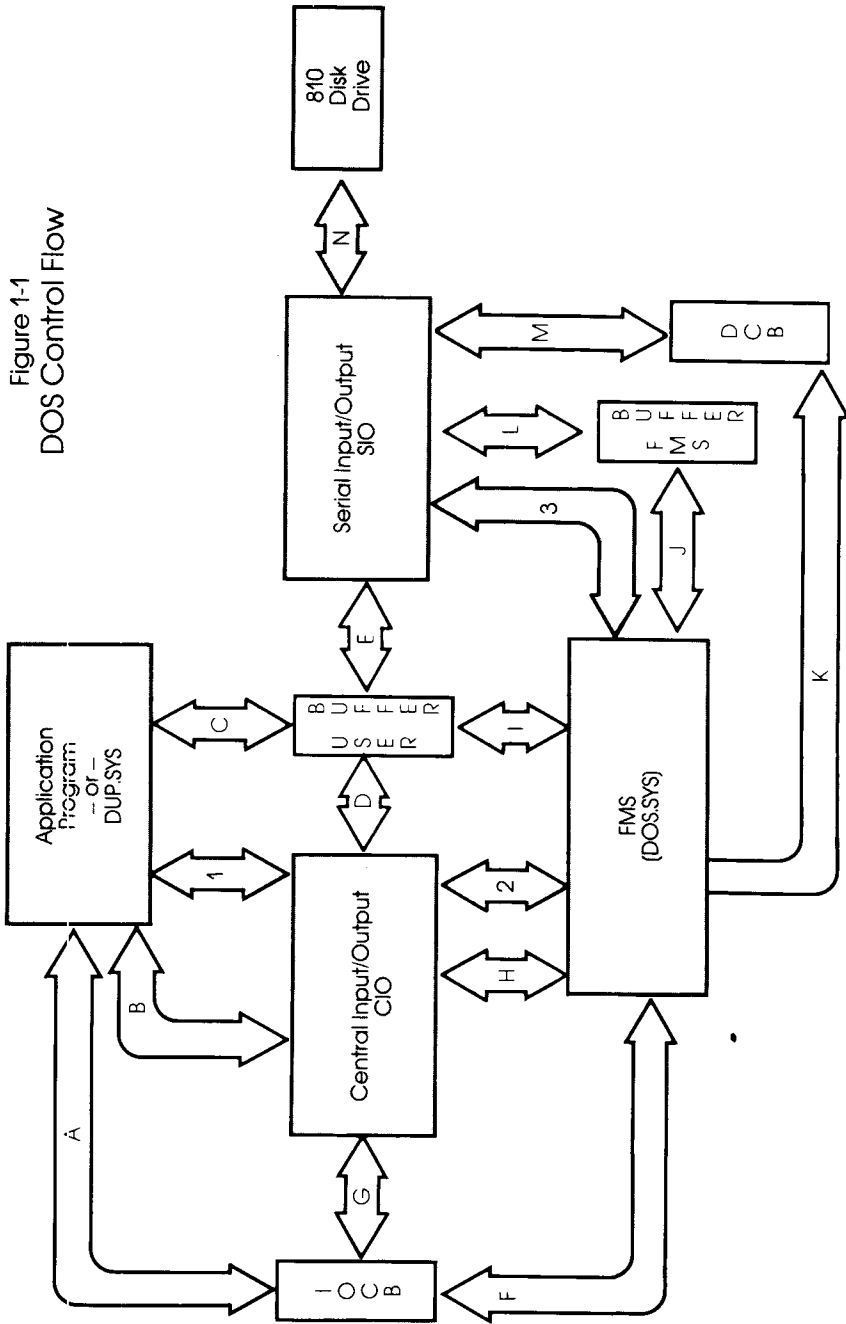


Figure 1-2
Application/CIO interface

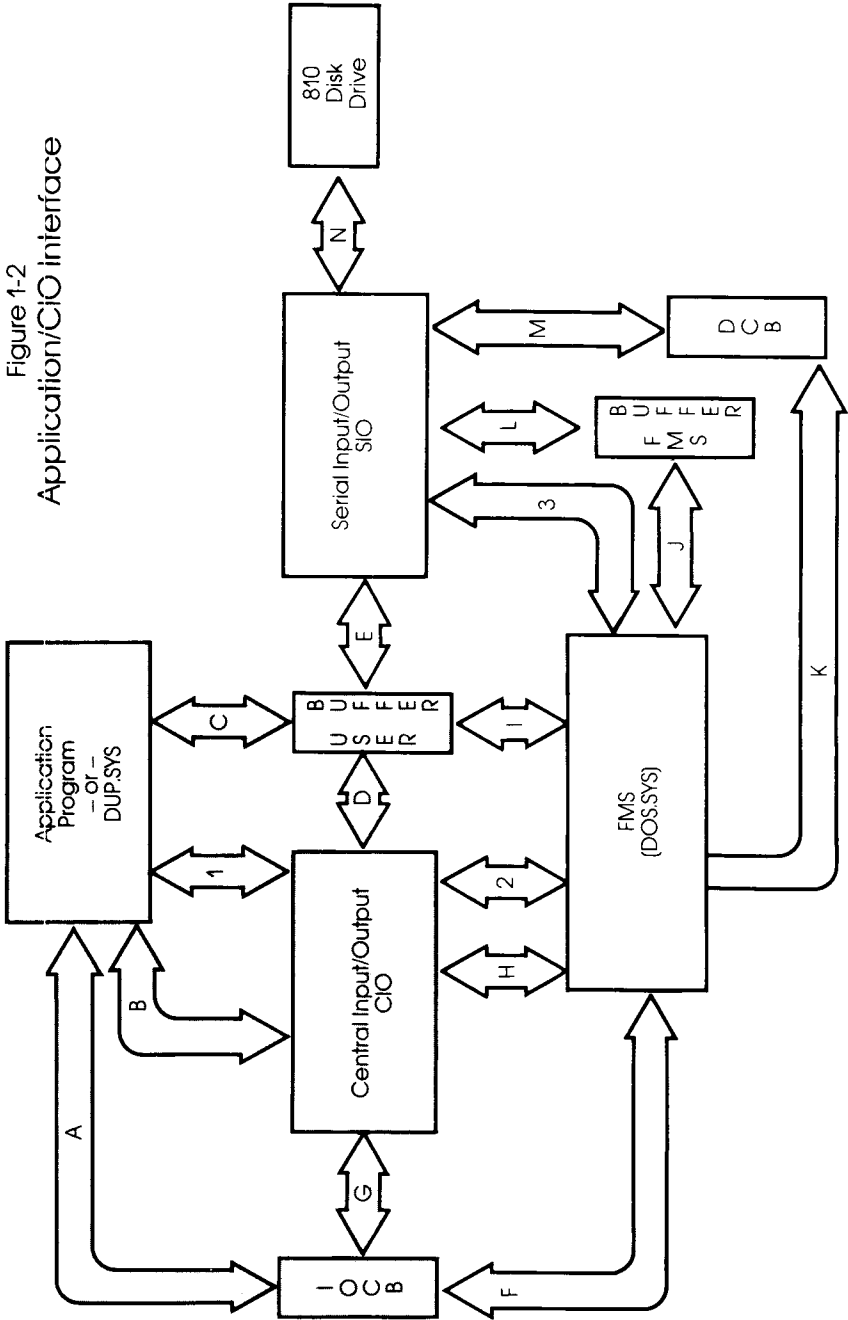


Figure 1-3
CIO-Device Handler Interface

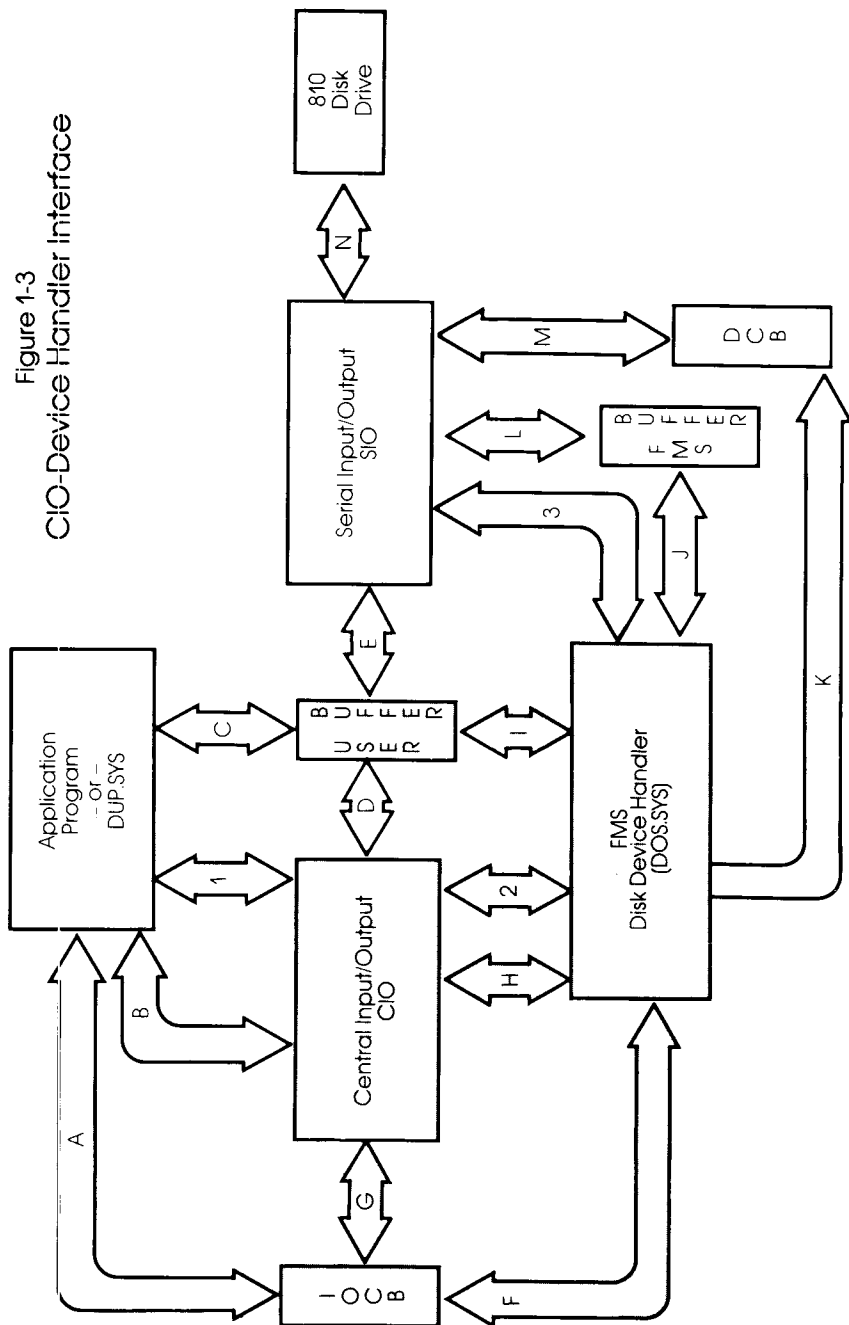
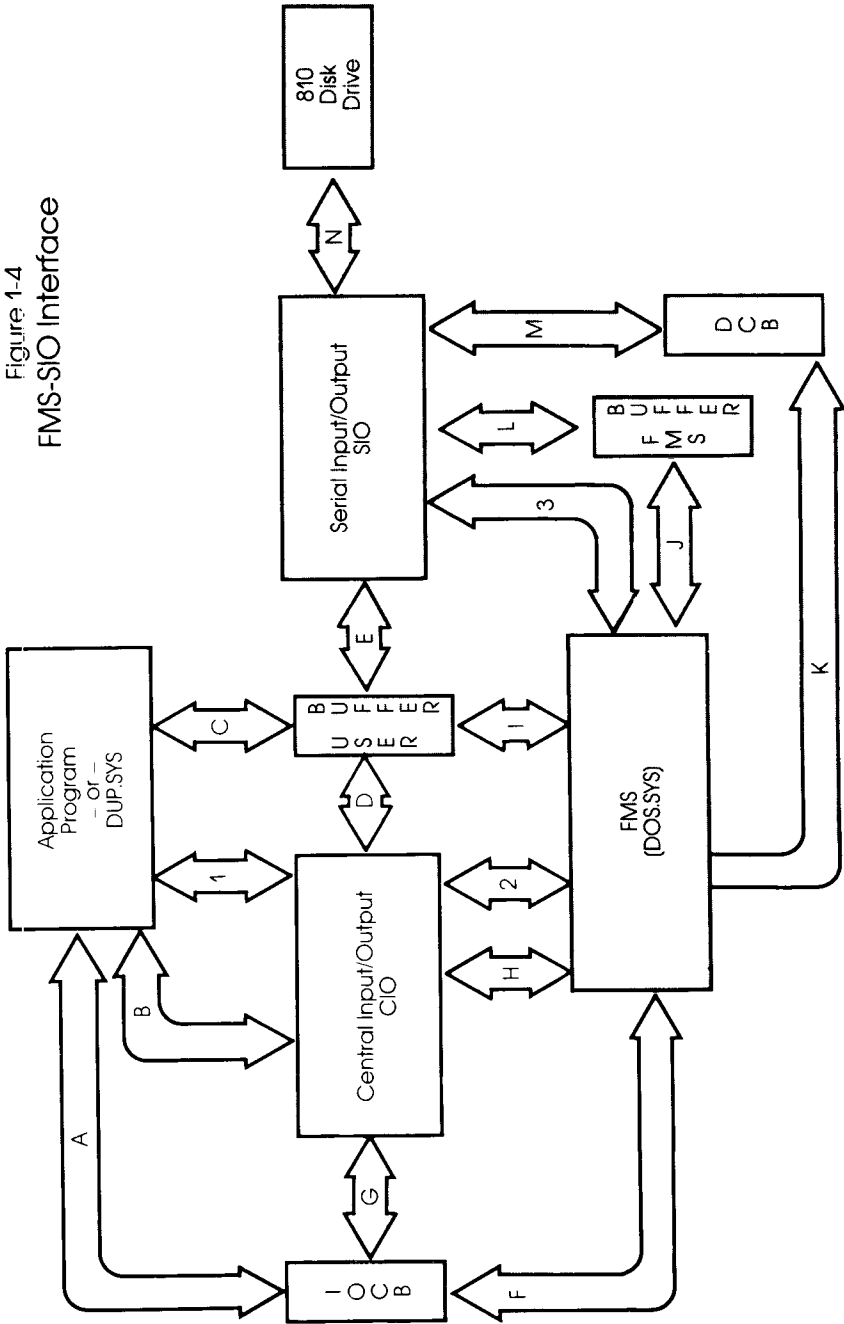


Figure 1-4
FMS-SIO Interface



Chapter Two

DISK ORGANIZATION

The purpose of FMS is to organize the 720 data sectors available on an 810 diskette into a system of named data files. FMS has three primary data structures that it uses to organize the disk: the Volume Table of Contents, the Directory, and Data Sectors. The Volume Table of Contents is a single disk sector which keeps track of which disk sectors are available for use in data files. The Directory consists of directory sectors. It is used to associate file names with the location of the files' sectors on the disk. Each Directory entry contains a file name, a pointer to the first data sector in the file, and some miscellaneous information. The Data sectors contain the actual data and some control information that link one data sector to the next data sector in the file. Figure 2-1 illustrates the relation between the Directory and the Data files.

Disk Directory

The Directory starts at disk sector \$169 and continues for eight contiguous sectors, ending with sector \$170. These sectors were chosen for the directory because they are in the center of the disk and therefore have the minimum average seek time from any place else on the disk. Each directory sector has space for eight file entries. Thus, it is possible to have up to 64 files on one disk.

A Directory entry is 16 bytes in size, as illustrated by Figure 2-2. The directory entry flag field gives specific status information about the current entry. The directory count field is used to store the number of sectors currently used by the file. The last eleven bytes of the entry are the actual file name. The primary name is left justified in the primary name field. The name extension is left justified in the extension field. Unused filename characters are blanks (\$20). The Start Sector Number field points to the first sector of the data file.

Data Sectors

A Data Sector is used to contain the file's data bytes. Each 128 byte data sector is organized to hold 125 bytes of data and three bytes of

control information as shown in Figure 2-3. The data bytes start with the first byte (byte 0) in the sector and run contiguously up to, and including, byte 124. The control information starts at byte 125.

The sector byte count is contained in byte 125. This value is the actual number of data bytes in this particular sector. The value may range from zero (no data) to 125 (a full sector). Any data sector in a file may be a short sector (contain less than 125 data bytes).

The left six bits of byte 126 contain the file number of the file. This number corresponds to the location of the file's entry in the Directory. Directory entry zero in Directory sector \$169 has the file number of zero. Entry one in Directory sector \$169 has the file number one – and so forth. The file number value may range from zero to 63 (\$3F). The file number is used to insure that the sectors of one file do not get mixed up with the sectors of another file.

The right two bits of byte 126 (and all eight bits of byte 127) are used to point to the next data sector in the file. The ten bit number contains the actual disk sector number of the next sector. Its value ranges from zero to 719 (\$2CF). If the value is zero, then there are no more sectors in the file sector chain. The last sector in the file sector chain is the End-Of-File sector. The End-Of-File sector may or may not contain data, depending upon the value of the sector byte count field.

Volume Table Of Contents (VTOC)

The VTOC sector is used to keep track of which disk sectors are available for data file usage. The VTOC sector is located at sector \$168. Figure 2-4 illustrates the organization of the VTOC sector. The most important part of the VTOC is the sector bit map.

The sector bit map is a contiguous string of 90 bytes, each of which contains eight bits. There are a total of 720 (90 x 8) bits in the bit map – one for each possible sector on an 810 diskette. The 90 bytes of bit map start at VTOC byte ten (\$0A). The leftmost bit (\$80 bit) of byte \$0A represents sector zero. The bit just to the right of the leftmost bit (\$40 bit) represents sector one. The rightmost bit (bit \$01) of byte \$63 represents sector 719.

The fact that FMS interprets the bit map as representing sectors zero through 719 is a bug. The Atari 810 disk drive will not accept commands for sector zero. It will accept commands for sector 720. In other words, the bit map is skewed by one. The problem cannot be fixed now because there are already tens of thousands of diskettes whose bit maps are to be interpreted as representing sectors zero through 719, and because some savvy applications writers have taken advantage

CHAPTER TWO

of this feature. (A bug which generates useful side effects is known in the programming profession as a *feature*.) Sector 720 can never be used by FMS and is therefore available for miscellaneous purposes.

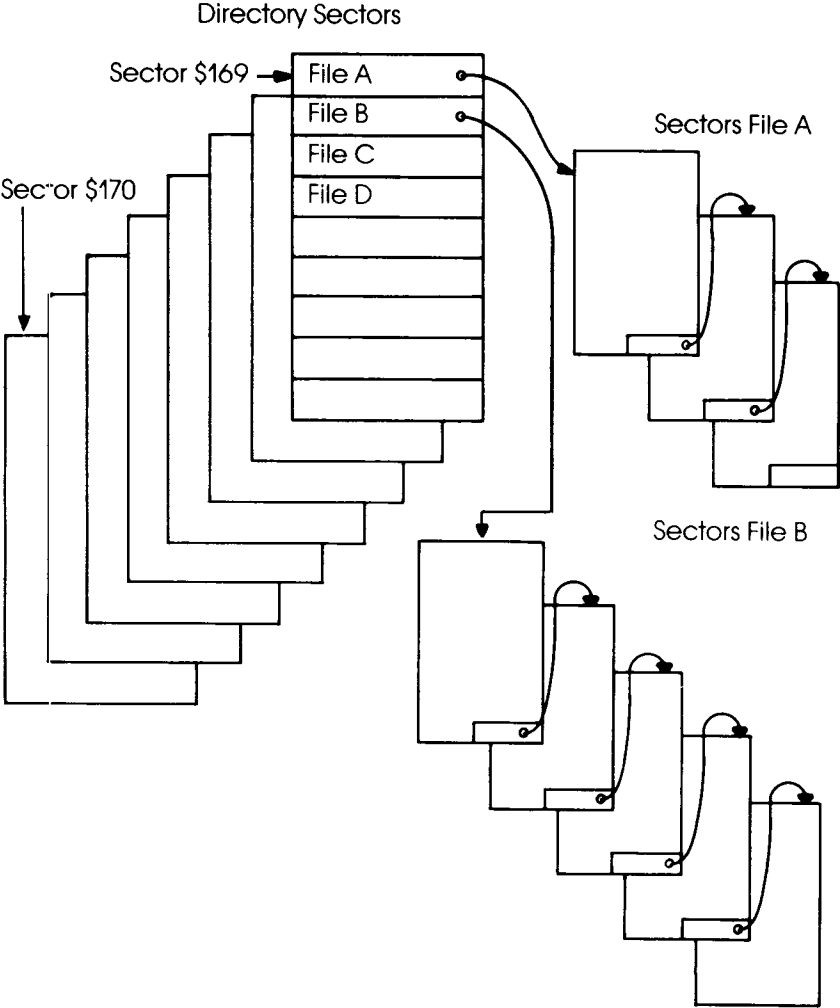


Figure 2-1

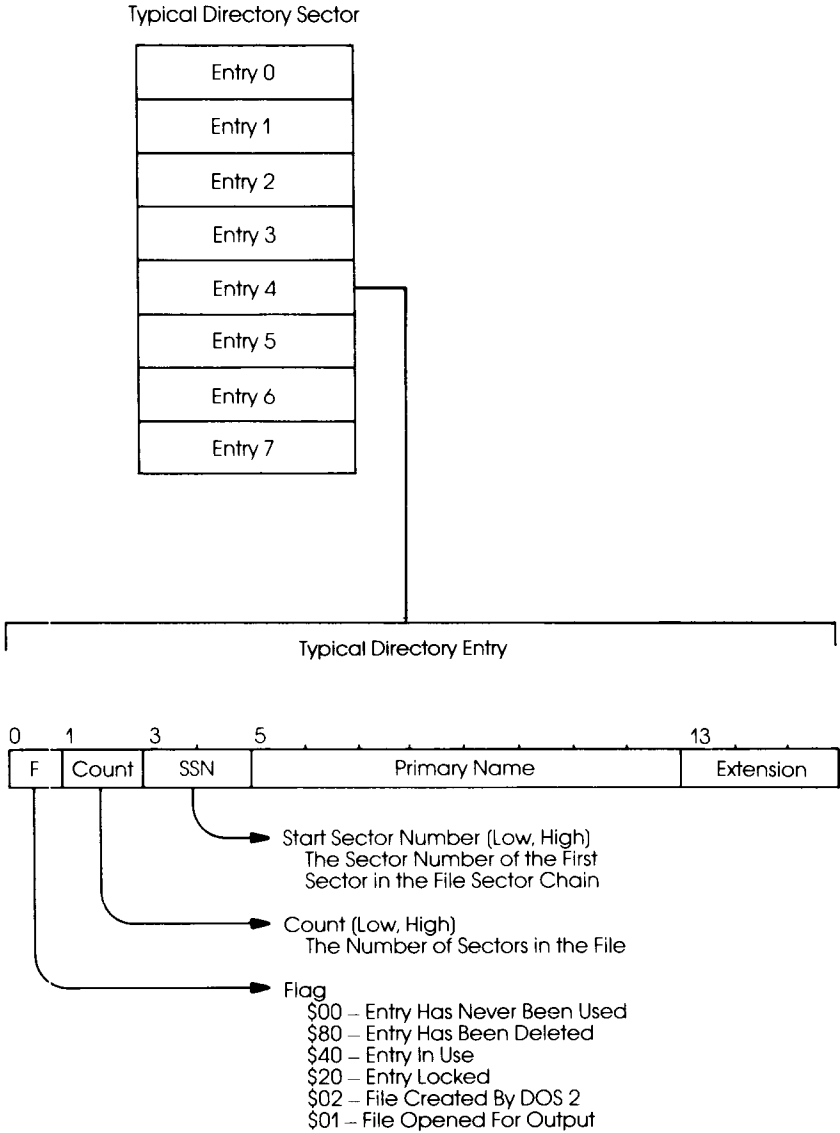


Figure 2-2

Chapter Three

FMS FILE CONTROL BLOCKS (FCB)

The FMS File Control Blocks are used to store information about files that are currently being processed. Each file that is being processed concurrently by FMS requires one FCB. Since the Atari system has eight IOCB's, FMS must be prepared to handle up to eight files concurrently, thus there are eight FCBs. The FCBs were designed to have a one-to-one correspondence with the IOCBs. When a file is to be processed with IOCB number three, FMS will use FCB number three for that file. When a file is to be processed with IOCB number five, FMS will use FCB number five for that file. Each FCB is the same size as an IOCB (16 bytes). The FCBs are located in a contiguous RAM area just like the IOCBs. When CIO calls FMS, the X register contains the displacement (IOCB number times 16) to the IOCB making the request. The FMS uses this displacement value to access both the IOCB information and the FCB information. Please refer to the listing at location \$1381 for the following discussion about the FCBs.

FCBFNO

The file number of the file currently being processed. The value (zero to 63) is shifted left two bits. When a file has been opened for reading, this value will be used to check for a file number mismatch in the data sectors. When a file is opened for write, this value will be placed in the file number field of the data sectors.

FCBOTC

Open Type Code. This value is used as a flag to indicate which mode the file has been opened for:

Input is \$04.

CHAPTER THREE

Output is \$08.
Update is \$0C.
Append is \$01.
Directory read is \$02.

FCBSLT

This is a flag used to indicate that the file being processed was created by DOS 1 rather than DOS 2. The Data Sector length byte has a different interpretation under DOS 1.

FCBFLG

This field is a working flag. If the value is \$80, then the file is eligible to acquire new data sectors. Files that are opened for Output or Append are eligible to acquire new data sectors. If the value is \$40, then the sector currently is in a memory buffer, has been modified, and needs to be written back to the disk.

FCBMNL

If the file is opened for Output or Append, this value will be either 125 or 253 depending upon the drive type. The 253 value is meant for the Atari 815 dual density drive. If the file is opened for Read or Update, then this value represents the number of data bytes that are in the data sector currently in a buffer. This value is obtained from the Data Sector data length field (byte 125 of the data sector.)

FCBDLN

This value points to the next data byte to be operated on in a data sector. If the file is opened for Output or Append, this value points to the next available (unused) data byte in the current data sector. If the file is opened for Update, then this value points to the next data sector byte to be either read or modified. If the file is opened for Input, then this value points to the next byte to be read.

FCBBUF

This value is an index into the sector buffer table. The sector buffer table is a list of buffer addresses. When a file is being processed, a sector buffer is required to hold data sectors. This field tells FMS which FMS buffer has been allocated to the file.

FCBCSN

The sector number of the sector currently in the buffer is stored in this field.

FCBLSN

The sector number of the next sector in the file chain is stored in this field.

FCBSSN

If the file has been Opened for Append, then this field contains the sector number of the start of the sectors to be appended to the file when the append file is closed.

Chapter Four

FMS

INITIALIZATION

DUP gets control whenever the system is booted or the RESET key is pressed. DUP will call the FMS initialization routine, DINIT at \$7E0.

DINIT

Functions:

- 1) Determine how many (and what type of) disk drives will be used.
- 2) Set up a drive table and allocate a drive buffer for each drive.
- 3) Allocate sector buffers and build the sector buffer table.
- 4) Clear the FCBs to zero.
- 5) Set MEMLO.
- 6) Enter the D: device into the Device Handler Table.
- 7) Exit to caller via RTS.

Drive Determination

The DRVBYT byte at \$70A is used to tell FMS how many disk drives will be used and what the drive number of the drives will be. The

CHAPTER FOUR

rightmost bit (bit \$01) indicates drive 1. The next left bit (\$02) indicated drive 2 – and so forth. If the bit is one, then the drive is to be used. If the drive is zero then the drive is not to be used. The code will allocate up the eight drives, even though the 810 hardware only has switches for drives 1,2,3 and 4.

If DRVBYT indicates that a drive is to be used, then FMS issues a status command to that drive to determine if it is active and what type (810 or 815) of drive it is.

Drive Allocations

The drive determination process sets up two tables (Figure 4-1). The first table is the DRVTBL. This table is indexed into by the drive number (minus one). If the value in the table is zero then the drive is not to be used. If the value is one, then the drive is an active 810 and requires one drive buffer. If the value is two, then the drive is an 815 and requires two 128 byte buffers.

The second table is the drive buffer table. The drive buffer table contains the address of the drive buffer to be used for each drive. This Drive Buffer will be used to hold the VTOC sector on the diskette in the drive. The table is separated into two sections: DBUFAL contains the least significant address byte and DBUFAH which contains the most significant address byte. The drive buffer table is also accessed by the drive number (minus one).

When a file is being processed, the Drive number is obtained from the IOCB Device Number field, ICDNO. The obtained value is decremented by one and is then used as an index into the Drive Tables. The Drive Type is copied from the DRVTBL entry to DRVTYP (\$12FE) for easy access by FMS. The Drive Buffer address is copied from the DBUFAL and DBUFAH table entries to the zero page drive buffer pointer, ZDRVA (\$45).

Sector Buffer Allocations

The SABYTE at location \$709 is used to inform FMS about the number of 128 areas to be allocated as sector buffers. One 128 buffer is required for each file which is to be processed concurrently on 810 drives. Two 128 byte buffers are required for each file which is to be processed concurrently on 815 drives.

The Sector Buffer Allocation table, SECTBL at \$1319, is used to indicate if a buffer is available for allocation to a file (Figure 4-2). If a buffer is available, the entry is set to zero. If the buffer is not available, the entry is a minus value. The table is 16 bytes in size and therefore can be used to allocate up to sixteen 128 byte buffers. During the

initialization process, entries which are to be unused are set to a minus value.

The Sector Buffer Address Table is a table of addresses which point to the individual sector buffers. The table is divided into two parts: SABUFL contains the least significant address byte, SABUFH contains the most significant address byte.

When a file is being processed, an available buffer number is found in SECTBL by search for a zero valued entry. The located buffer is allocated to the file by entering a minus value (\$80) into the table and placing the corresponding buffer number into the DCB buffer number field, FCBBUF. When the file processing is done, the buffer is deallocated by setting the SECTBL entry to zero.

Setting MEMLO

The Atari MEMLO location (\$2E7) is set after the FMS buffers have been allocated. The address of the last sector buffer allocated is incremented by 128. This value is then placed into MEMLO.

Device Handler Table Entry

The Device Handler Table (\$31A) is searched for a "D" entry or the first (from the top) empty entry. When an appropriate entry is found, FMS inserts (or reenters) "D" as a DEVICE NAME and sets the DEVICE vector entry to point to the FMS Device Vector table at DFMSDH (\$7CB).

CHAPTER FOUR

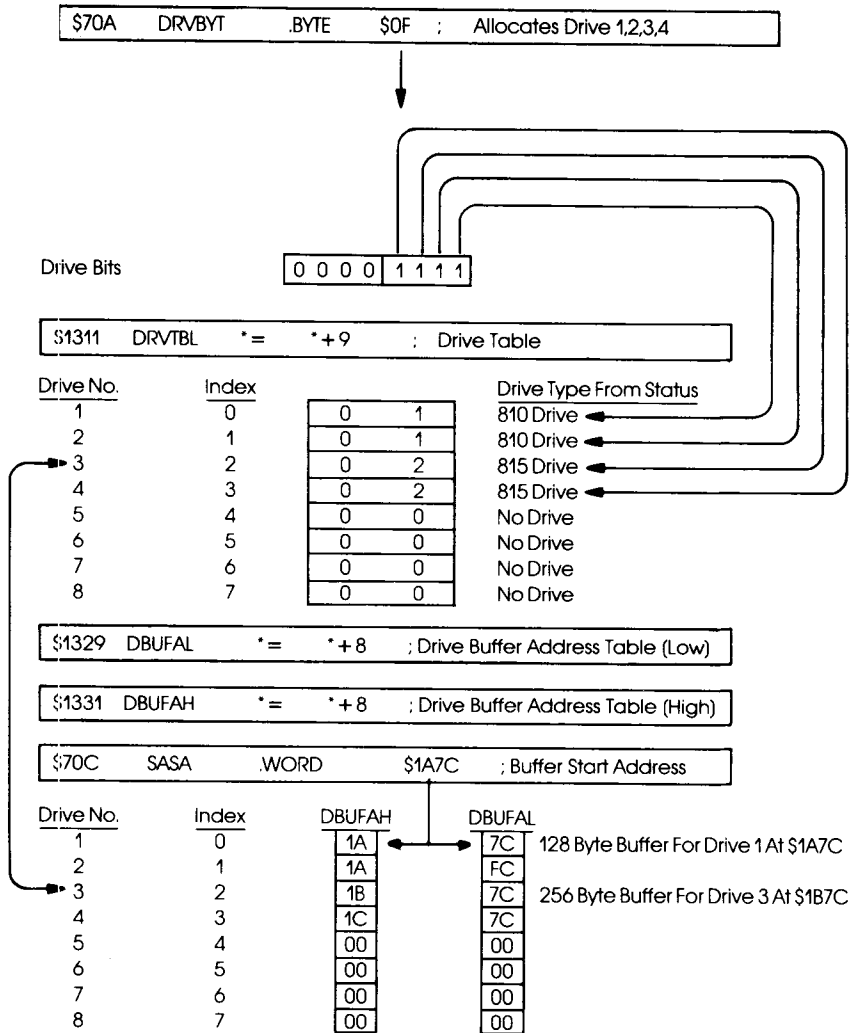


Figure 4-1
Drive Tables

CHAPTER FOUR

\$1319 SECTBL * = * + 16 ; Sector Allocation Table

Buffer Number		
0	00	Buffer Is Available
1	00	
2	00	
3	00	
4	00	
5	00	
6	00	Buffer Is Available
7	FF	Buffer NOT Available
8	FE	
9	FD	
10	FC	
11	FB	
12	FA	
13	F9	
14	F8	
15	F7	
16	F6	Buffer NOT Available

\$709 SABYTE .BYTE 7 ; Number Of 128 Byte Sector Buffers

\$1339 SABUFL * = * + 16 ; Sector Buffer Address (Low) Table

\$1349 SABUFH * = * + 16 ; Sector Buffer Address (High) Table

(Last Drive Buffer Address + Drive Type (1 or 2) * 128)

Buffer Number	SABUFL	SABUFH	
0	1C	8C	
1	1D	0C	(Previous Entry + 128)
2	1D	8C	
3	1E	0C	
4	1E	8C	Sector Buffer 4 Address = \$1E8C
5	1F	0C	
6	1F	8C	
7	20	0C	
8	00	00	
9	00	00	
10	00	00	
11	00	00	
12	00	00	
13	00	00	
14	00	00	
15	00	00	

Figure 4-2
Sector Allocation Tables

Chapter Five

FMS ENTRY

The Device Vector Table for FMS is located at DFMSDH (\$7CB). The address of this table is placed in the Device Handler Table by the FMS Initialization routine. When CIO needs to call an FMS function (Figure 1, control path 2), it will locate the address of the function via the table at DFMSDH. This table is the standard Atari Device Handler Vector Table. The six entries are for:

- Open
- Close
- Get Byte
- Put Byte
- Status
- Device Dependent (XIO) Commands

Each of the six FMS entry points starts with a subroutine call to the FMS SETUP routine. SETUP (\$1164) prepares FMS parameters to deal with the particular task to be performed.

SETUP

Address – \$1164

Entry Registers – A = Possible 'Put Data' data byte.
X = IOCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = IOCB number times 16.
Y = Sector Buffer Index.

Functions:

- 1) Initialize ERRNO to \$9F. This value will be used in the FMS exit routines to form a FMS error number in the event of error.
- 2) Save the X Register in CURFCB. This value will be used as an index to the proper IOCB and the proper FCB for the current operation.
- 3) Save the value of the stack register as it was upon entry to

FMS. This value will be used in the FMS exit routine.

- 4) Set up drive information values from the drive number contained in the zero page IOCB field ICDNOZ.
- 5) Allocate a sector buffer to the FCB if one is not already allocated.

Chapter Six

FMS EXIT

There are two types of FMS exits: the normal exit and the error exits. Both of these exit types end up calling the RETURN routine.

RETURN

Address – \$12D3

Entry Registers – A = Return Code.
X = Don't Care.
Y = Don't Care.

Exit Registers – A = Possible 'Get Byte' data byte.
X = IOCB number times 16.
Y = Return Code.

Functions:

- 1) The X register is loaded with the current IOCB number times 16 from CURFCB.
- 2) The return code is placed in the IOCB status field (ICSTA).
- 3) The stack register is restored to point to the stack displacement at FMS entry from the value saved in ENTSTK.
- 4) The possible "Get Data" data byte is loaded into the A register.
- 5) The Y register is loaded with the return code.

- 6) The caller (CIO) is returned to via the RTS instruction.

GREAT And FGREAT

GREAT and FGREAT are the exit points used by FMS when the operation has terminated normally. FGREAT is located at \$12EA and is used to free the sector buffer that has been allocated to the FCB. The FRESBUF routine is used to free the buffer. FGREAT exits directly to GREAT (\$12F0). The GREAT exit point loads the normal return code (\$01) into the A register and goes to RETURN.

Error Exits

The ERREOF exit is called when an end of file condition is found. ERREOF loads the end-of-file condition code (\$88) in the A register and goes to RETURN.

The ERRIO exit is called if an error occurs during an I/O operation (Figure 1, control flow 3). The error code from the DCB (control path K) is loaded into the A register as the FMS return code and control is passed to RETURN.

All other errors exits are at the ERxxx labels starting at \$12B5. The error code is developed by means of a series of 6502 INC instructions which increment the ERRNO (which was initialized to \$9F at FMS entry). The final instruction at the end of the INC chain loads the final ERRNO value into the A register and control is passed directly to RETURN.

Chapter Seven

DEVICE DEPENDENT COMMANDS

A Device Dependent Command is any command which is not Open, Close, Get Byte, Put Byte, or Status. When the command value in the IOCB is greater than 15 (\$0F), CIO will call the Device Handler Device Dependent Command routine. The Device Handler must determine if the command is a valid command for that device. The Device Dependent Commands that for FMS are:

- Rename
- Delete
- Lock
- Unlock
- Point
- Note
- Format

The FMS Device Dependent Command routine starts at DFMDDC.

DFMDDC

Address – \$BA7

Entry Registers – A = Don't Care.
X = IOCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = IOCB and FCB number times 16.
Y = Unknown.

Function:

- 1) Call SETUP
- 2) If the command is Format (254), then go to the Format routine, XFORMAT at \$D18.
- 3) If the command is not Format, then check that the command

CHAPTER SEVEN

value is \$20 through \$26. If the command value is not in this range then exit via the ERDVDC (Command Error) routine.

4) If the command is valid, go to the command via the DCDCVT vector table.

XFORMAT

The XFORMAT routine executes the FORMAT Device Dependent Command.

Address – \$D18

Entry Registers – A = Don't Care.

X = IOCB and FCB number times 16.

Y = Don't Care.

Exit Registers – A = Unknown.

X = Unknown.

Y = Unknown.

Functions:

1) Issue the format I/O command to the drive. This will cause the drive to perform the physical formatting of the disk. If the command returns with good status and there were no bad sectors reported, then continue with the logical format operations. In the event of physical format errors, exit via the ERDBAD error exit.

2) Clear the drive buffer to zero.

3) Set the sector count values into the DVDMSN (VTOC displacement one) and the DVDNSA (VTOC displacement three) fields.

4) Set all 90 sector bit map bits to one (available).

5) Deallocate the first four sectors for the boot sectors.

6) Deallocate the middle nine sectors for the VTOC and the Directory.

7) Write the VTOC to the Disk.

8) Clear the eight directory sectors to zero.

9) Exit via the FGREAT exit.

XDELETE

The XDELETE routine executes the DELETE Device Dependent Command.

Address – \$C32

Entry Registers – A = Don't Care.

X = IOCB and FCB number times 16.
Y = Don't Care.

Exit Parameters – A = Unknown.
X = Unknown.
Y = Unknown.

Functions:

- 1) The filename is decoded via the FNDCODE routine.
- 2) The first filename is searched for via the SFDIR routine.
- 3) The file, if found, is deleted via the XDELO routine.
- 4) If the file just deleted was DOS.SYS then the boot record is re-written via the DELDOS routine.
- 5) The directory is searched for the next matching entry. If an entry is found then the process repeats at step three. If no further matching directory entries are found, then exit via FGREAT.

XDELO

The XDELO routine is used to delete the file whose directory entry is indicated by the CDIRD (current Directory Displacement) byte (\$1305).

Address – \$C53

Entry Registers – A = Don't Care.
X = IOCB and FCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = Unknown.
Y = Unknown.

Functions:

- 1) The OPVTOC routine is called to insure that the disk is not write protected.
- 2) The TSTLOCK routine is called to insure that the file is not locked.
- 3) The file deleted bit is set in the directory entry flag and the directory sector is written back to the disk.
- 4) The VTOC sector bit map bits for the sectors in the file are set to one to make them eligible for reuse. This process is achieved by reading each sector in the file sector chain and calling the FRESECT routine to change the VTOC bit map.
- 5) The VTOC Write Required Bit is set so that the VTOC will be written back to the disk.

CHAPTER SEVEN

XRENAME

The XRENAME routine executes the RENAME Device Dependent Command.

Address – \$BD9

Entry Registers – A = Don't Care.
X = IOCB and FCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = Unknown.
Y = Unknown.

Functions:

- 1) The filename is decoded via the FNDCODE routine.
- 2) The directory is searched for the first entry to be renamed. If no entry is found then the ERFNF (File not found) exit is taken.
- 3) The TSTLOCK routine is called to insure that the file is not locked.
- 4) If TSTDOS determines that the old filename is DOS.SYS then the boot record is rewritten via the DELDOS routine.
- 5) If new filename is DOS.SYS, then the boot record is rewritten via the SETDOS routine.
- 6) The filename in the directory is changed to the new filename.
- 7) The directory sector is rewritten.
- 8) The directory is searched for the next filename match. If a match is found, then the process repeats at step three. If no further match is found then, exit via FGREAT.

XLOCK And XUNLOCK

The XLOCK routine executes the LOCK Device Dependent Command. The XUNLOCK routine executes the UNLOCK Device Dependent Command.

Address – \$C7C

Entry Registers – A = Don't Care.
X = IOCB and FCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = Unknown.
Y = Unknown.

Functions:

- 1) The XLOCK entry sets the LOCK bit value, DFDLOC (\$20), into TEMP4. The XUNLOCK entry sets a zero value into TEMP4. Both routines then go to XLCOM.
- 2) The filename is decoded via the FNDCODE routine.
- 3) The directory is searched for the first file entry match. If no match is found, the ERFNF (file not found) exit is taken.
- 4) The files directory flag is modified to either LOCKED or UNLOCKED by means of the value previously set into TEMP4.
- 5) The Directory sector is written back to the disk.
- 6) The CSFDIR routine is called to find the next filename match. If a match is found, then the process repeats at step four. If no match is found, then exit via FGREAT.

XPOINT

The XPOINT routine executes the POINT Device Dependent Command.

Address – \$CBA

Entry Registers – A = Don't Care.

X = IOCB and FCB number times 16.

Y = Don't Care.

Exit Registers – A = Unknown.

X = Unknown.

Y = Unknown.

Functions:

- 1) If the FCBFLG indicates that the file can acquire sectors (Opened for Output or Append), then exit via the ERRPOT (point error) exit.
- 2) If the current sector is not the same as the sector POINTed to by the IOCB AUX3 and AUX4 fields, then write the current sector back to the disk (if it has been changed).
- 3) Read the POINTed to sector into the sector buffer.
- 4) Set the FCB next byte pointer, FCBDLN, to the value indicated by the user Point data in the IOCB AUX5 field.
- 5) Exit to FGREAT.

XNOTE

The XNOTE routine executes the NOTE Device Dependent Command.

CHAPTER SEVEN

Address – \$D03

Entry Registers – A = Don't Care.

X = IOCB and FCB number times 16.

Y = Don't Care.

Exit Registers – A = Unknown.

X = Unknown.

Y = Unknown.

Functions:

- 1) The current sector number and data displacement into the sector is moved to the appropriate IOCB fields, ICAUX3, ICAUX4, ICAUX5.
- 2) Exit via GREAT.

Chapter Eight

FMS OPEN ROUTINES

The FMS Open routine, DFMOPN, is called directly by CIO via the FMS Device Vector Table, DFMSDH at \$7CB.

DFMOPN

The DFMOPN routine is the FMS file open routine.

Address – \$8AB

Entry Registers – A = Don't Care.
X = IOCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = Unknown.
Y = Unknown.

Functions:

- 1) Initialize for this operation by calling SETUP.
- 2) Decode the filename via FNDCODE.
- 3) Examine the open code in ICAUX1 for the open-for-directory-read command. If this is a directory read command, go to LISTDIR.
- 4) If not a directory read, then search the directory for the first match on the file name and save the resulting search condition on the stack.
- 5) Determine the exact type of Open operation to be performed by examining the IOCB ACUX1 field. If INPUT, go to DFOIN. If Output, go to DFOUT. If Update, go to DFOUPD. If Append, go to DFOAPN. If none of the above, exit via the ERDVDC (device command error) exit.

CHAPTER EIGHT

DFOIN

DFOIN (\$8D8) is entered when opening a file for Input. The routine pops the stack to determine if the directory search for the file name was successful. If the file name was found in the directory, then go to DFOUI. If the search was not successful, then exit to ERFNF (file not found).

DFOUPD

DFOUPD (\$8DD) is entered when opening a file for Update (Input and Output). The routine pops the stack to determine if the file name was found in the directory. If the file was not found, then exit to ERFNF (file not found). If the file was found, insure that the file is not Locked by calling TSTLOCK. If the file is unlocked, then continue at DFOUI.

DFOUI

DFOUI (\$8E3) is entered to finish opening a file for Input or Update. The read setup routine, DFRDSU, is called. FMS then exits via the GREAT exit.

DFDRDSU

DFDRDSU (\$9AE) is entered to set up a data file for reading. It begins by calling SETFCB to set some standard file information into the FCB. It continues by setting up the FCB with various other parameters to read the first data sector in the file. This sector is read via the RDNS0 routine. When the sector has been read into the sector buffer, the code returns to the caller.

DFOAPN

DFOAPN (\$BEC) is entered to open a file for Append.

- 1) Pop the stack to determine if the file has been found in the directory. If the file was not found exit via ERFNF.
- 2) If the file was created by DOS 1, then exit via ERAPO.
- 3) Insure the file is not locked by calling TSTLOCK.
- 4) Insure the diskette is not write protected by calling OPVTOC.
- 5) Allocate a new sector for the start of the Append chain by calling GETSECTOR.
- 6) Save the sector number of the sector obtained in FCBSSN so that it will be available when the file is closed.

- 7) Continue opening the file as if it were being opened for Output at DHFOX2.

DFOOUT

The DFOOUT (\$911) routine is entered when opening a file for Output.

- 1) Pop the stack to determine if the file was found in the directory.
- 2) If the file was found, then delete it via the XDEL0 (\$C53) routine.
- 3) If the file was not found, then make a new entry in the directory via the code at DFOX1 (\$91D).
- 4) Allocate a data sector for the file via the GETSECTOR routine.
- 5) Put the necessary information about the file into the directory and write the directory sector back to the disk.
- 6) Continue at DHFOX2.

DHFOX2

DHFOX2 (\$97C) is entered to finish the Open process for files that are being opened for Output or Append.

- 1) Finish initializing the FCB via SETFCB.
- 2) If the TSTDOS routine determines that the file name being opened is DOS.SYS, then write out DOS via the WRTDOS routine.
- 3) Exit via GREAT.

SETFCB

The SETFCB (\$995) routine is used in the various Open file routines to place certain common data into the FCB.

Chapter Nine

FMS CLOSE ROUTINES

The FMS close routine is called directly by CIO via the FMS Device Vector Table, DFMSDH at \$7CB.

DFMCLS

Address – \$B15

Entry Registers – A = Don't Care.
X = IOCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = Unknown.
Y = Unknown.

Functions:

- 1) Initialize via call to SETUP.
- 2) If the file was not opened for some form of output (Output, Update or Append) then clear the FCB open flag, FCBOTC and exit via FGREAT.
- 3) If the FCBFLG indicates that the file has not acquired sectors, then continue at CLUPDT to close the Update file.
- 4) Write the last data sector via WRTLSEC.
- 5) Read the file's directory sector into the directory buffer via the RRDIR routine.
- 6) Get the sector count from the directory.
- 7) If the file was opened for Output (i.e. it is not open for Append), then continue at CLOUT.
- 8) Read all the data sector of the file until the end-of-file sector is found.
- 9) Place the sector address of the start of the Append chain into the link sector field of the (old) end-of-file sector.
- 10) Continue at CLOUT.

CLOUT

The CLOUT (\$B50) routine is entered to finish closing a file that had been opened for Output or Append.

- 1) The sector count field of the directory is updated.
- 2) The open for output flag is turned off.
- 3) The file in use flag is set.
- 4) The directory sector is written back to the disk by the DRTDIR routine.
- 5) The VTOC sector is written back to the disk by the WRTVTOC routine.
- 6) The FCB open code flag, FCBOTC, is cleared to zero.
- 7) Exit via FGREAT.

CLUPDT

The CLUPDT (\$B75) is called to finish the closing of a file that had been opened for Update.

- 1) If the current sector in the sector buffer has been modified then write it back to the disk via the WRCSIO routine.
- 2) Clear the FCB open flag, FCBOTC, to zero.
- 3) Exit via FGREAT.

Chapter Ten

GET BYTE ROUTINE

The FMS GET BYTE routine, DFMGET, is called directly by CIO via the FMS Device Vector Table, DFMSDH at \$7CB. The GET BYTE routine's function is to get and return the next sequential data byte to CIO.

DFMGET

Address – \$ABF

Entry Registers – A = Don't Care.
 Y = IOCB number times 16.
 X = Don't Care.

Exit Registers – A = Unknown.
 Y = Unknown.
 X = Unknown.

Functions:

- 1) Initialize via the SETUP routine.
- 2) If the FCB is opened for Directory read, then go to GDCHAR.
- 3) If the current sector is empty, attempt burst I/O (see Burst I/O section), then continue with number four.
- 4) Read the next sector via the RDNXTS routine. If the read sector operation did not return an end-of-file condition, then continue at step three, else exit via ERREOF (end-of-file error).
- 5) Get the data byte from the sector and place it in SVDBYT for the exit routines.
- 6) If the next byte in the file is the end-of-file byte, exit via RETURN with the impending end-of-file condition code (\$03), else exit via GREAT.

Chapter Eleven

PUT BYTE ROUTINE

The FMS PUT BYTE routine, DFMPUT, is called directly by CIO via the FMS Device Vector Table, DFMSDH at \$7CB. The PUT BYTE routine's function is to place the single data byte transmitted by CIO into the data sector.

DFMPUT

Address – \$99C

Entry Registers – A = The “put data” data byte.
X = The IOCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = Unknown.
Y = Unknown.

Functions:

- 1) The data byte in the A register is saved in SVDBYT.
- 2) SETUP is called to initialize for this operation.
- 3) If the caller was not CIO, then prevent a burst I/O operation from occurring.
- 4) If the file was not opened for output, then exit via ERDVDC (device command error).
- 5) If the current data sector is full, write the sector via WRTNXS, then attempt burst I/O (see BURST I/O section). If a burst I/O operation did take place, then get the next byte after the area just written by burst I/O and place it into the SVDBYT cell.
- 6) Increment the sector data byte count.
- 7) Move the data byte from SVDBYT to the next available data byte in the sector.
- 8) Set the sector modified flag in the FCB.
- 9) Exit via GREAT.

Chapter Twelve

BURST I/O

The CIO is designed to fill or empty a large user buffer with data bytes sent to or received from a device handler, a byte at a time. To fill a thousand-byte buffer, CIO would have to call FMS one thousand times in rapid succession. While the process is simple and easy to implement by both CIO and the Device Handlers, it can be very slow. This is particularly true in the case of FMS which has a great deal of overhead code to go through each time it is called. FMS circumvents most of the CIO/FMS calls for large data transfers via the BURST I/O routines.

Burst I/O operates by reading or writing data sectors directly into the user buffer (Figure 1, data path I). There are a number of tests that must be passed before a burst I/O operation can take place. If any of the tests fail, then the CIO/FMS data transfer reverts to the normal mode of operation.

When the PUT BYTE routine is called, it will call the WTBUR (\$A1F) routine when it is ready to start filling a new data sector. WTBUR will not allow a burst I/O operation to happen if the file has been opened for Update. If the file has not been opened for Update, then WTBUR goes to the common read/write burst I/O test routine, TBURST at \$A28. If the file has been opened for Update, then exit Burs: I/O indicating that a Burst I/O did not happen. When WTBUR calls TBURST, it has the A register set to non-zero to indicate that it is write.

When the GET BYTE routine is called, it will call the RTBUR (\$A26) routine when it is ready to read a new data sector. RTBUR indicates that it is read by setting the A register to zero and then enters TBURST.

TBURST

- 1) Save the A register in BURTYP. This value will indicate if the burst operation is a read or a write.
- 2) If the I/O command in the IOCB is for text I/O (a transfer that is to end when the Atari end-of-line (\$9B) character is transferred), then TBURST will exit indicating (carry set) that a burst I/O operation did not occur.

- 3) If the user buffer length in the IOCB is not at least a full sector in size, then exit without doing a burst I/O.
- 4) If all the above tests pass, then perform a burst I/O operation. The first step in the burst I/O operation is to change the zero page sector buffer pointer, ZXBA (\$47) from the FMS sector buffer address to the user buffer address.
- 5) If the operation is read, then read the next sector via RDNXTS. If the read sector operation produced an end-of-file, then go to BUREOF, else go to BBINC.
- 6) If the operation is write, then the area in the user buffer, where the three bytes of data sector control information is to be placed, will be saved. The data will be written via the WRTNXS routine. The saved user data will then be copied back into the user buffer. The code then continues at BBINC.

BBINC

The BBINC routine is entered after a single burst I/O sector has been read or written. BBINC updates data counters in the FCB and in the IOCB and tests for the end of the Burst I/O.

- 1) The zero page sector buffer pointer is incremented by the length of data in a sector (125 or 253).
- 2) The user buffer length is decremented by the length of data in a sector.
- 3) The TBLEN routine is called to determine if there is enough room left in the user buffer to read or write another full sector (128 or 256 bytes). If another sector can be read or written, then the process repeats at NXTBUR (\$A3E).
- 4) If there is not enough room in the user buffer to perform another full sector read or write, then BUREOF is entered.

BUREOF

- 1) The final address in the zero page sector pointer, ZSBA (\$47), is moved to the IOCB buffer address field.
- 2) The value in the zero page sector buffer pointer is restored by the SSBA routine.
- 3) The caller is returned to with the carry cleared to indicate that a burst I/O operation has happened.

Chapter Thirteen

READING THE DIRECTORY AS A FILE

A formatted subset of the data in the Directory can be read as if the Directory were a disk file. This is accomplished by using the open directory code (\$02) in the IOCB ICAUX1 byte. When FMS recognizes this code in the Open routine (at \$8B1), it will go directly to the LISTDIR routine. The LISTDIR routine prepares the FCB for reading the directory as a file. The GET BYTE routine will recognize the read directory condition from information stored in the FCBOTC field (see \$AC2) and go directly to the directory read character I/O routine GDCHAR.

LISTDIR

Address – \$DAD

Entry Registers – A = Don't Care.

X = IOCB and FCB number times 16.

Y = Don't Care.

Exit Registers – A = Unknown.

X = Unknown.

Y = Unknown.

Functions:

- 1) The TEMP4 byte is used to count the characters that have been transmitted by GDBYTE from the formatted line buffer. LISTDIR sets this value to zero to indicate the start of a new formatted line.
- 2) The SFDIR routine is called to start a wild card search for the file name in the directory.
- 3) If a match is found then FDENT is called to format the entry and prepare for the GDBYTE calls. Exit via GREAT.
- 4) If a match is not found, then LDCNT is called to prepare to send the xxx FREE SECTORS line.

GDCHAR

GDCHAR (\$DB9) is entered from GET BYTE to get a single data byte from a formatted directory line.

- 1) The TEMP4 flag is tested. If the value is negative, then all formatted information has been transmitted. Exit is via the ERREOF (end-of-file error) exit.
- 2) The value in TEMP4 is used as an index into the formatted line buffer to get the next character. The character is placed into SVDBYT for loading into the A register by the RETURN routine.
- 3) The character retrieved from the buffer is examined for the EOL (\$9B) character.
- 4) If the character is not an EOL, then exit is via GREAT.
- 5) If the character was an EOL, then the line length is examined to see if the line was a directory entry line (i.e., if the length was 17) or the final xxx FREE SECTORS.
- 6) If the line was the final line, then TEMP4 is set to a negative value (\$80) to indicate that all formatted lines have been sent. Exit is via GREAT.
- 7) If the line was not the final line, then CSFDIR is called to find the next matching file name.
- 8) If a file name match is found, then FDENT is called to format the found entry into the formatted line buffer. Exit is via GREAT.
- 9) If a file name match is not found, then go to LDCNT to format the final line.

LDCNT

LDCNT (\$DE9) formats the final line of a directory read.

- 1) Read the VTOC.
- 2) Get the free sector count from the VTOC and convert it to ATASCII via the CVDX routine.
- 3) Move the FREE SECTORS message to the formatted line buffer.
- 4) EXIT is via FGREAT.

FDENT

The FDENT (\$E21) routine formats the current directory entry into the formatted line buffer for subsequent reading by GDBYTE.

- 1) The directory flag is checked for the file locked condition. If

-
- the file is locked, then the “*” is placed in the formatted line.
- 2) The file name is moved from the directory entry to the formatted line.
 - 3) The file sector count is converted to ATASCII and placed in the formatted line.
 - 4) The EOL character is placed in the formatted line.
 - 5) Exit is via the RTS instruction.

Chapter Fourteen

SECTOR I/O ROUTINES

The FMS performs sector I/O by calling the SIO routine in the OS ROM (Figure 1, control path 3). All sector I/O calls in the FMS occur from the BSIO routine. There are several other routines that are designed to set up information for BSIO. These routines deal with reading and writing sectors of a particular type such as data sectors, directory sectors, and the VTOC sector.

BSIO

Address – \$76C

Entry Registers – A = Sector number most significant byte.
Y = Sector number least significant byte.
X = If 1, then 128 byte I/O (810 drive).
If 2, then 256 byte I/O (815 drive).

Exit Registers – A = Status byte from DCB.
Y = Unknown.
X = IOCB and FCB number times 16.

Functions:

- 1) The sector number is stored in the DCB from the A, Y register pair. The DCB is the interface control block for SIO calls.
- 2) If the carry is clear, then the DCB is set up for read data. If the carry is set, then the DCB is set up for write data.
- 3) The serial bus ID for the disk, and the disk timeout values are placed into the DCB.
- 4) The error retry counter, RETRY, is set for four retries.
- 5) The I/O data length is set to 128 or 256 depending upon the data in the X register.
- 6) The serial I/O routine (\$E459) is called to execute the I/O.
- 7) If the I/O operation was good, then the X register is loaded with the IOCB (and FCB) number times 16 from the CRFCB cell and the status byte from the DCB is loaded into the A register. Return is via the RTS instruction.
- 8) If the I/O operation was bad, then the retry counter is decremented. If the retry value is positive, then the I/O is retried. If the value is negative, then the routine is exited in the manner described in step seven.

DSIO

The DSIO routine is called to perform data sector I/O operations.

Address – \$11F7

Entry Registers – A = Sector number most significant byte.
Y = Sector number least significant byte.
X = IOCB and FCB number times 16.

Exit Registers – A = I/O condition code.
Y = Unknown.
X = IOCB and FCB number times 16.

Functions:

- 1) The sector buffer address is obtained from the zero page sector buffer pointer ZSBA (\$47) and placed in the DCB buffer address field, DCBBUF.
- 2) The drive type byte is loaded into the X register from DRVTYPE. If the drive is an 810, then the value will be one. If the drive is an 815, then the value will be two.
- 3) BSIO is called.
- 4) The DSIO caller is returned to via the RTS instruction.

CHAPTER FOURTEEN

RDDIR And WRTDIR

The RDDIR and the WRTDIR routines are used to perform Directory sector I/O operations. The RDDIR entry (\$106E) sets the carry to indicate read. The WRTDIR entry (\$1071) clears the carry to indicate write. Both of the routines continue at DIRIO.

DIRIO

- 1) Save the read/write flag (carry sense) on the stack.
- 2) Set the address of the directory buffer into the DCB buffer field, DCBBUF.
- 3) The CDIRS cell contains the number of the directory sector to be read or written. This value ranges from zero to seven. The DIRIO routine creates the actual sector number to read or write by adding \$169 to the CDIRS value. The resulting sector number is placed in the A,Y register combination.
- 4) Continue at DSYSIO.

RDVTOC And WRTVTOC

The RDVTOC and WRTVTOC routine are called to initiate I/O to and from the VTOC sector. The RDVTOC routine (\$108B) first checks the write required byte in the VTOC sector buffer. If the value of this byte is not zero, then the VTOC is already in the buffer (and has been changed). If the VTOC is already in the buffer, then the read does not have to be done; therefore, the RDVTOC routine will return to the caller. If the write-required byte is zero, then RDVTOC will clear the carry to indicate that the operation is read. The WRTVTOC routine (\$1095) sets the write required byte to zero, then sets the carry to indicate a write operation. Both RDVTOC and WRTVTOC continue at VTIO.

VTIO

- 1) The read/write flag is pushed onto the stack.
- 2) The VTOC sector buffer address is moved from the zero page drive buffer address pointer ZDRVA (\$45) to the DCB buffer pointer, DCBBUF.
- 3) The A,Y register combination is loaded with the VTOC sector number (\$168).
- 4) Continue at DSYSIO.

DSYSIO

- 1) The read/write sense is popped from the stack.

- 2) The drive type value is loaded into the X register from DRVTYP.
- 3) BSIO is called.
- 4) If the I/O operation was good, then return to the caller via the RTS instruction.
- 5) If the I/O operation was bad, the exit via the ERRSYS exit (fatal system I/O error).

OPVTOC

The OPVTOC routine (\$10BF) is used by various FMS routines to insure that the diskette is not write protected before executing functions that will write to the disk. This routine will read the VTOC via RDVTOC and then attempt to write the VTOC via WRTVTOC. If the diskette is write protected, the WRTVTOC will cause an I/O error exit (error number 144). If the diskette is not write protected, then the routine will return to the caller. When OPVTOC does return to the caller, the current disk VTOC is in the drive buffer.

Chapter Fifteen

FILE NAME DECODE ROUTINE

The FNDCODE routine is used to transform the user supplied file name into a form that is usable in FMS for wild card searching of the directory. The primary and extension parts of the user file name are padded with blanks and question marks as required. The following examples show the types of transform performed by FNDCODE:

<u>User File Name</u>	<u>Transformed File Name</u>
D:*. *	??????????
D1:GLOP.*	GLOP ???
D1:GLOP.BAS	GLOP BAS
D2:*.ASM	?????????ASM
D:GL?P.S*	GL?P S??
D1:G*	G???????

FNDCODE

Address – \$E9E

Entry Registers – A = Don't Care.
X = IOCB and FCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = IOCB and FCB number times 16.
Y = Unknown.

Functions:

- 1) The user file name buffer is searched for the colon (:) delimiter. If the delimiter is not found within 256 characters then exit to ERRFN routine (file name error).
- 2) The FMS file name buffer, FNAME, is cleared to blanks.
- 3) The EXT SW byte is set to zero. When EXT SW is zero, the primary file name field is being processed. When EXT SW is

- minus, then the extension file name field is being processed.
- 4) The next character in the user file name buffer is examined.
 - 5) If the character is an *asterisk* (*), then the field is padded with question mark characters to the end of the field.
 - 6) If the character is a period and the extension field is being processed, then exit via the RTS instruction.
 - 7) If the character is a period and the primary field is being processed, then switch to the extension field processing.
 - 8) If the character is a question mark, then put it into the FNAME via FDSCHAR.
 - 9) If the character is alphanumeric (A through Z, or 0 through 9), then put it into FNAME via FDSCHAR.
 - 10) If the character is none of the above, then assume that end of the filename has been found and exit via the RTS instruction.
 - 11) If a character was stored, then continue at step four.

FDSCHAR

- 1) If the character counter register, X, indicates that the primary field is full, then exit without storing the character.
- 2) If the character counter register, X, indicates that the extension field name is full, then exit without storing the character.
- 3) Store the character into FNAME indexed by the X register.
- 4) Increment the X register.
- 5) Return to caller via the RTS instruction.

Chapter Sixteen

DIRECTORY SEARCHING

The Directory search routine searches the directory entries for a file name that matches the name in FNAME. The routine has two entry points: SFDIR which is used to begin the search at the start of the directory, and CSFDIR, which is used to continue searching the directory at the entry just past the previously found matching entry.

The routines have five memory cells that they use for controlling the search operation: DHOLES, DHOLED, CDIRS, CDIRD and SFNUM. The CDIRS cell contains the current relative directory sector number (zero through seven). The CDIRD cell contains the displacement into the directory sector of the current entry. DHOLES gives the relative directory sector number (zero through seven) of the first hole or available entry in the directory. The DHOLED cell gives the displacement to the first available entry that is the hole. The SFNUM cell is used to contain the current file number of the entry being examined. The value in SFNUM will be from zero through 63.

If the value of DHOLES is \$FF at the end of the search, then the directory is full.

The directory search routine will exit with the carry clear if a match was found. It will exit with the carry set if no matching entry was found.

SFDIR

The SFDIR routine (\$F21) is called to start searching the directory at the start of the directory.

- 1) Initialize DHOLES, CDIRS, SFNUM to \$FF.
- 2) Initialize CDIRD to \$70.
- 3) Continue at CSFDIR.

CSFDIR

The CSFDIR routine (\$F31) is called to continue searching the directory.

- 1) Increment the file number, SFNUM.

- 2) Increment CDIRD by the size of a directory entry (16).
- 3) If the CDIRD is now greater than, or equal to, 128 (\$80) then increment CDIRS by one. If the value of CDIRD is now eight, then exit with the carry set to indicate that a match was not found. If CDIRD is less than eight, then read the next directory sector via RDDIR. Set CDIRD to zero.
- 4) If the directory entry flag field is zero then the end of the used portion of the directory has been reached. If a hole has not been found, then mark this entry as a hole. Exit with the carry set to indicate that the file was not found.
- 5) If the directory entry flag field indicates that the file is open for output, then skip this entry.
- 6) If the directory entry flag field indicates that the file has been deleted, and a hole has not been found, then mark this entry as a hole and continue searching the directory.
- 7) If the file is in use, then check the file name in the directory entry for a match with the name in FNAME. Wild card characters in FNAME (question marks) are assumed to match the corresponding characters in the directory entry file name.
- 8) If the names match, then exit with the carry clear to indicate that a match was found.
- 9) If a match was not found, then continue to search the directory.

Chapter Seventeen

WRITE NEXT SECTOR

The write next sector routine, WRTNXS, is used to write a data sector to disk.

Address – \$F94

Entry Registers – A = Don't Care.

X = IOCB and FCB number times 16.

Y = Don't Care.

Exit Registers – A = Unknown.

X = IOCB and FCB number times 16.

Y = Unknown.

Functions:

- 1) If the file has been opened for update, and the sector has not been modified, then do not write the sector. Read the next data sector and then return to caller.
- 2) If the file has been opened for update, and the sector has been modified, then write the current sector. Read the next data sector into the sector buffer and return to the caller.
- 3) If the file is not opened for update, then allocate a new sector to the file by calling GETSECTOR.
- 4) Move the sector byte count from the FCB FCBDLN field to the data sector byte count field.
- 5) Move the address of the newly acquired sector from the FCB FCBLSN field into the link field of the current data sector.
- 6) Write the current sector to the disk via WRCSIO.
- 7) If the I/O was bad, mark the FCB by placing a zero value into FCBOTC as closed and exit via RETURN with the I/O error number as the return code.
- 8) If the I/O was good, then increment the FCB sector counter field, FCBCNT.

-
- 9) Call MVLSN to move the sector number of the link sector number field of the FCB, FCBSLN, to the current sector number field of the FCB, FCBCSN.
 - 10) Set the current data length field of the FCB, FCBDLN, to zero.
 - 11) Set the maximum data length field of the FCB, FCBMLN, to 125 (if 810 drive) or 253 (if 815 drive).
 - 12) Return to user via the RTS instruction.

Chapter Eighteen

READ NEXT SECTOR

The read next sector routine, RDNXTS, reads the next sector in the file sector chain into the sector buffer. If there are no more sectors in the chain, then the routine returns with the carry set to indicate end-of-file. If the routine returns with the carry clear, then the next sector has been read.

RDNXTS

Address – \$100F

Entry Registers – A = Don't Care.
X = IOCB and FCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = IOCB and FCB number times 16.
Y = Unknown.

Functions:

- 1) If the file has been opened for Update, then WRTNXS is

CHAPTER EIGHTEEN

- called to write the current sector if it has been modified.
- 2) If the FCB link sector number field, FBCLSN, is zero then there are no further sectors to read. Return to the caller with the carry set to indicate that the end-of-file has been reached.
 - 3) Call MVLSN to move the FCB link sector number field, FCBLSN, the FCB current sector number field, FCBCSN.
 - 4) Call RWCSIO with the carry set to read the next sector.
 - 5) If the I/O operation was bad, exit via the ERRIO exit (I/O error).
 - 6) Insure that the file number in the sector just read agrees with the file number in the FCB. If the file numbers are not the same, exit via the ERFNMM exit (file number mismatch). Note: if the routine was called by delete, return to delete indicating end-of-file.
 - 7) Move the link sector number from the data sector to the FCB link sector field in the FCB, FCBLSN.
 - 8) Move the sector data length information from the data sector to the FCB maximum data length field, FCBSLN.
 - 9) Reset the FCB data length field, FDBDLN, to zero.
 - 10) Return to the caller with the carry clear to indicate that a sector has been read.

Chapter Nineteen

GET AND FREE SECTOR ROUTINES

The get sector routine, GETSECTOR, is called when a new sector is needed. The routine searches the bit map in the VTOC for a free sector. The sector found is deallocated from the bit map and the sector number is returned to the caller. The free sector routine, FRESECT, is given a sector number to be freed. FRESECT locates the required bit map bit in the VTOC and turns it on (sets it to one). The sector is now eligible for reuse.

GETSECTOR

Address – \$1106

Entry Registers – A = Don't Care.
X = IOCB and FCB number times 16.
Y = Don't Care.

Exit Registers – A = Unknown.
X = IOCB and FCB number times 16.
Y = Unknown.

Functions:

- 1) The Y register is used as an index into the bit map bytes.
- 2) The bit bytes are examined sequentially from the first bit map byte to the last bit map byte until a non-zero byte is found. The displacement to this byte is saved in TEMP1.
- 3) If no bits are found in the bit map, then the ERRNSA exit (no sectors available) is taken.
- 4) The number-of-sectors-available-field, in the VTOC, is decremented by one.
- 5) The VTOC write required byte in the VTOC is set to a non-

CHAPTER NINETEEN

zero value to indicate that the VTOC has been changed and must be written back to the disk.

- 6) The non-zero bit map byte that was found in the bit map search is retrieved. The bits in this byte are shifted left until a bit moves into the carry flag. The carry is then set clear and the bits shifted back to their original position. The byte with the newly allocated sector bit turned off is placed back into the bit map.
- 7) The number of bits shifted and the index to the bit map byte are used to develop the sector number represented by the bit.
- 8) The sector number is stored in the FCB link sector field, FCBLSN.
- 9) The user then returned to via the RTS instruction.

FRESECT

Address – \$10C5

Entry Registers – A = Don't Care.

X = IOCB and FCB number times 16.

Y = Don't Care.

Exit Registers – A = Unknown.

X = IOCB and FCB number times 16.

Y = Unknown.

Functions:

- 1) The sector to be freed is in the FCB current sector field, FCBCSN. If the sector number is zero, then FRESECT exits back to the user via the RTS instruction.
- 2) The sector number is divided by eight to determine the bit map byte which represents the sector. The remainder from this division represents the bit within the byte.
- 3) The byte is retrieved from the bit map, the bit is turned on, and the byte placed back into the bit map.
- 4) The number of available sectors field in the VTOC is incremented by one.
- 5) The VTOC write required byte is set to non-zero to indicate that the VTOC has been changed and needs to be written back to the disk.
- 6) The caller is returned to via the RTS instruction.

Chapter Twenty

THE BOOT PROCESS

When the Atari computer is turned on, the routines in the OS ROM will (under certain conditions) read the first sector from the disk in drive one into memory. It will then examine certain specific locations in this record to decide how to boot the disk. In the following discussion, refer to Figure 20-1. The OS ROM code will load BRCNT consecutive sectors (starting with sector one) onto memory, starting at the address contained in BLDADR. When the OS ROM code has finished this task, it will make a JSR call to the code that is seven bytes into the start of the boot area. In the case of FMS, this is the JMP XBCONT instruction at \$706. The XBCONT code will continue the boot load process.

The XBCONT code examines the DFSFLG to see if a DOS.SYS file exists. If the file exists, then the sector number of the first sector in DOS.SYS will be in DFLINK. The routine will then read all the sectors in the chain starting at DFLINK into the memory area pointed to by DFLADR. When the entire DOS.SYS file is read into memory, XBCONT returns to the OS ROM code.

The OS ROM code will eventually vector through the BINTADR so that the FMS can initialize itself. In the DOS 2.0S system, BINTADR points into the DUP.SYS code. DUP.SYS then receives control from the OS ROM rather than the FMS. One of the tasks that DUP.SYS performs during its initialization is to call the FMS initialization routine.

XBCONT

The XBCONT routine (\$714) is entered by the OS ROM code during the boot process to allow the boot process to continue in the manner best suited for the code being booted.

Functions:

- 1) If the DFSFLG indicates that a DOS.SYS file does not exist, then the OS ROM is returned to with the carry set to indicate that the boot has failed.

CHAPTER TWENTY

- 2) The address contained in DFLADR is moved to the zero page address pointer, ZBUFP, and to the DCB buffer pointer field, DCBBUF.
- 3) The sector number contained in DFLINK is loaded into the A, Y register pair, the carry is cleared to indicate read, and BSIO is called to read a DOS.SYS sector.
- 4) The next sector link is obtained from the link field of the data sector just read.
- 5) If the sector link value is zero, then the DOS.SYS end-of-file has been reached. The OS ROM will be returned to with the carry clear to indicate that the boot read was good.
- 6) If the sector link value is not zero, then the zero page buffer pointer and the DCB buffer pointer are incremented by the amount of data in the sector (125 for 810 drives, 253 for 815 drives).
- 7) The process continues by reading the next sector into memory.

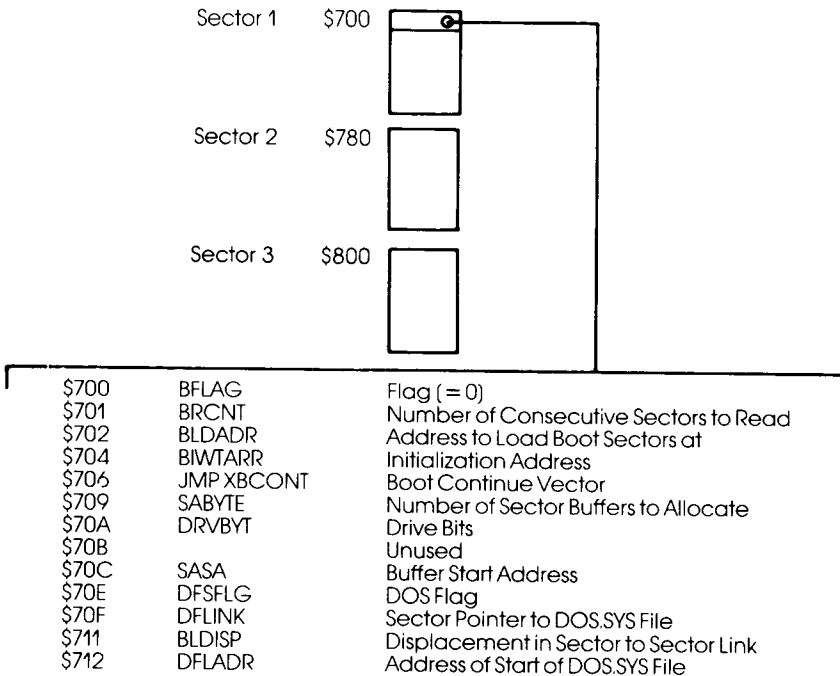


Figure 20-1
Boot Records

Chapter Twenty-One

MAINTAINING THE BOOT RECORD

The boot record (sector 1) contains information about the DOS.SYS file. When DOS.SYS is opened for output, FMS will write all of FMS out to the disk as part of the open process. It will also modify sector zero to indicate that a DOS.SYS file exists and to indicate where on the disk it is. If DOS.SYS is ever Deleted or Renamed (to something not DOS.SYS), then the boot record must be modified to indicate that a DOS.SYS file does not exist. If a file is ever renamed to DOS.SYS, then the boot record is modified to point to the new DOS.SYS file.

WRTDOS

The WRTDOS routine (\$120A) is used to write a new DOS.SYS file to disk and to update the boot record to indicate that a DOS.SYS file exists.

Functions:

- 1) The sector number which is contained in the FCB sector number link field, FCBLSN, is used as the first sector of the DOS.SYS file. This sector number is placed in the boot record area in page seven along with the other necessary information.
- 2) Sectors one, two, and three are written from the memory area from \$700 through \$87F.
- 3) The FMS is written to the DOS.SYS via the WDO routine.
- 4) Exit is via GREAT.

WDO

The WDO routine (\$1267) is used to write the FMS to the DOS.SYS file.

Functions:

- 1) The address contained in DFLADR is moved to the zero page

CHAPTER TWENTY-ONE

buffer pointer, ZBUFP.

- 2) The FMS is copied from its area in memory to the file sector buffer in 125 byte chunks.
- 3) The buffers are written to disk by the WRTNXS routine.
- 4) The process continues until the entire FMS area has been written.
- 5) The caller is returned to via the RTS instruction.

DELDOS

The DELDOS routine (\$1219) is used to modify the boot record to indicate that DOS.SYS does not exist.

Functions:

- 1) The DFSFLG is set to zero to indicate that DOS.SYS does not exist.
- 2) The area from \$700 to \$87F is written to sectors one, two, and three.
- 3) The caller is returned to via the RTS instruction.

ATARI DOS 2.0S

Copyright © 1982 Optimized Systems Software, Inc.

This listing is protected against unauthorized reproduction by the Copyright Law of the United States. Any reproduction utilized for profit or other commercial advantage is precluded without the specific prior written authorization of Optimized Systems Software, Inc., the owner of the copyright. Any such reproduction does not constitute fair use and may subject the individual to both civil and criminal penalties. Federal Law provides for a maximum fine of \$10,000 or imprisonment for not more than one year, or both, for infringement of this copyright.

Contact the President, Optimized Systems Software, Inc., 10379 Lansdale Avenue, Cupertino, California, 95014, prior to reproducing or utilizing any portion of this listing. Any attempt to change the form of publication of this listing, that is, rendering it into machine-readable form or otherwise, is a precluded reproduction if done for profit or other financial advantage.

ATARI DOS 2.0S

FMS - 128/256 BYTE SECTOR (2.0S)
--- Copyright and Author Notice ---

```
0000      1001      .PAGE " --- Copyright and Author Notice ---"
          1002 ;
          1003 ;
          1004 ;COPYRIGHT (C) 1978,1979,1980,1982
          1005 ;OPTIMIZED SYSTEMS SOFTWARE,
          1006 ;CUPERTINO, CA.
          1007 ;
          1008 ;THIS PROGRAM MAY NOT BE REPRODUCED,
          1009 ;STORED IN A RETRIEVAL SYSTEM, OR
          1010 ;TRANSMITTED IN WHOLE OR IN PART,
          1011 ;IN ANY FORM, OR BY ANY MEANS, BE IT
          1012 ;ELECTRONIC,MECHANICAL, PHOTOCOPYING,
          1013 ;RECORDING, OR OTHERWISE WITHOUT THE
          1014 ;PRIOR WRITTEN PERMISSION OF
          1015 ;      OPTIMIZED SYSTEMS SOFTWARE, INC.
          1016 ;      10379 LANSDALE AVENUE
          1017 ;      CUPERTINO, CALIFORNIA 95014 (U.S.A.)
          1018 ;
          1019 ;      PHONE: (408) 446-3099
          1020 ;
          1021 ;
          1022 ;*****
          1023 ;
          1024 ; PROGRAMMER PAUL LAUGHTON
          1025 ;      UPDATED: 19-AUG-80
          1026 ;
          1027 ;*****
          1028 ;
```

System Equates

```
0000      1029      .PAGE "      System Equates"
          1030 ;*****
          1031 ;
          1032 ;
          1033 ;
0700      1034 FMSORG =      $700
0043      1035 FMSZPG =      $43
0340      1036 IOCBORG =    $340
0003      1037 LMASK  =      03      ;LINK MASK
0300      1038 DCBORG =    $300
E453      1039 DHADR  =    $E453
009B      1040 EOL    =      $9B
031A      1041 DEVTAB =    $31A
0020      1042 ZICB  =      $20
02E7      1043 LMADR  =    $2E7
1540      1044 DUPINIT =    $1540      ;INIT ADDR FOR DUP
0102      1045 STAK  =      $102      ;STACK LOC FOR PUT BYTE
00DF      1046 OSBTM  =      $DF      ;HI BYTE OF ADDR LESS THAN OS
          SPACE
0246      1047 DSKTIM =    $246      ;ADDR OF OS WORST CASE DISK
          TIME OUT
000F      1048 TIMEOUT =    15      ;TIME OUT VALUEE OF 15 SECS.
```

IOCB

```
0000      1049      .PAGE "      IOCB"
0000      1050      *= IOCBORG
          1051 ;
          1052 ; IOCB - IO CONTROL BLOCK
          1053 ; THERE ARE 8 I/O CONTROL BLOCKS
          1054 ; 1 IOCB IS REQUIRED FOR EACH
          1055 ; CURRENTLY OPEN DEVICE OR FILE
          1056 ;
```

ATARI DOS 2.0S

```

1057 IOCB
0340 1058 ICHID *= +1 ;DEVICE NUMBER
0341 1059 ICDNO *= +1 ;DEVICE HANDLER
0342 1060 ICCOM *= +1 ; I/O COMMAND
0343 1061 ICSTA *= +1 ;I/O STATUS
0344 1062 ICBAL *= +1
0345 1063 ICBAH *= +1 ;BUFFER ADR (H,L)
0346 1064 ICPUT *= +2 ;PUT CHAR DH ADDR
0348 1065 ICBLL *= +1
0349 1066 ICBLH *= +1 ;BUFFER LEN (H,L)
034A 1067 ICAUX1 *= +1 ;AUX 1
034B 1068 ICAUX2 *= +1 ;AUX 2
034C 1069 ICAUX3 *= +1 ;AUX 3
034D 1070 ICAUX4 *= +1 ;AUX 4
034E 1071 ICAUX5 *= +1 ;AUX 5
034F 1072 ICAUX6 *= +1 ;AUX 6
0010 1073 ICLEN = *-IOCB
1074 ;
0350 1075 *= **ICLEN*7 ;SPACE FOR 7 MORE IOCB'S
1076 ;
1077 ; ICCOM VALUE EQUATES
1078 ;
0001 1079 ICOIN = $01 ;OPEN INPUT
0002 1080 ICOOUT = $02 ;OPEN OUTPUT
0003 1081 ICIO = $03 ;OPEN UN/OUT
0004 1082 ICGBR = $04 ;GET BINARY RECORD
0005 1083 ICGTR = $05 ; GET TEXT RECORD
0006 1084 ICGBC = $06 ;GET BINARY CHAR
0007 1085 ICGTC = $07 ;GET TEXT CHAR
0008 1086 ICPBR = $08 ;GET BINARY RECORD
0009 1087 ICPTR = $09 ;PUT TEXT RECORD
000A 1088 ICPBC = $0A ;PUT BINARY CHAR
000B 1089 ICPTC = $0B ;PUT TEXT CHAR
000C 1090 ICCLOSE = $0C ;CLOSE FILE
000D 1091 ICSTAT = $0D ;GET STATUS
000E 1092 ICDDC = $0E ;DEVICE DEPENDENT
000F 1093 ICMAX = $0E ;MAX VALUE
000F 1094 ICFREE = $0F ;IOCB FREE INDICATOR
1095 ;
1096 ; ICSTA VALUE EQUATES
1097 ;
0001 1098 ICSOK = $01 ;STATUS GOOD, NO ERRORS
0002 1099 ICSTR = $02 ;TRUNCAIATED RECORD

```

IOCB

```

0003 1100 ICSEOF = $03 ;END OF FILE
0003 1101 ICSBRK = $80 ;BREAK KEY ABORT
0031 1102 ICSDNR = $81 ;DEVICE NOT READY
0032 1103 ICSNED = $82 ;NON EXISTENT DEVICE
0033 1104 ICSDER = $83 ;DATA ERROR
0034 1105 ICSIVC = $84 ;INVALID COMMAND
0035 1106 ICSNOP = $85 ;DEVICE/FILE NOT OPEN
0036 1107 ICSIVN = $86 ;INVALID IOCB #
0037 1108 ICSWPC = $87 ;WRITE PROTECT
1109 ;
1110 ; ZERO PAGE IOCB LABELS
1111 ;
0021 1112 ICDNOZ = ICDNO-IOCB+ZICB
0028 1113 ICBL LZ = ICBLL-IOCB+ZICB ;BUF LEN
0029 1114 ICBLHZ = ICBLH-IOCB+ZICB
0024 1115 ICBALZ = ICBAL-IOCB+ZICB ;BUF ADDR
0025 1116 ICBAHZ = ICBAH-IOCB+ZICB
0022 1117 ICCOMZ = ICCOM-IOCB+ZICB
0026 1118 ICPUTZ = ICPUT-IOCB+ZICB ;PUT RTN ADDR

```

ATARI DOS 2.0S

DCB

```
17A0      1119      .PAGE "      DCB"
17A0      1120      *=      DCBORG
          1121      ;
          1122      ; DCB - DATA CONTROL BLOCK
          1123      ; THE DCB IS AN IOCB LIKE CONTROL
          1124      ; BLOCK USED TO INTERFACE THE DISK
          1125      ; FILE MANAGEMENT SYSTEM TO THE
          1126      ; DISK HANDLER
          1127      ;
          1128      DCB
0300      1129      DCBSBI *=      *+1      ;SERIAL BUS ID
0301      1130      DCBDRV *=      *+1      ;DISK DRIVE #
0302      1131      DCBCMD *=      *+1      ;COMMAND
0303      1132      DCBSTA *=      *+1      ;I/O STATUS
0304      1133      DCBBUF *=      *+2      ;I/O BUFFER ADDR (H,L)
0305      1134      DCBTO  *=      *+2      ;TIME OUT CNT
0303      1135      DCBCNT *=      *+2      ;I/O BYTE COUNT
030A      1136      DCBSEC *=      *+2      ;I/O SECTOR NUMBER
          1137      ;
          1138      ; DCBCMD VALUE EQUATES
          1139      ;
0052      1140      DCBCRS =      'R      ;Read sector      ($52)
0050      1141      DCBCWS =      'P      ;Put sector      ($50)
0053      1142      DCBCST =      'S      ;Status request ($53)
0021      1143      DCBCFD =      'I      ;FORMAT DISKETTE ($21)
          1144      ;
          1145      ; *** SPECIAL NOTE:
          1146      ;      DCBCWS may be changed to 'W ($57)
          1147      ;      if desired to have disk perform
          1148      ;      a verifying read after each write.
          1149      ;      Disk write ('W) operations will take
          1150      ;      longer, but will be more reliable.
          1151      ;
          1152      ;
          1153      ; DCBSTA VALUE EQUATES
          1154      ;
0001      1155      DCBSOK =      $01      ;STATUS NORMAL
0001      1156      DCBDNR =      $81      ;DEVICE NOT READY
0002      1157      DCBCNR =      $82      ;CONTROLLER NOT READY
0003      1158      DCBDER =      $83      ;DATA ERROR
0004      1159      DCBIVC =      $84      ;INVALID COMMAND
0007      1160      DCBWPR =      $87      ; WRITE PROTECT
```

ZERO PAGE

```
030C      1161      .PAGE "      ZERO PAGE"
030C      1162      *=      FMSZPG
          1163      ;
0043      1164      ZBUFP  *=      *+2      ;BUFFER PTR
0045      1165      ZDRVA  *=      *+2      ;ZERO PG DRIVE PTR
0047      1166      ZSBA   *=      *+2      ;ZERO PG SECTOR BUF PTR
0049      1167      ERRNO  *=      *+1      ;ERROR NUMBER
          1168      ;
          1169      ;
004A      15         .INCLUDE #E:
004A      20         .INCLUDE #D:ATFMS1.SRC
```

BOOT RECORD

```
004A      2000      .PAGE "BOOT RECORD"
004A      2001      *=      FMSORG
          2002      ;
          2003      ; THE FOLLOWING BYTES ARE STORED
          2004      ; ON DISK SECTOR 0 THEY COMPRISE
```

ATARI DOS 2.0S

```

2005 ; THE BOOT LOAD RECORD
2006 ;
0700 00 2007 BFLG .BYTE 0 ;BOOT FLAG UNUSED=0
0701 03 2008 BRCNT .BYTE 3 ;NO CONSECITIVE BOOT RECORDS TO
; READ
0702 0007 2009 BLDADR .WORD FMSORG ;BOOT LOAD ADDR
0704 4015 2010 BINTADR .WORD DUPINIT ;INIT ADDR
0706 4C1407 2011 BCONT JMP XBCONT ;BOOT READ CONT PT
2012 ;
2013 ; THE FOLLOWING BYTES ARE SET BY
2014 ; THE CONSOLE PROCESSOR. THEY ARE
2015 ; ACTED UPON DURING FMS INIT ONLY.
2016 ; THEY ARE PART OF THE BOOT RECORD
2017 ; THUS DEFINING THE DEFAULT
2018 ;INITIALIZATION PARMS
2019 ;
0709 03 2020 SABYTE .BYTE 3 ;MAX # CONCURRENT OPEN FILES
070A 01 2021 DRVBYT .BYTE 01 ;DRIVE BITS
070B 00 2022 SAFBFW .BYTE 0 ;STORAGE ALLOCATION DIR SW
070C 0115 2023 SASA .WORD ENDFMS ;STORAGE ALLOCATION START ADDR
2024 ;
2025 ; THE FOLLOWING CODE READS THE FMS
2026 ; AND CONSOLE PROCESSOR (DOS) FROM
2027 ; THE DOS.SYS FILE
2028 ;
070E 00 2029 DFSFLG .BYTE 0 ;DOS FLAG
2030 ;
2031 ; 00 NO DOS FILE
2032 ; 01 128 BYTE SECTOR DISK
2033 ; 02 256 BYTE SECTOR DISK
2034 ;
070F 00 2035 DFLINK .BYTE 0,0 ;DOS FILE START SECTOR NUMBER
0710 00
0711 7D 2036 BLDISP .BYTE 125 ;DISPL TO SECTOR LINK
0712 CB07 2037 DFLADR .WORD DFMSDH ;ADDR START OF DOS.SYS FILE
2038 ;
2039 XBCONT
00714 AC0E07 2040 LDY DFSFLG ;GET DOS FLAG
0717 F036 2041 BEQ BFAL ;BR IF NO DOS.SYS FILE
2042 ;
0719 AD1207 2043 LDA DFLADR ;MOVE LOAD START ADDR
071C 8543 2044 STA ZBUFF ;TO ZERO PAGE PTR
071E 8D0403 2045 STA DCBBUF ; AND TO DCB
0721 AD1307 2046 LDA DFLADR+1
0724 8544 2047 STA ZBUFF+1
0726 8D0503 2048 STA DCBBUF+1
2049 ;

BOOT RECORD
2050 ;
0729 AD1007 2051 LDA DFLINK+1 ;GET 1ST SECTOR #
072C AC0F07 2052 LDY DFLINK
072F 18 2053 XBC1 CLC
0730 AE0E07 2054 LDX DFSFLG ;LOAD DISK TYPE CODE
0733 206C07 2055 JSR BSIO ;GO READ BOOT SECTOR
0736 3017 2056 BMI BFAL
2057 ;
0738 AC1107 2058 LDY BLDISP ;POINT TO LINK
073B B143 2059 LDA (ZBUFF),Y ;GET LINK HI
073D 2903 2060 AND #LMASK ;MASK TO LINK BITS
073F 48 2061 PHA
0740 C8 2062 INY
0741 1143 2063 ORA (ZBUFF),Y
0743 F00E 2064 BEQ BGOOD
0745 B143 2065 LDA (ZBUFF),Y ;GET LINK LOW
0747 AB 2066 TAY

```


ATARI DOS 2.0S

```

0748 205707 2067      JSR  INCBA      ;GO INCREMENT BUF ADR
                2068 ;
074B 68      2069      PLA              ;RESTORE LINK HI
074C 4C2F07 2070      JMP  XBC1      ;GO READ NEXT SECTOR
                2071 ;
074F A9C0    2072 BFAIL LDA  #$C0      ;SET FOR CARRY SET
0751 D001    2073      BNE  XBRTN     ;ANY P,Y = $80
                2074 ;
0753 68      2075 BGOOD PLA              ;SET FOR CARRY CLEAR
                2076 ;
0754 0A      2077 XBRTN ASL  A              ;
0755 AB      2078      TAY              ;
0756 60      2079      RTS              ;
                2080 ;
0757 18      2081 INCBA CLC              ;
0758 A543    2082      LDA  ZBUFP     ;INC BUFFER PTR
075A 6D1107 2083      ADC  BLDISP    ;BY DATA LINK (125)
075D 8D0403 2084      STA  DCBBUF
0760 8543    2085      STA  ZBUFP
0762 A544    2086      LDA  ZBUFP+1
0764 6900    2087      ADC  #0
0766 8D0503 2088      STA  DCBBUF+1
0769 8544    2089      STA  ZBUFP+1
076B 60      2090      RTS              ;
                2091 ;

```

SECTOR I/O

```

076C          2092      .PAGE "SECTOR I/O"
                2093 ;
                2094 ; BSIO - DO SECTOR I/O
                2095 ;
076C          2096 BSIO = *
                2097 ;
076C 8D0B03 2098      STA  DCBSEC+1 ; SET SECTOR HI
076F 8C0A03 2099      STY  DCBSEC  ;SECTOR LO
                2100 ;
0772 A952    2101 BSIOR  LDA  #DCBCRS ;ASSUME READ SECTOR
0774 A040    2102      LDY  #$40   ;AND GET DATA
0776 9004    2103      BCC  DSIO1  ;BR IF READ
                2104 ;
0778 A950    2105      LDA  #DCBCWS ;ELSE LOAD WRITE SECTOR
077A A080    2106      LDY  #$80   ;AND PUT DATA
                2107 ;
                2108 DSIO1
077C 8D0203 2109      STA  DCBCMD  ;SET COMMAND
077F 8C0303 2110      STY  DCBSTA  ;AND SIO CMD
                2111 ;
0782 A931    2112      LDA  #$31   ;DISK SERIAL BUS ID
0784 A00F    2113      LDY  #TIMOUT ;TIMEOUT DEFAULT LOADED
                2114 ;
                2115 DSIO2
0786 8D0003 2116      STA  DCBSBI  ;SET ID
0789 8C0603 2117      STY  DCBTO   ;SET TIME OUT
                2118 ;
078C A903    2119      LDA  #3     ;SET RETRY COUNT
078E 8DFF12 2120      STA  RETRY
                2121 ;
0791 A900    2122      LDA  #0     ;ASSUME 128 BYTE
0793 A080    2123      LDY  #$80   ;SECTOR DISK
0795 CA      2124      DEX
0796 F004    2125      BEQ  DSIO3  ;SO BR
                2126 ;
0798 A901    2127      LDA  #1     ;ELSE IS 256
079A A000    2128      LDY  #0
                2129 ;
079C 8D0903 2130 DSIO3 STA  DCBCNT+1 ;SET I/O BYTE CNT

```

ATARI DOS 2.0S

```

079F 8C0803 2131      STY  DCBCNT
                2132 ;
                2133 DSIO4
07A2 2059E4 2134      JSR  $E459      ;CALL SERIAL I/O
07A5 101D     2135      BPL  DSIO5      ;IF GOOD I/O THEN RTS
                2136 ;
07A7 CEFF12 2137      DEC  RETRY      ;TST IF ANOTHER RETRY AVAIL
07AA 3018     2138      BMI  DSIO5      ;NO THEN RTS WITH ERROR
                2139 ;
07AC A240     2140      LDX  #$40      ;DO RETRY-RESET TYPE ACTION
07AE A952     2141      LDA  #DCBCRS   ;ASSUME READ-CK IF IS
07B0 CD0203 2142      CMP  DCBCMD   ;IF COMMAND GET SECTOR

SECTOR I/O
07B3 F009     2143      BEQ  STRTYP   ;YES THEN STORE GETSECTOR IN O
07B5 A921     2144      LDA  #DCBCFD   ;TEST IF FORMAT CMD
07B7 CD0203 2145      CMP  DCBCMD   ;IT ALSO RECIEVES DATA
07BA F002     2146      BEQ  STRTYP   ;YES THEN SET AS GET DATA
07BC A280     2147      LDX  #$80      ;ELSE STORE PUTSECTOR
07BE 8E0303 2148      STRTYP STX  DCBSTA
                2149 ;
07C1 4CA207 2150      JMP  DSIO4      ;RETRY THE I/O
                2151 ;
07C4 AE0113 2152      DSIO5 LDX  CURFCB   ;RELOAD CURRENT FCB
07C7 AD0303 2153      LDA  DCBSTA   ;AND I/O STATUS SET FLAGS
07CA 60       2154      RTS
                2155 ;

FILE MANGER ENTRY POINT
07CB         2156      .PAGE "FILE MANGER ENTRY POINT"
                2157 ;
                2158 ; DFMSDH - DISK FILE MANAGEMENT DISK
                2159 ; HANDLER ENTRY POINT
                2160 ;
                2161 DFMSDH
07CB AA08     2162      .WORD DFMOFN-1 ;OPEN FILE
07CD 140B     2163      .WORD DFMCLS-1 ;CLOSE FILE
07CF BE0A     2164      .WORD DFMGET-1 ;GET FILE
07D1 CB09     2165      .WORD DFMPUT-1 ;PUT BYTE
07D3 000B     2166      .WORD DFMSTA-1 ;STATUS
07D5 A60B     2167      .WORD DFMDDC-1 ;DEVICE DEPENDENT CMD
                2168 ;
                2169 ; INITIALIZATION CODE
                2170 ;
                2171 ; GIVE ROOM FOR BOOT EXPANSION !!!
                2172 ;
07D7         2173      *=    $7E0
07E0         2174      DINIT =    *
                2175 ;
                2176 ; SET UP DRIVE INFO
                2177 ;
                2178 ; DRVTL - 8 BYTES-ONE FOR EACH POSSIBLE DRIVE
                2179 ;
                2180 ; 0 = NO DRIVE
                2181 ; 1 = 128 BYTE SECTOR DRIVE
                2182 ; 2 = 256 BYTE SECTOR DRIVE
                2183 ;
                2184 ; DBUFA(L,H) 8 TWO BYTE ENTRYS THE
                2185 ; DRIVE (VTOC) BUFFER ADR FOR A DRIVE
                2186 ;
07E0 AD0C07 2187      LDA  SASA      ;MOVE START OF ALLOC
07E3 8543     2188      STA  ZBUFP     ;AREA TO ZBUFP
07E5 AD0D07 2189      LDA  SASA+1
07E8 8544     2190      STA  ZBUFP+1
                2191 ;

```

ATARI DOS 2.0S

```

07EA AD0A07 2192      LDA  DRVBYT      ;TEMP 1 IS DRIVE
07ED 8D0C13 2193      STA  TEMP1      ;EXCESS BITS FROM BOOT
                2194 ;
07F0 A207 2195      LDX  #7         ;TEMP 2 IS
                2196 ;
07F2 8E0D13 2197 DIA  STX  TEMP2      ;DR # MINUS 1
07F5 0E0C13 2198      ASL  TEMP1      ;SHIFT DR BIT TO CARRY
07F8 B00D 2199      BCS  DIHAVE     ;BR IF DR EXISTS
                2200 ;
07FA A900 2201      LDA  #0         DRVTBL,X ;SET NO DRIVE
07FC 9D1113 2202      STA  DRVTBL,X
07FF 9D2913 2203      STA  DBUFAL,X
0802 9D3113 2204      STA  DBUFAH,X
0805 F036 2205      BEQ  DIDDEC     ;GO DEC DRIVE #
                2206 ;

```

FILE MANGER ENTRY POINT

```

                2207 DIHAVE
0807 A005 2208      LDY  #DVDWRQ    ;SET WRITE READ OFF
0809 A900 2209      LDA  #0
080B 9143 2210      STA  (ZBUF),Y ;IN THE DRIVE BUFFER
                2211 ;
080D E8 2212      INX                ;PUT DR # IN DCB
080E 8E0103 2213     STX  DCBDRV
0811 A953 2214      LDA  #DCBCST   ;GET DRIVE STATUS
0813 8D0203 2215     STA  DCBCMD
0816 2053E4 2216     JSR  DHADR
                2217 ;
0819 A002 2218      LDY  #2         ;ASSUME 256 BYTE DRIVE
081B ADEA02 2219     LDA  $2EA      ;GET STATUS BYTE
081E 2920 2220      AND  #$20
0820 D001 2221      BNE  DI256     ;BR IF 256
0822 88 2222      DEY
                2223 ;
0823 98 2224 DI256 TYA
0824 AE0D13 2225     LDX  TEMP2      ;SET DR TYPE INTO
0827 9D1113 2226     STA  DRVTBL,X ;TBL AT DRIVE DISPL
082A A543 2227      LDA  ZBUF      ;MOVE CURRENT ALLOC
082C 9D2913 2228     STA  DBUFAL,X ;ADDR TO DBUFA
082F A544 2229      LDA  ZBUF+1    ;AND INC ALLOC
0831 9D3113 2230     STA  DBUFAH,X ;BY 128 BYTES
0834 207008 2231     JSR  DINCBP   ;VIA DINCBP
                2232 ;
0837 88 2233      DEY                ;IF DR WAS A
0838 F003 2234     BEQ  DIDDEC     ;128 BYTES THEN DONE
                2235 ;
083A 207008 2236     JSR  DINCBP   ;ELSE INC PTR BY 128
                2237 ;
083D CA 2238 DIDDEC DEX                ;DEC DRIVE
083E 10B2 2239     BPL  DIA        ;BR IF MORE TO TEST
                2240 ;
                2241 ; SET UP SECTOR ALLOCATION TABLE
                2242 ;
                2243 ; THE SECTOR ALLOCATION TABLE (SECTBL)
                2244 ; WAS 16 ONE BYTE ENTRIES ONE FOR
                2245 ; EACH POSSIBLE 128 BYTE BUFFER SABYTE
                2246 ; IN THE BOOT RECORD DETERMINES THE
                2247 ; NUMBER OF ENTRIES TO ALLOCATE
                2248 ; NON-ALLOCATED BYTE ARE MINUS
                2249 ;
                2250 ; SABUF(L,H) CONTAINS THE ADDR OF THE SECTOR BUFFER
                2251 ;
0840 AC0907 2252     LDY  SABYTE    ;GET AND SAVE COUNT
0843 A200 2253      LDX  #0
                2254 ;
0845 A900 2255     DINTS LDA  #0        ;ASSUME ALLOCATE

```

ATARI DOS 2.0S

```

0047 88      2256      DEY          ;DEC COUNT OF ALLOCATED
0048 1001    2257      BPL   DISETS   ;IF PLUS STILL ALLOCATE
004A 98      2258      TYA          ;ELSE DE ALLOCATE

```

FILE MANGER ENTRY POINT

```

                2259 ;
004B 9D1913 2260 DISETS STA  SECTBL,X ;SET ALLOCATE BYTE
004E 98      2261      TYA          ;IF NO ALLOCATED
004F 300D    2262      BMI   DISNI   ;THEN DON'T ALLOCATE BUF
                2263 ;
0051 A543   2264      LDA  ZBUFF  ;MOVE BUFFER ADDR
0053 9D3913 2265      STA  SABUFL,X ;TO SECTOR BUF PTR
0056 A544   2266      LDA  ZBUFF+1
0058 9D4913 2267      STA  SABUFH,X
005B 207008 2268      JSR  DINCBP  ;INC SECTOR ADDR
                2269 ;
005E E8     2270 DISNI  INX          ;INC BUF #
005F E010   2271      CPX   #16    ;IF NOT ALL 16
0061 D0E2   2272      BNE  DINXTS  ;DO AGAIN
                2273 ;
                2274 ; SET LOW MEM
                2275 ;
0063 A543   2276      LDA  ZBUFF  ;MOVE FINAL ADDR
0065 8DE702 2277      STA  LMADR  ;TO LOW MEM PTR
0068 A544   2278      LDA  ZBUFF+1
006A 8DE802 2279      STA  LMADR+1
                2280 ;
006D 4C7E08 2281      JMP  CLRFCB  ;CONT INIT
                2282 ;
                2283 ; DINCBP - INC ZBUFF BY 128
                2284 ;
0070 18     2285 DINCBP CLC
0071 A543   2286      LDA  ZBUFF
0073 6980   2287      ADC  #128
0075 8543   2288      STA  ZBUFF
0077 A544   2289      LDA  ZBUFF+1
0079 6900   2290      ADC  #0
007B 8544   2291      STA  ZBUFF+1
007D 60     2292      RTS
                2293 ;
                2294 ; CLEAR FCBS TO ZERO
                2295 ;
007E       2296 CLRFCB =      *
007E A07F   2297      LDY  #$7F   ;128 OF FCB
0080 A900   2298      LDA  #0
0082 998113 2299 CFCBX  STA  FCB,Y   ;TO BE CLEARED
0085 88     2300      DEY
0086 D0FA   2301      BNE  CFCBX
                2302 ;

```

FILE MANGER ENTRY POINT

```

0088       2303      . PAGE
                2304 ;
0088 A000   2305      LDY  #0
008A B91A03 2306 ADI1  LDA  DEVTAB,Y ;FIND AH
008D F00C   2307      BEQ  ADI2   ;UNUSED
008F C944   2308      CMP  #'D    ;OR DISK
0091 F008   2309      BEQ  ADI2   ;EMPTY
0093 C8     2310      INY
0094 C8     2311      INY
0095 C8     2312      INY
0096 C01E   2313      CPY  #30
0098 D0F0   2314      BNE  ADI1
009A 00     2315      BRK          ;ELSE BREAK
                2316 ;

```

ATARI DOS 2.0S

```

089B A944 2317 ADI2 LDA #'D ;SET DISK
089D 991A03 2318 STA DEVTAB,Y
08A0 A9CB 2319 LDA #DFMSDH&255 ;SET FMS ADDR
08A2 991B03 2320 STA DEVTAB+1,Y
08A5 A907 2321 LDA #DFMSDH/256
08A7 991C03 2322 STA DEVTAB+2,Y
2323 ;
08AA 60 2324 RTS

OPEN

08AB 2325 .PAGE "OPEN"
2326 ;
2327 ; DFMOPN - FILE OPEN EXECUTION ENTRY PT
2328 ;
2329 DFMOPN

08AB 206411 2330 JSR SETUP ; DO FCB SET UP
08AE 209E0E 2331 JSR FNDCODE ;GO DECODE FILE NAME
08B1 BD4A03 2332 LDA ICAUX1,X ; GET AUX1 (OPEN TYPE CODES)
08B4 9D8213 2333 STA FCBOTC,X ;PUT INTO FCB
08B7 2902 2334 AND #OPDIR ; IS THIS LIST DIRECTORY
08B9 F003 2335 BEQ OPN1 ;BR IF NOT
08BB 4CAD0D 2336 JMP LISTDIR ;GOTO DIR LIST CODE
2337 ;
08BE 20210F 2338 OPN1 JSR SFDIR ;GO SEARCH FILE DIR
08C1 08 2339 PHP
2340 ;
08C2 BD8213 2341 LDA FCBOTC,X ;GET OPEN TYPE CODE
08C5 C904 2342 CMP #OPIN ;INPUT
08C7 F00F 2343 BEQ DFOIN
08C9 C908 2344 CMP #OPOUT ;OUTPUT
08CB F044 2345 BEQ DFOOUT
08CD C90C 2346 CMP #OPIN+OPOUT ;UPDATE
08CF F00C 2347 BEQ DFOUPD
08D1 C909 2348 CMP #OPOUT+OPAPND ;APPEND
08D3 F017 2349 BEQ DFOAPN
08D5 4CBF12 2350 JMP ERDVDC ;ERROR
2351 ;
2352 ; DFOIN - OPEN FOR INPUT
2353 ;
08D8 2354 DFOIN = *
08D8 28 2355 PLP ;GET SEARCH FLAG
08D9 B00E 2356 BCS OPNER1 ;ERROR IF NOT FOUND
08DB 9006 2357 BCC DFOUI
2358 ;
2359 ; DFOUPD - OPEN FOR UPDATA
2360 ;
08DD 2361 DFOUPD = *
08DD 28 2362 PLP ;GET SEARCH FLAG
08DE B009 2363 BCS OPNER1 ;BR NOT FOUND
08E0 20AC0C 2364 JSR TSTLOCK ;TEST LOCK
2365 ;
08E3 2366 DFOUI = *
08E3 20AE09 2367 JSR DFRDSU ;SET UP FOR READ
08E6 4CF012 2368 JMP GREAT ;DONE
2369 ;
08E9 4CBB12 2370 OPNER1 JMP ERFNF ;FILE NOT FOUND

OPEN

08EC 2371 .PAGE
2372 ;
2373 ; DFOAPN - OPEN APPEND
2374 ;
08EC 2375 DFOAPN = *
08EC 28 2376 PLP ;GET READ STATUS

```

ATARI DOS 2.0S

```

08ED B0FA 2377      BCS  OPNER1      ;BR NOT FOUND
08EF AC0513 2378    LDY  CDIRD       ;IF OLD.
08F2 B90114 2379    LDA  FILDIR+DFDFL1,Y ;FILE TYPE
08F5 2902 2380      AND  #DFDNLD     ;THEN
08F7 F015 2381      BEQ  APOER       ;ERROR
08F9 20AC0C 2382    JSR  TSTLOCK    ;TEST LOCKED
08FC 20BF10 2383    JSR  OPVTOC     ;READ VTOC
08FF 200611 2384    JSR  GETSECTOR  ;GET A SECTOR
0902 9D8E13 2385    STA  FCBSN+1,X  ;MOVE START SECTOR #
0905 BD8B13 2386    LDA  FCBLN,X    ;TO START SECTOR #
0908 9D8D13 2387    STA  FCBSN,X
090B 4C7C09 2388    JMP  DHFOX2     ;CONTINUE AS OPEN
090E 4CB712 2389    APOER JMP  ERAPO
                2390 ;
                2391 ; DFOOUT - OPEN FOR OUTPUT
                2392 ;
0911                2393 DFOOUT = *
0911 28 2394        PLP                    ;GET SEARCH FLAG
0912 B009 2395      BCS  DFOX1
                2396 ;
0914 20530C 2397    JSR  XDEL0      ;DELETE THE FILE OR FILES
0917 AC0513 2398    LDY  CDIRD
091A 4C4809 2399    JMP  OPN1A
                2400 ;
091D                2401 DFOX1 = *
091D AD0213 2402    LDA  DHOLES     ;WAS THERE A HOLE
0920 3070 2403      BMI  OPNER2     ;BR IF NO HOLE
0922 8D0613 2404    STA  CDIRS     ;SAVE HOLE SECTOR AS CURRENT
                DIR SEC
0925 206E10 2405    JSR  RDDIR     ;GO READ CURRENT DIR SECTOR
0928 AD0313 2406    LDA  DHOLED    ;MOVE HOLE DISPL TO
092B 8D0513 2407    STA  CDIRD     ;CUR DIR DISPL
092E AD0413 2408    LDA  DHFNUM    ;MOVE HOLE FN
0931 8D0713 2409    STA  SFNUM     ;TO CURRENT
0934 20BF10 2410    JSR  OPVTOC
0937 AC0513 2411    LDY  CDIRD
093A A20A 2412      LDX  #10
093C A920 2413      LDA  #S20
093E 990614 2414    OPN1B STA  FILDIR+DFDFPN,Y ;BLANK FILL FILE ENTRY
                FOR FILE NAME
0941 C8 2415        INY
0942 CA 2416        DEX
0943 10F9 2417      BPL  OPN1B
0945 AE0113 2418    LDX  CURFCB
                2419 ;
0948                2420 OPN1A = *
0948 200611 2421    JSR  GETSECTOR ;GET A SECTOR

OPEN
094B AC0513 2422    LDY  CDIRD       ;GET DIR DISPL
094E 990514 2423    STA  FILDIR+DFDSSN+1,Y ;PUT SECTOR INTO DIR
                REC
0951 BD8B13 2424    LDA  FCBLN,X
0954 990414 2425    STA  FILDIR+DFDSSN,Y
                2426 ;
0957 A943 2427      LDA  #DFDINU+DFDOUT+DFDNLD ;SET DIR ENTRY IN
                USE
0959 990114 2428    STA  FILDIR+DFDFL1,Y
095C A900 2429      LDA  #0          ; SET NOT LOCKED
095E 990314 2430    STA  FILDIR+DFDCNT+1,Y ;SET COUNT = 0
0961 990214 2431    STA  FILDIR+DFDCNT,Y
                2432 ;
0964 A200 2433      LDX  #0
0966 BD5913 2434    OPN2 LDA  FNAME,X    ;MOVE FILE NAME
0969 C93F 2435      CMP  #'?         ;IF WILD CARD
096B F003 2436      BEQ  OPN2A     ;CHANGE TO BLANK

```

ATARI DOS 2.0S

```

096D 990614 2437 STA FILDIR+DFDPFN,Y ;TO DIRECTORY
0970      2438 OPN2A = *
0970 C8      2439 INY
0971 E8      2440 INX
0972 E00B    2441 CPX #11
0974 90F0    2442 BCC OPN2
          2443 ;
0976 AE0113 2444 LDX CURFCB ;RESTORE X REG
0979 207110 2445 JSR WRDIR ;GO WRITE DIRECTORY
097C      2446 DHFOX2 = *
097C 209509 2447 JSR SETFCB
097F 20E20F 2448 JSR WRN6 ;FIX UP AS IF WRITE
0982 A980    2449 OPN3 LDA #FCBFAS ;SET NEW FILE
0984 9D8513 2450 STA FCBSLG,X
0987 209B12 2451 JSR TSTDOS ;IF NOT DOS
098A D003    2452 BNE DHFOX3 ;BR
098C 4C0A12 2453 JMP WRTDOS ;ELSE DO IT
098F      2454 DHFOX3 = *
098F 4CF012 2455 JMP GREAT
          2456 ;
0992 20BD12 2457 OPNER2 JSR ERDFULL ;DIRECTORY FULL
          2458 ;
          2459 ;
0995      2460 SETFCB = *
0995 A900    2461 LDA #0 ;CLEAR
0997 9DR513 2462 STA FCBSLG,X ;FLAG
099A AD0713 2463 OPNF1 LDA SFNUM ;MOVE FILE NUM TO FCB
099D 0A      2464 ASL A
099E 0A      2465 ASL A
099F 9D8113 2466 STA FCBFNO,X
09A2 A900    2467 LDA #0
09A4 9D8713 2468 STA FCBDLN,X ;DATA LENGTH
09A7 9D8F13 2469 STA FCBCNT,X ;SET CNT = 0
09AA 9D9013 2470 STA FCBCNT+1,X
09AD 60      2471 RTS
09AE 209509 2472 DFRDSU JSR SETFCB ;SET UP FCB
09B1 AC0513 2473 LDY CDIRD ;MOVE START SECTOR TO LINK

OPEN
09B4 B90114 2474 LDA DFDFL1+FILDIR,Y ;SET NEW
09B7 2902    2475 AND #DFDNLN ;SECTOR
09B9 9D8413 2476 STA FCBSLT,X ;FLAG
09BC B90414 2477 LDA FILDIR+DFDSSN,Y
09BF 9D8B13 2478 STA FCBSLN,X
09C2 B90514 2479 LDA FILDIR+DFDSSN+1,Y
09C5 9D8C13 2480 STA FCBSLN+1,X
09C8 201710 2481 JSR RDNSO ;READ 1ST SECTOR
09CB 60      2482 RTS
09CC      25 ;
09CC      30 .INCLUDE #E:
          .INCLUDE #D:ATFMS2.SRC

PUT BYTE
09CC      3000 .PAGE "PUT BYTE"
          3001 ;
          3002 ; DFMPUT - PUT A FILE BYTE
          3003 ;
          3004 DFMPUT
09CC 8D0813 3005 STA SVDBYT
09CF BD4103 3006 LDA ICDNO,X
09D2 8521    3007 STA ICDNO-IOCB+ZICB
09D4 206411 3008 JSR SETUP
09D7 AC0013 3009 LDY ENTSTK ;CHK TO SEE IF ENTRY WASN'T
          FROM CIO
09DA B90201 3010 LDA STAK,Y ;IF HI BYTE RTS IS NOT IN OS
          ADDR

```

ATARI DOS 2.0S

```

09DD C9DF 3011      CMP #OSBTM      ;SPACE THEN A NON-CIO ENTRY
09DF B004 3012      BCS FRMCIO     ;BR IF FROM CIO
09E1 A900 3013      LDA #0          ;ELSE PREVENT FROM DOING BURST
                                I/O
09E3 8522 3014      STA ICCOMZ
09E5 BD8213 3015   FRMCIO LDA FCBOTC,X  ;IF NOT OPEN
09E8 2908 3016      AND #OPOUT     ;OUTPUT
09EA F02D 3017      BEQ PUTER      ;ERROR
09EC BC8713 3018      LDY FCBDLN,X  ;GET DATA LENGTH
09EF 98 3019        TYA
09F0 DD8613 3020      CMP FCMBLN,X  ;IF SECTOR NOT FULL
09F3 9011 3021      BCC PUT1      ;THEN BR
09F5 20940F 3022     JSR WRNXXS   ;ELSE WRITE FULL SECTOR
09F8 B022 3023      BCS PEOF      ;BR IF EOF
09FA 201F0A 3024     JSR WTBUR    ;TEST BURST
09FD A000 3025      LDY #0
09FF B005 3026      BCS PUT1      ;BR IF NOT BURST
0A01 B124 3027      LDA (ICBALZ),Y ;PUT NEXT BYTE
0A03 8D0813 3028     STA SVDBYT  ;AFTER BURST AREA
                                3029 ;
0A06 FE8713 3030   PUT1  INC FCBDLN,X  ;INC DATA LEN
0A09 AD0813 3031     LDA SVDBYT  ;GET DATA BYTE
0A0C 9147 3032      STA (ZSBA),Y ;AND PUT IN SECTOR BUFFER
                                3033 ;
0A0E A940 3034      LDA #FCBFMS  ;INDICATE SECTOR MODIFIED
0A10 1D8513 3035     ORA FCBFLG,X
0A13 9D8513 3036     STA FCBFLG,X
                                3037 ;
0A16 4CF012 3038     JMP GREAT    ;DONE
                                3039 ;
0A19 4CBF12 3040   PUTER JMP ERDVDC
0A1C 4CF412 3041   PEOF  JMP ERREOF

```

BURST I/O

```

0A1F          3042      .PAGE "BURST I/O"
                                3043 ;
                                3044 ; TEST BURST I/O AND DO IF POSSIBLE
                                3045 ;
0A1F BD8513 3046   WTBUR LDA FCBFLG,X  ;IF NOT ACQUIRING SECTORS
0A22 1026 3047      BPL NOBURST  ;THEN UPDATE AND
0A24 3002 3048      BMI TBURST   ;NO BURST
                                3049 ;
0A26 A900 3050   RTBUR  LDA #0          ;SET READ TYPE
                                3051 ;
0A28 8D1013 3052   TBURST STA BURTYP   ;SET BURST TYPE
0A2B A522 3053      LDA ICCOMZ     ;IF CMD
0A2D 2902 3054      AND #2         ;IS TEXT MODE
0A2F F019 3055      BEQ NOBURST   ;THEN NO BURST
                                3056 ;
0A31 20AE0A 3057     JSR TBLEN    ;IF USER BUFFER LESS
0A34 B014 3058      BCS NOBURST  ;THEN SECTOR, NO BURST
                                3059 ;
0A36 A524 3060      LDA ICBALZ   ;MOVE USER BUFFER
0A38 8547 3061      STA ZSBA     ;ADDR TO SECTPOR
0A3A A525 3062      LDA ICBHZ   ;BUFFER PTR
0A3C 8548 3063      STA ZSBA+1
                                3064 ;
0A3E AD1013 3065   NXTBUR LDA BURTYP   ;GET I/O TYPE
0A41 3009 3066      BMI WRBUR    ;BR IF WRITE
                                3067 ;
0A43 200F10 3068     JSR RDNXTS  ;DO SECTOR READ
0A46 9033 3069      BCC BBINC    ;BR IF EOF
0A48 B053 3070      BCS BUREOF   ;BR RD EOF
                                3071 ;
0A4A 38 3072     NOBURST SEC      ;INDICATE NO BURST
0A4B 60 3073      RTS

```


ATARI DOS 2.0S

```

3074 ;
0A4C ADF812 3075 WRBUR LDA DRVMDL ;WRITE FULL SECTOR
0A4F 9D8713 3076 STA FCBDLN,X ;DATA COUNT
3077 ;
0A52 A8 3078 TAY
0A53 B147 3079 LDA (ZSBA),Y ;SAVE DATA TO BE
0A55 8D0913 3080 STA SVD1 ;TO BE CLOBBERED
0A58 C8 3081 INY
0A59 B147 3082 LDA (ZSBA),Y ;BY WRTNXT
0A5B 8D0A13 3083 STA SVD2
0A5E C8 3084 INY
0A5F B147 3085 LDA (ZSBA),Y
0A61 8D0B13 3086 STA SVD3
3087 ;
0A64 20940F 3088 JSR WRTNXS ;WRITE SECTOR
3089 ;
0A67 ACF812 3090 LDY DRVMDL ;RESTORE CLOBBERED DATA
0A6A AD0913 3091 LDA SVD1
0A6D 9147 3092 STA (ZSBA),Y

BURST I/O

0A6F C8 3093 INY
0A70 AD0A13 3094 LDA SVD2
0A73 9147 3095 STA (ZSBA),Y
0A75 C8 3096 INY
0A76 AD0B13 3097 LDA SVD3
0A79 9147 3098 STA (ZSBA),Y
3099 ;
3100 ;
0A7B 18 3101 BBINC CLC
0A7C A547 3102 LDA ZSBA ;INC SECTOR
0A7E 7D8613 3103 ADC FCBMLN,X ;BUFFER ADDR BY
0A81 8547 3104 STA ZSBA ;ACTUAL DATA LEN
0A83 A548 3105 LDA ZSBA+1 ;GOT OT PUT
0A85 6900 3106 ADC #0
0A87 8548 3107 STA ZSBA+1
3108 ;
0A89 38 3109 SEC
0A8A A528 3110 LDA ICBL LZ ;DEC USER
0A8C FD8613 3111 SBC FCBMLN,X ;BUFFER LEN BY
0A8F 8528 3112 STA ICBL LZ ;ACTUAL DATA LEN
0A91 A529 3113 LDA ICBLHZ ;GOT OR PUT
0A93 E900 3114 SBC #0
0A95 EA 3115 NOP
0A96 8529 3116 STA ICBLHZ
3117 ;
0A98 20AE0A 3118 JSR TBLN ;IF USER BUF LEN
0A9B 90A1 3119 BCC NXTBUR ;NOW >= SECTOR, DO AGAIN
3120 ;
0A9D 3121 BUREOF = * ;END OF BURSTING
0A9D A547 3122 LDA ZSBA ;MOVE FINAL ADDR BACK
0A9F 8524 3123 STA ICBALZ ;TO USER BUF PTR
0AA1 A548 3124 LDA ZSBA+1
0AA3 8525 3125 STA ICBAHZ
3126 ;
0AA5 BC8813 3127 LDY FCBBUF,X ;RESTORE ZSBA
0AA8 88 3128 DEY
0AA9 20D011 3129 JSR SSBA
3130 ;
0AAC 18 3131 BURST CLC
0AAD 60 3132 RTS
3133 ;
3134 ; TEST USER BUF LEN FOR BURST
3135 ;
0AAE 3136 TBLN = . *
0AAE ADFE12 3137 LDA DRVTYP ;IF DRIVE NOT

```

ATARI DOS 2.0S

```

ØAØ1 C9Ø1 3138      CMP #1          ;128 BYTE SECTOR TYPE
ØAØ3 DØØ4 3139      BNE TBL256       ;THEN DO 256 BYTE TEST
                    314Ø ;
ØAØ5 A528 3141      LDA ICBL LZ
ØAØ7 3ØF3 3142      BMI BURST
                    3143 ;
ØAØ9 A529 3144 TBL256 LDA ICBLHZ      ;IF BUF LEN HI >= 256

```

BURST I/O

```

ØAØB DØEF 3145      BNE BURST       ;THEN CAN BURST
ØAØD 38 3146       SEC
ØAØE 6Ø 3147       RTS

```

GET BYTE

```

ØAØF 3148          .PAGE "GET BYTE"
                    3149 ;
                    315Ø ;
                    3151 ; DFMGET - GET A FILE BYTE
                    3152 ;
ØAØF 3153 DFMGET = *
ØAØF 2Ø6411 3154    JSR SETUP       ;GO SET UP
ØAØ2 BD8213 3155    LDA FCBOTC,X     ;IF OPEN FOR
ØAØ5 29Ø2 3156     AND #ØPDIR      ;DIR CNT
ØAØ7 FØØ3 3157     BEQ GET1
ØAØ9 4CB9ØD 3158    JMP GDCHAR      ;THEN GO TO DIR RTN
                    3159 ;
ØAØC BD8713 316Ø GET1 LDA FCBDLN,X   ;GET DATA LEN
ØAØF DD8613 3161    CMP FCBMLN,X   ;TEST EMPTY SECTOR
ØAØD 9ØØB 3162     BCC GET2        ;BR IF NOT EMPTY
ØAØ4 2Ø26ØA 3163    JSR RTBUR      ;DO BURST IF POSSIBLE
ØAØ7 2ØØF1Ø 3164    JSR RDNXTS     ;GET NEXT SECTOR
ØAØA 9ØFØ 3165     BCC GET1        ;BR IF NOT EOF
ØAØC 3166 GEOF = *
ØAØC 4CF412 3167    JMP ERREOF     ;ELSE EOF ERROR
                    3168 ;
ØAØF A8 3169 GET2  TAY
ØAØØ B147 317Ø     LDA (ZSBA),Y    ;GET DATA BYTE
ØAØ2 8DØ813 3171    STA SVDBYT     ;SAVE THE BYTE
ØAØ5 C8 3172      INY
ØAØ6 98 3173      TYA
ØAØ7 9D8713 3174    STA FCBDLN,X   ;AND SET NEW VALUE
ØAØA 3175 EFLOOK = *
ØAØA BC8B13 3176    LDY FCBL SN,X   ;DO EOF LOOK AHEAD
ØAØD DØØF 3177     BNE GET3        ;IF LSN NOT ZERO
ØAØE BC8C13 3178    LDY FCBL SN+1,X ;THEN
ØAØ2 DØØA 3179     BNE GET3        ;NOT EOF
ØAØ4 DD8613 318Ø    CMP FCBMLN,X   ;IF LSN=Ø THEN CHECK FOR
ØAØ7 9ØØ5 3181     BCC GET3        ;LAST BYTE
ØAØ9 A9Ø3 3182     LDA #ØØ3       ;IF LAST BYTE THEN RTS
ØAØFB 4CD312 3183    JMP RETURN
                    3184 ;
ØAØE 4CFØ12 3185 GET3 JMP GREAT

```

STATUS

```

ØØØ1 3186          .PAGE "STATUS"
                    3187 ;
                    3188 ; DFMSTA - GET A FILE STATUS
                    3189 ;
                    319Ø DFMSTA
ØØØ1 2Ø6411 3191    JSR SETUP       ; SETUP
ØØØ4 2Ø9EØE 3192    JSR FNDCODE     ; DECODE FILE NAME
ØØØ7 2Ø21ØF 3193    JSR SFDIR      ; SEARCH FOR FILE
ØØØA BØØ6 3194     BCS SFNF      ; BR NOT FOUND

```

ATARI DOS 2.0S

```

0B0C 20AC0C 3195      JSR  TSTLOCK   ;TEST LOCKED
0B0F 4CF012 3196      JMP  GREAT     ;FILE EXISTS AND UNLOCKED
          3197 ;
0B12 4CBB12 3198 SFNF JMP  ERFNF

```

CLOSE

```

0B15          3199      .PAGE "CLOSE"
          3200 ;
          3201 ; DFMCLOSE - CLOSE A FILE
          3202 ;
          3203 DFMCLS
0B15 206411 3204      JSR  SETUP
0B18 BD8213 3205      LDA  FCBOTC,X  ;GET OPEN CODE
0B1B 2908   3206      AND  #OPOUT    ;IF NOT OUTPUT
0B1D F04E   3207      BEQ  CLDONE    ;THEN DONE
          3208 ;
0B1F 3E8513 3209      ROL  FCBFLG,X  ;IF NOT ACQUIRING SECTORS
0B22 9051   3210      BCC  CLUPDT    ;THEN IS UPDATE
          3211 ;
0B24 20AB0F 3212      JSR  WRTLSEC   ;WRITE LAST SECTOR
          3213 ;
0B27 20800B 3214      JSR  RRDIR     ;GO GET DIRECTORY
0B2A BD9013 3215      LDA  FCBCNT+1,X ;GET CNT OF SECTORS
0B2D 48     3216      PHA
0B2E BD8F13 3217      LDA  FCBCNT,X
0B31 48     3218      PHA
          3219 ;
0B32 BD8213 3220      LDA  FCBOTC,X  ;GET OPEN CODE
0B35 2901   3221      AND  #OPAPND   ;IF NOT APPEND
0B37 F017   3222      BEQ  CLOUT     ;BR
          3223 ;
0B39 20AE09 3224      JSR  DFRDSU   ;ELSE SET UP FOR READ
0B3C 200F10 3225 APP1 JSR  RDNXTS    ;READ TO EOF
0B3F 90FB   3226      BCC  APP1
          3227 ;
0B41 BD8D13 3228      LDA  FCBSN,X   ;MOVE START SECTOR
0B44 9D8B13 3229      STA  FCBLSN,X  TO EOF LINK SECTOR
0B47 BD8E13 3230      LDA  FCBSN+1,X
0B4A 9D8C13 3231      STA  FCBLSN+1,X
0B4D 20B30F 3232      JSR  WRN2     ;THEN WRITE AS NOT EOF
          3233 ;
0B50 AC0513 3234 CLOUT LDY  CDIRD     ;GET DIR DISPL
0B53 18     3235      CLC
0B54 68     3236      PLA
0B55 790214 3237      ADC  FILDIR+DFDCNT,Y
0B58 990214 3238      STA  FILDIR+DFDCNT,Y
0B5B 68     3239      PLA
0B5C 790314 3240      ADC  FILDIR+DFDCNT+1,Y
0B5F 990314 3241      STA  FILDIR+DFDCNT+1,Y
          3242 ;
0B62 A942   3243      LDA  #DFDINU+DFDNLD ;SET ENTRY TO IN USE
0B64 990114 3244      STA  FILDIR+DFDFLL,Y
0B67 207110 3245      JSR  WRDIR    ;WRITE DIR
0B6A 209510 3246      JSR  WRTVTOC ;WRITE VTOC
          3247 ;
0B6D A900   3248 CLDONE LDA  #0      ;CLEAR OPEN CODE
0B6F 9D8213 3249      STA  FCBOTC,X

```

CLOSE

```

0B72 4CEA12 3250      JMP  FGREAT
          3251 ;
0B75          3252 CLUPDT = *
0B75 3E8513 3253      ROL  FCBFLG,X  ;IF SECTOR NOT MODIFIED
0B78 90F3   3254      BCC  CLDONE    ;THEN DONE

```

ATARI DOS 2.0S

```

0B7A 20F80F 3255      JSR  WRCSIO      ;ELSE WRITE IT
0B7D 4C6D0B 3256      JMP   CLDONE     ; THEN DONE
                          3257 ;

CLOSE

0B80          3258      .PAGE
                          3259 ;
                          3260 ; RE-READ DIR RECORD
                          3261 ;
0B80          3262      RRDIR = *
0B80 BD8113 3263      LDA  FCBFNO,X ;GET FILE NUMBER
0B83 4A     3264      LSR  A
0B84 4A     3265      LSR  A
0B85 8D0713 3266      STA  SFNUM
                          3267 ;
                          3268 ;

0B88 209B0B 3269      JSR  FNSHFT ;SET ACU=FILE NO/64
0B8B 8D0613 3270      STA  CDIRS ;TO GET DIR SECTOR
0B8E 209B0B 3271      JSR  FNSHFT ;SET ACU TO REM=16
0B91 209D0B 3272      JSR  FNSHF1
0B94 0A     3273      ASL  A
0B95 8D0513 3274      STA  CDIRD ;TO GET DIR DISPL
                          3275 ;
0B98 4C6E10 3276      JMP  RDDIR
0B9B A900   3277      FNSHFT LDA #0
0B9D A003   3278      FNSHF1 LDY #3 ;SHIFT 3 BITS OF
0B9F 1E8113 3279      FNSHF2 ASL FCBFNO,X ;FILE NO INTO ACU
0BA2 2A     3280      ROL  A
0BA3 88     3281      DEY
0BA4 D0F9   3282      BNE  FNSHF2
0BA6 60     3283      RTS

DEVICE DEPENDENT COMMAND

0BA7          3284      .PAGE "DEVICE DEPENDENT COMMAND"
                          3285 ;
                          3286 ; DFMDCC - DEVICE DEPENDENT CMD EXECUTION
                          3287 ;
                          3288 DFMDCC

0BA7 206411 3289      JSR  SETUP ;SET UP FOR EXECUTION
0BAA BD4203 3290      LDA  ICCOM,X ;GET COMMAND
0BAD C9FE   3291      CMP  #254 ;IS IT FORMAT
0BAF F025   3292      BEQ  XFV ;BR IF
0BB1 C927   3293      CMP  #MAXDCC ;TEST RANGE
0BB3 B01E   3294      BCS  DVDCER ;BR OUT OF RANGE
0BB5 38     3295      SEC
0BB6 E920   3296      SBC  #$20 ;SUBTRACT BASE OF CMDS
0BB8 9019   3297      BCC  DVDCER ;BR OUT OF RANGE
0BBA 0A     3298      ASL  A
0BBB A8     3299      TAY
0BBC B9C50B 3300      LDA  DVDCVT,Y
0BBF 48     3301      PHA  ;PUSH EXECUTION ADDR
0BC0 B9C60B 3302      LDA  DVDCVT+1,Y
0BC3 48     3303      PHA
0BC4 60     3304      RTS
                          3305 ;
                          3306 DVDCVT

0BC5 0BD8   3307      .DBYTE XRENAME-1 ;20-RENAME
0BC7 0C31   3308      .DBYTE XDELETE-1 ;21-DELETE
0BC9 0BD2   3309      .DBYTE DVDCER-1 ;INVALID CMD
0BCB 0C7B   3310      .DBYTE XLOCK-1 ;23-LOCK
0BCD 0C82   3311      .DBYTE XUNLOCK-1 ;24-UNLOCK
0BCF 0CB9   3312      .DBYTE XPOINT-1 ;25-POINT
0BD1 0D02   3313      .DBYTE XNOTE-1 ;26-NOTE
                          3314 ;
0027          3315      MAXDCC = $27 ;MAX DVDC+1

```

ATARI DOS 2.0S

```

3316 ;
0BD3 4CBF12 3317 DVDRCR JMP ERDVDC
0BD6 4C180D 3318 XFV JMP XFORMAT ;FORMAT VECTOR

RENAME

0BD9 3319 .PAGE "RENAME"
3320 ;
3321 ;XRENAME - RENAME A FILE OR FILES
3322 ;
3323 XRENAME
0BD9 209E0E 3324 JSR FNDCODE ;DECODE FILE NAME
0BDC 8C0D13 3325 STY TEMP2 ;SAVE FNAME INDEX
0BDF 20210F 3326 JSR SFDIR ;GO FINE FILE IN DIR
0BE2 9003 3327 BCC XRN1 ;BR IF FOUND
0BE4 4CBB12 3328 JMP ERFNF
3329 ;
0BE7 20AC0C 3330 XRN1 JSR TSTLOCK ;TEST LOCK
0BEA 209B12 3331 JSR TSTDOS ; IF NOT DOS
0BED D003 3332 BNE XRN1A ;THEN
0BEF 201912 3333 XRN1 JSR DELDOS ; DON'T CHANGE SO
3334 XRN1A
0BF2 AC0D13 3335 LDY TEMP2 ;GET INDEX FOR END FN1
0BF5 20B40E 3336 JSR FNDCNX ;GO DECODE NEXT FILE NAME
0BF8 209B12 3337 JSR TSTDOS ;IF NOT DOS
0BFB D00F 3338 BNE XRN1B ;THEN
0BFD AC0513 3339 LDY DIRD
0C00 B90514 3340 LDA FILDIR+DFDSSN+1,Y
0C03 48 3341 PHA
0C04 B90414 3342 LDA FILDIR+DFDSSN,Y
0C07 A8 3343 TAY ;A,Y NEW DOS
0C08 68 3344 PLA
0C09 205312 3345 JSR SETDSO ;GO WRITE SECTOR ZERO
3346 ;
3347 XRN1B
0C0C A200 3348 LDX #0
0C0E AC0513 3349 LDY DIRD
3350 ;
0C11 BD5913 3351 XRN2 LDA FNAME,X ;MOVE FILE NAME
0C14 C93F 3352 CMP #'? ;FROM FNAME TO DIR ENT
0C16 F003 3353 BEQ XRN3 ;BUT DON'T CHANGE WILD CARD
0C18 990614 3354 STA FILDIR+DFDPPN,Y ;CHARS INDICATED IN
FNAME
0C1B C8 3355 XRN3 INY
0C1C E8 3356 INX
0C1D E00B 3357 CPX #11
0C1F 90F0 3358 BCC XRN2
0C21 AE0113 3359 XRN1 LDX CURFCB ;RESTORE X-REG
3360 ;
0C24 207110 3361 XRN1 JSR WRDIR ;GO WRITE CIR DIR RECORD
3362 ;
0C27 209E0E 3363 JSR FNDCODE ;GET OLD FILENAME AGAIN
0C2A 20310F 3364 JSR CSFDIR ;CONTINUE SEARCH OF DIR
0C2D 90B8 3365 BCC XRN1 ;BR IF FOUND ANOTHER
3366 ;
0C2F 4CEA12 3367 JMP FGREAT ;GO TO GOOD ENDING

DELETE

0C32 3368 .PAGE "DELETE"
3369 ;
3370 ; XDELETE - DELETE ALL FILENAMES THAT MATCH
3371 ;
3372 XDELETE
0C32 209E0E 3373 JSR FNDCODE ;GO DECODE FILENAME
0C35 20210F 3374 JSR SFDIR ;SEARCH DIR FOR FILENAME
```

ATARI DOS 2.0S

```

0C38 B03F 3375      BCS  DFNF      ;BR NOT FOUND
0C3A          3376 XDELX = *
0C3A 20530C 3377      JSR  XDEL0
0C3D 209B12 3378      JSR  TSTDOS
0C40 D003 3379      BNE  XDELY
0C42 201912 3380      JSR  DELDOS
          3381 XDELY
          3382 ;
0C45 207110 3383 XDEL3 JSR  WRDIR      ;WRITE DIR ENTRY
0C48 20310F 3384      JSR  CSFDIR      ;LOOK FOR NEXT MATCH
0C4B 90ED 3385      BCC  XDELX      ;BR IF FOUND
0C4D 209510 3386      JSR  WRTVTOC
0C50 4CEA12 3387      JMP  FGREAT
          3388 ;
0C53 20BF10 3389 XDEL0 JSR  OPVTOC
          3390 ;
0C56 AC0513 3391 XDEL1 LDY  CDIRD      ;GET DIR DISPL
0C59 20AC0C 3392      JSR  TSTLOCK     ;GO TEST LOCK
0C5C A980 3393      LDA  #DFDEDE    ;LOAD DELETED FLAG
0C5E 990114 3394      STA  FILDIR+DFDFL1,Y ;DELETE FILE
          3395 ;
0C61 20AE09 3396      JSR  DFRDSU
0C64 4C6C0C 3397      JMP  XDEL2A
          3398 ;
0C67 200F10 3399 XDEL2 JSR  RDNXTS     ;READ NEXT SECTOR
0C6A B006 3400      BCS  XDEL4
0C6C          3401 XDEL2A = *
0C6C 20C510 3402      JSR  FRESECT    ;FREE CURRENT SECTOR
0C6F 4C670C 3403      JMP  XDEL2
          3404 ;
0C72          3405 XDEL4 = *
0C72 A005 3406      LDY  #DWDWRQ    ;TURN ON WRITE REQ'D
0C74 A9FF 3407      LDA  #$FF
0C76 9145 3408      STA  (ZDRVA),Y
0C78 60 3409      RTS
          3410 ;
0C79 4CBB12 3411 DFNF  JMP  ERFNF      ;FILE NOT FOUND

LOCK AND UNLOCK

0C7C          3412      .PAGE "LOCK AND UNLOCK"
          3413 ;
          3414 ; XLOCK - LOCK A FILE
          3415 ; XUNLOCK - UNLOCK A FILE
          3416 ;
          3417 XLOCK
0C7C A920 3418      LDA  #DFDLOC    ; SET LOCK
0C7E 8D0F13 3419      STA  TEMP4
0C81 D005 3420      BNE  XLCOM      ;GO TO COMMON
          3421 XUNLOCK
0C83 A900 3422      LDA  #0        ;SET UNLOCK
0C85 8D0F13 3423      STA  TEMP4
          3424 ;
0C88 209E0E 3425 XLCOM JSR  FNDCODE     ; DECODE FILE NAME
0C8B 20210F 3426      JSR  SFDIR      ;FIND 1ST MATCH
0C8E 9003 3427      BCC  XLC1      ;BR MATCH FOUND
0C90 4CBB12 3428      JMP  ERFNF      ;BR NOT FOUND
          3429 ;
0C93 AC0513 3430 XLC1 LDY  CDIRD      ;GET CURRENT DISPL
0C96 B90114 3431      LDA  FILDIR+DFDFL1,Y ;GET LOCK BYTE
0C99 29DF 3432      AND  #$DF      ;TURN OFF LOCK
0C9B 0D0F13 3433      ORA  TEMP4      ;OR IN LOCK/UNLOCK
0C9E 990114 3434      STA  FILDIR+DFDFL1,Y ;SET NEW LOCK BYTE
0CA1 207110 3435      JSR  WRDIR      ;GO WRITE
          3436 ;
0CA4 20310F 3437      JSR  CSFDIR      ;LOOK FOR NEXT MATCH
0CA7 90EA 3438      BCC  XLC1      ;BR FOUND

```

ATARI DOS 2.0S

```

0CA9 4CEA12 3439      JMP  FGREAT      ;ELSE DONE
          3440 ;
          3441 ; TSTLOCK - TEST FILE LOCKED
          3442 ;
          3443 TSTLOCK
0CAC AC0513 3444      LDY  CDIRD      ;GET DIR DISPL
0CAF B90114 3445      LDA  FILDIR+DFDFL1,Y ;LOAD LOCK BYTE
0CB2 2920      3446      AND  #DFDLOC    ;MASK LOCK BIT
0CB4 D001      3447      BNE  TLF        ;BR IF LOCKED
0CB6 60        3448      RTS
          3449 ;
0CB7 4CC112 3450 TLF  JMP  ERFLOCK

POINT
0CBA          3451      .PAGE "POINT"
          3452 ;
          3453 ; XPOINT - POINT REQUEST
          3454 ;
          3455 XPOINT
0CBA BD8513 3456      LDA  FCBFLG,X  ;IF ARQ SECTORS
0CBD 3041      3457      BMI  PERR1    ;POINT INVALID
0CBF BD4D03 3458      LDA  ICAUX4,X  ;IF REQUEST IS NOT
0CC2 DD8A13 3459      CMP  FCBCSN+1,X ;SAME AS CURRENT
0CC5 D008      3460      BNE  XP1        ;THEN BR
0CC7 BD4C03 3461      LDA  ICAUX3,X
0CCA DD8913 3462      CMP  FCBCSN,X
0CCD F01E      3463      BEQ  XP2        ;ELSE NO NEED TO CHANGE
          3464 ;
0CCF BD8513 3465 XP1  LDA  FCBFLG,X  ;IF NOT MODIFIED
0CD2 F008      3466      BEQ  XP1A       ;BR
0CD4 20F80F 3467      JSR  WRCSIO   ;ELSE WRITE IT
0CD7 A900      3468      LDA  #0
0CD9 9D8513 3469      STA  FCBFLG,X
0CDC          3470 XP1A  =      *
0CDC BD4D03 3471      LDA  ICAUX4,X
0CDF 9D8C13 3472      STA  FCBLSN+1,X
0CE2 BD4C03 3473      LDA  ICAUX3,X
0CE5 9D8B13 3474      STA  FCBLSN,X
0CE8 201710 3475      JSR  RDNSO   ;READ REQ SECTOR
0CEB B00A      3476      BCS  XPERR
          3477 ;
0CED BD4E03 3478 XP2  LDA  ICAUX5,X  ;TEST REQ DATA LEN
0CF0 DD8613 3479      CMP  FCBMLN,X  ;LESS THEN MAX
0CF3 9005      3480      BCC  XP3
0CF5 F003      3481      BEQ  XP3
0CF7          3482 XPERR  =      *
0CF7 4CC312 3483      JMP  ERRPDL   ;IF NOT THEN ERROR
          3484 ;
0CFA 9D8713 3485 XP3  STA  FCBDLN,X  ;SET NEW DATA LEN
0CFD 4CF012 3486      JMP  GREAT
          3487 ;
0D00 4CB912 3488 PERR1 JMP  ERRPOT

NOTE
0D03          3489      .PAGE "NOTE"
          3490 ;
          3491 ; XNOTE - EXECUTE NOTE REQUEST
          3492 ;
          3493 XNOTE
0D03 BD8713 3494      LDA  FCBDLN,X  ;DATA LENGHT VALUE
0D06 9D4E03 3495      STA  ICAUX5,X  ;TO AUX 2
0D09 BD8913 3496      LDA  FCBCSN,X  ;CUR SEC NO (LO)
0D0C 9D4C03 3497      STA  ICAUX3,X  ;TO AUX 3
0D0F BD8A13 3498      LDA  FCBCSN+1,X ;CUR SEC NO (HI)

```

ATARI DOS 2.0S

```

0D12 9D4D03 3499      STA  ICAUX4,X  ;TO AUX 4
0D15 4CF012 3500      JMP  GREAT

```

FORMAT

```

0D18          3501      .PAGE "FORMAT"
                3502 ;
                3503 ; XFORMAT - FORMAT A DISKETTE
                3504 ;
                3505 XFORMAT
0D18 A548      3506      LDA  ZSBA+1      ;MOVE VTOC BUF ADR
0D1A 8D0503    3507      STA  DCBBUF+1    ;TO DCB
0D1D A547      3508      LDA  ZSBA
0D1F 8D0403    3509      STA  DCBBUF
0D22 A921      3510      LDA  #DCBCFD      ;FORMAT
0D24 8D0203    3511      STA  DCBCMD      ;TO DCB
0D27 A940      3512      LDA  #$40        ;TELL SIO RECIEVING DATA
0D29 8D0303    3513      STA  DCBSTA
0D2C AEF012    3514      LDX  DRVTYPE      ;GET DR TYPE 128 OR 256
0D2F A931      3515      LDA  #$31        ;BUS I.D.
0D31 AC4602    3516      LDY  DSKTIM      ;GET FORMAT TIME OUT VALUE
0D34 208607    3517      JSR  DSIO2      ;GOTO LOCAL DISK HANDLER THEN
                                SIO
                                3518 ;
0D37 1019      3519      BPL  XF0        ;IF NO ERRORS CONT FORMATING
0D39 C090      3520      CPY  #$90        ;ELSE CK FOR DEVICE DONE ERROR
0D3B D012      3521      BNE  XFERR      ;NO, THEN ERROR EXIT
                                3522 ;
0D3D          3523      TSTFMT = *      ;ELSE CK FOR BAD SECTOR INFO
0D3D A000      3524      LDY  #0          ;RETURNED BY CONTROLLER
0D3F B147      3525      LDA  (ZSBA),Y
0D41 C9FF      3526      CMP  #$FF
0D43 D007      3527      BNE  XFBAD      ;BAD SECTORS RET ERR MSG
0D45 C8        3528      INY
0D46 B147      3529      LDA  (ZSBA),Y
0D48 C9FF      3530      CMP  #$FF
0D4A F003      3531      BEQ  XFERR      ;NOT BAD SEC ERR, REQ ERR EXIT
0D4C 4CB512    3532      XFBAD JMP  ERDBAD
                                3533 ;
0D4F 4CD312    3534      XFERR JMP  RETURN    ;DO ERROR EXIT
                                3535 ;
                                3536 XF0
0D52 A900      3537      LDA  #0
0D54 A8        3538      TAY
0D55 9145      3539      XF1  STA  (ZDRVA),Y
0D57 C8        3540      INY
0D58 10FB      3541      BPL  XF1
                                3542 ;
0D5A A000      3543      LDY  #0          ;SET
0D5C A902      3544      LDA  #2          ;TYPE = 2
0D5E 9145      3545      STA  (ZDRVA),Y
0D60 C8        3546      INY
0D61 A9C3      3547      LDA  #$C3      ;SET MSN AND
0D63 9145      3548      STA  (ZDRVA),Y ;NSA=107=2C3
0D65 C8        3549      INY
0D66 C8        3550      INY
0D67 9145      3551      STA  (ZDRVA),Y

```

FORMAT

```

0D69 A902      3552      LDA  #$02
0D6B 88        3553      DEY
0D6C 9145      3554      STA  (ZDRVA),Y
0D6E C8        3555      INY
0D6F C8        3556      INY
0D70 9145      3557      STA  (ZDRVA),Y

```


ATARI DOS 2.0S

```

3558 ;
0D72 A00A 3559 LDY #DVDSMP
0D74 A9FF 3560 LDA #$FF ;SET SECTOR MAP TO
0D76 9145 3561 XF2 STA (ZDRVA),Y ;ALL ONES
0D78 C8 3562 INY
0D79 C064 3563 CPY #DVDSMP+90
0D7B D0F9 3564 BNE XF2
3565 ;
0D7D A90F 3566 LDA #$0F ;DEALLOCATE 1ST 4 SECTORS
0D7F A00A 3567 LDY #DVDSMP ;FOR BOOT
0D81 9145 3568 STA (ZDRVA),Y
3569 ;
0D83 A037 3570 LDY #DVDSMP+45 ;DEALLOCATE MIDDLE 9
0D85 A900 3571 LDA #0
0D87 9145 3572 STA (ZDRVA),Y ;FOR
0D89 C8 3573 INY ;VTOC AND FILE DIR
0D8A A97F 3574 LDA #$7F
0D8C 9145 3575 STA (ZDRVA),Y
3576 ;
0D8E 209510 3577 JSR WRTVTOC ;WRITE THE VTOC
3578 ;
0D91 A900 3579 LDA #0 ;0 FILLE DIR SECTORS
0D93 A8 3580 TAY
0D94 990114 3581 XF3 STA FILDIR,Y ;USE FILE DIR BUFFER
0D97 C8 3582 INY
0D98 10FA 3583 BPL XF3
3584 ;
0D9A A907 3585 LDA #7 ;WRITE TO ALL 8 DIR SECTORS
0D9C 8D0613 3586 STA CDIRS
0D9E 207110 3587 XF4 JSR WRDTR
0DA2 CE0613 3588 DEC CDIRS
0DA5 10F8 3589 BPL XF4
3590 ;
0DA7 201912 3591 JSR DELDOS ;SET NO DOS
3592 ;
0DAA 4CEA12 3593 JMP FGREAT ;DONE

```

LIST DIRECTORY

```

0DA0 3594 .PAGE "LIST DIRECTORY"
3595 ;
3596 ; LISTDIR - LIST THE DIRECTORY
3597 ; GDCHAR - GET NEXT DIR CHARACTER
3598 ; THE DIRECTORY IS LISTED VIA OPEN
3599 ; LIST DIRECTORY FUNCTION EACH DIR
3600 ; ENTRY THAT MATCHES THE FILE SPEC
3601 ; IS CONVERTED TO A PRINTABLE FORMAT
3602 ; INTO A SECTOR BUFFER. THE GET BYTE
3603 ; ENTRY IS USED TO GET THE PRINTABLE
3604 ; CHARACTERS ONE AT A TIME. THE
3605 ; LAST LINE PRINTED IS ALWAYS A
3606 ; COUNT OF THE NUMBET OF SECTORS IN USE
3607 ; AND THE NUMBER REMAINING AVAILABLE SECTORS
3608 ;
3609 LISTDIR
0DA0 A900 3610 LDA #0
0DA0 8D0F13 3611 STA TEMP4
0DB2 20210F 3612 JSR SFDIR ;SEARCH FOR A FILE NAME
0DB5 902C 3613 BCC LDENT1 ;BR IF FOUND
0DB7 B030 3614 BCS LDCNT ;BR IF NOT FOUND
3615 ;
3616 GDCHAR
0DB9 2C0F13 3617 BIT TEMP4 ;TEST FLAG
0DBC 3053 3618 BMI LDDONE ;BR IF ALL DONE
3619 ;
0DBE AC0F13 3620 LDY TEMP4 ;GET COUNT OF CHARS SENT
0DC1 B147 3621 LDA (ZSBA),Y ;GET NEXT CHAR

```

ATARI DOS 2.0S

```

0DC3 8D0813 3622      STA  SVDBYT      ; IN SVDBYT
0DC6 EE0F13 3623      INC  TEMP4       ; INC COUNT
0DC9 C99B  3624      CMP  #EOL        ;TEST IF EOL DONE
0DCB D009  3625      BNE  GDCRTN     ;BR NOT EOL
0DCD C011  3626      CPY  #17         ;WAS THIS AN ENTRY
0DCF B008  3627      BCS  LDENT      ;BR IF IT WAS
0DD1 A980  3628      LDA  #S0         ;ELSE INDICATE END
0DD3 8D0F13 3629      STA  TEMP4       ; IN TEMP4
      3630 ;
0DD6 4CF012 3631 GDCRTN JMP  GREAT      ;DONE
      3632 ;
0DD9 A900  3633 LDENT  LDA  #0         ;CLEAR CHAR COUNTER
0ddb 8D0F13 3634      STA  TEMP4
0DDE 20310F 3635      JSR  CSFDIR     ;SEARCH FOR NEXT MATCH
0DE1 B006  3636      BCS  LDCNT      ;BR NO MORE MATCHES
      3637 LDENT1
0DE3 20210E 3638      JSR  FDENT      ;FORMAT ENTRY
0DE6 4CF012 3639      JMP  GREAT      ;DONE
      3640 ;
0DE9 208B10 3641 LDCNT  JSR  RDVTOC     ;READ VTOC
0DEc A004  3642      LDY  #DVDNSA+1 ;GET # SECTOR AVR
0DEE B145  3643      LDA  (ZDRVA),Y
0DF0 48  3644      PHA

```

LIST DIRECTORY

```

0DF1 88  3645      DEY
0DF2 B145 3646      LDA  (ZDRVA),Y
0DF4 A8  3647      TAY
0DF5 68  3648      PLA
      3649 ;
0DF6 20570E 3650      JSR  CVDX       ;AND CONVERT
      3651 ;
0DF9 A003 3652      LDY  #3         ;SET EOL
0DFB A20C 3653      LDX  #FSCML-1  ;PUT IN CUTE
0DFD BD140E 3654 MVFSCM LDA  FSCM,X      ; MSG
0E00 9147 3655      STA  (ZSBA),Y
0E02 C8  3656      INY
0E03 CA  3657      DEX
0E04 10F7 3658      BPL  MVFSCM
0E06 20670E 3659      JSR  CVDY
      3660 ;
0E09 A900 3661      LDA  #0         ; SET CHAR CNT
0E0B 8D0F13 3662      STA  TEMP4
0E0E 4CEA12 3663      JMP  FGREAT
      3664 ;
      3665 LDDONE
0E11 4CF412 3666      JMP  ERREOF     ;END OF FILE
      3667 ;
0E14 53  3668 FSCM  .BYTE "SROTCE$ EERF "
0E15 52
0E16 4F
0E17 54
0E18 43
0E19 45
0E1A 53
0E1B 20
0E1C 45
0E1D 45
0E1E 52
0E1F 46
0E20 20
0E0D 3669 FSCML = *-FSCM
0E21 35 .INCLUDE #E:
0E21 40 .INCLUDE #D:ATFMS3.SRC

```

ATARI DOS 2.0S

LIST DIRECTORY

```

0E21      4000      .PAGE
          4001 ;
          4002 ; FORMAT DIR ENTRY INTO A SECTOR BUFFER
          4003 ;
          4004 FDENT
0E21 A000      4005      LDY #0      ;START AT DISPL ZERO
0E23 A920      4006      LDA #$20     ;START WITH A BLANK
0E25 9147      4007      STA (ZSBA),Y
0E27 AE0513    4008      LDX CDIRD
0E2A BD0114    4009      LDA FILDIR+DFDFL1,X
0E2D 2920      4010      AND #DFDLOC ;BUT IF FILE LOCKED
0E2F F004      4011      BEQ LD1
0E31 A92A      4012      LDA #'*      ;CHANGE TO AST
0E33 9147      4013      STA (ZSBA),Y
0E35 C8        4014      LD1      INY
0E36 A920      4015      LDA #$20     ;FOLLOWED BY A BLANK
0E38 9147      4016      STA (ZSBA),Y
0E3A C8        4017      INY
          4018 ;
0E3B BD0614    4019      LD2      LDA FILDIR+DFDPPN,X ;MOVE THE I2 CHAR
0E3E 9147      4020      STA (ZSBA),Y ;FILE NAME
0E40 E8        4021      INX
0E42 C8        4022      INY
0E44 C00D      4023      CPY #13
0E46 90F5      4024      BCC LD2
          4025 ;
0E48 A920      4026      LDA #$20     ;FOLLOWED BY A BLANK
0E4B 9147      4027      STA (ZSBA),Y
0E4D C8        4028      INY
0E4F 8C0F13    4029      STY TEMP4     ;SAVE INDEX = 15
          4030 ;
0E4E AE0513    4031      LDX CDIRD
0E51 BC0214    4032      LDY FILDIR+DFDCNT,X ;SET A,Y
0E54 BD0314    4033      LDA FILDIR+DFDCNT+1,X ;=SECTOR COUNT
          4034 ;
          4035 CVDX
0E57 A264      4036      LDX #100     ;CONVERT AND MOVE
0E59 20710E    4037      JSR CVDIGIT ;100S DIGIT
0E5C A20A      4038      LDX #10
0E5E 20710E    4039      JSR CVDIGIT ;10S DIGIT
0E61 98        4040      TYA
0E62 208D0E    4041      JSR STDIGIT ;1S DIGIT
          4042 ;
0E65 A011      4043      LDY #17     ;THEN PUT OUT
0E67 A99B      4044      CVDY     LDA #EOL     ;AND EOL
0E69 9147      4045      STA (ZSBA),Y
0E6E A000      4046      LDY #0
0E6F 8C0F13    4047      STY TEMP4     ;SET CHAR CNT = 0
0E70 60        4048      RTS
          4049 ;
0E71 8E0E13    4050      CVDIGIT STX TEMP3     ;SAVE DIGIT VALUE

```

LIST DIRECTORY

```

0E74 A2FF      4051      LDX #$FF
          4052 ;
0E76 8D0D13    4053      CVD1     STA TEMP2     ;SAVE CURR VALUE HI
0E79 8C0C13    4054      STY TEMP1     ;AND LOW
0E7C E8        4055      INX
0E7E 38        4056      SEC
0E7F AD0C13    4057      LDA TEMP1     ;SUBTRACT DIGIT VALUE
0E81 ED0E13    4058      SBC TEMP3     ;FROM CUR VALUE
0E84 A8        4059      TAY
0E85 AD0D13    4060      LDA TEMP2
0E88 E900      4061      SBC #0

```

ATARI DOS 2.0S

```

0E8A B0EA 4062      BCS  CVD1      ;IF NOT GONE MINUS, DO AGAIN
          4063 ;
0E8C 8A 4064      TXA          ;DIGIT TO ACU
0E8D 0930 4065 STDIGIT ORA #S30 ;PLUS ASCII ZERO
0E8F AC0F13 4066 LDY  TEMP4    ;GET OUTPUT INDEX
0E92 9147 4067      STA  (ZSBA),Y ;AND SET DIGIT
0E94 EE0F13 4068      INC  TEMP4    ; INC OUTPUT INDEX
0E97 AD0D13 4069      LDA  TEMP2    ;LOAD VALUE HI
0E9A AC0C13 4070      LDY  TEMP1    ;AND VALUE LO
0E9D 60 4071      RTS

```

FILE NAME DECODE

```

0E9E          4072      .PAGE "FILE NAME DECODE"
          4073 ;
          4074 ; FNDCODE - DECODE A FILE NAME
          4075 ;
          4076 ; THE USER FILENAME IS POINTED TO BY
          4077 ;ZBUFF, IT IS ON THE FORM P.X WHERE P
          4078 ; IS THE PRIMARY FILE NAME (1 TO 8 CHARS)
          4079 ; AND X IS THE EXTENDED FILE NAME
          4080 ; (0 TO 4 CHARS). THE PERIOD IS OPTIONAL
          4081 ; (IF NOT PRESENT, THEN NO EXTENSION).
          4082 ; THE DECODED FILENAME WILL BE 12 CHARS
          4083 ; IN LENGTH. THE P FIELD WILL BE
          4084 ; LEFT JUSTIFIED IN THE 1ST 8 BYTES.
          4085 ; THE X FIELD WILL BE LEFT JUSTIFIED IN
          4086 ; THE LAST 4 BYTES. BLANKS ARE USED
          4087 ; TO PAD THE FIELDS TO FULL SIZE.
          4088 ; IF THE USER SPECIFIED P OR X FILEDS
          4089 ; CONTAIN MORE THAN 8 OR 4 CHARS, THEN THE
          4090 ; EXTRA CHARS ARE IGNORED. THE '*'
          4091 ; WILD CARD CHAR WILL CAUSE THE REST
          4092 ; OF THE FIELDS TO FILLED WITH THE
          4093 ; '?' WILD CARD CHAR. ANY NON-ALPHANUMERIC
          4094 ; CHAR TERMINATES THE FILENAME.
          4095 ;
          4096 FNDCODE

0E9E BD4403 4097      LDA  ICBAL,X
0EA1 8543 4098      STA  ZBUFF
0EA3 BD4503 4099      LDA  ICBAH,X
0EA6 8544 4100      STA  ZBUFF+1
0EAB A002 4101      LDY  #2          ;FIND THE 'D'
0EAA B143 4102 FD0A  LDA  (ZBUFF),Y
0EAC 88 4103      DEY
0EAD 3058 4104      BMI  FNDERR   ;BR IF 256 CHARS SEEN
0EAF C93A 4105      CMP  #' :
0EB1 D0F7 4106      BNE  FD0A
          4107 FD0B
0EB3 C8 4108      INY
          4109 ;
          4110 FNDCNX

0EB4 A20B 4111      LDX  #11      ;CLEAR FILENAME TO BLANKS
0EB6 A920 4112      LDA  #S20

0EB8 9D5913 4113 FD0  STA  FNAME,X
0EBB CA 4114      DEX
0EBC 10FA 4115      BPL  FD0
          4116 ;
0EBE A200 4117      LDX  #0          ;SET FNAME CHAR CNT TO 0
0EC0 8E0C13 4118      STX  EXTSW   ;SET NOT IN EXTENSION
          4119 ;
          4120 ;
0EC3 C8 4121 FD1  INY          ;INC ZBUFF INDEX
0EC4 B143 4122      LDA  (ZBUFF),Y ;GET BUF CHAR

```

ATARI DOS 2.0S

FILE NAME DECODE

```

                                4123 ;
0EC6 C92A 4124      CMP #'*      ;TEST FOR WILD CARDS
0EC8 D00B 4125      BNE FD3      ;BR NOT WILD CARD
                                4126 ;
0ECA A93F 4127 FD2  LDA #'?      ;LOAD ? WILD CARD
0ECC 200A0F 4128    JSR FDSCHAR  ;GO STORE IT
0ECF 90F9 4129      BCC FD2      ;BR IF PORX NOT FULL
0ED1 10F0 4130      BPL FD1      ; BR IF AT START OF X
0ED3 302E 4131      BMI FDEND     ;BR IF AT X END
                                4132 ;
0ED5 C92E 4133 FD3  CMP #'.'      ;WAS CHAR FIELD SEPERATOR
0ED7 D00C 4134      BNE FD4      ;BR IF NOT
0ED9 2C0C13 4135    BIT EXTSW    ;WAS THERE ALREADY 1 CHAR
0EDC 3025 4136      BMI FDEND     ;BR IF WAS END
0EDE A208 4137      LDX #8      ;ADV FNAME INDEX TO XFIELD
0EE0 6E0C13 4138    ROR EXTSW    ;SET EXTSW - MINUS
0EE3 90DE 4139      BCC FD1      ;CONT WITH NEXT CHAR
                                4140 ;
0EE5 C93F 4141 FD4  CMP #'?      WAS IT WILD CARD
0EE7 F014 4142      BEQ FD6      ;BR IF WILD CARD
                                4143 ;
0EE9 C941 4144      CMP #'A      ;IS CHAR ALPHA
0EEB 9004 4145      BCC FD5      ;BR NOT ALPHA
0EEE C95B 4146      CMP #5B     ;TEXT HI ALPHA
0EEF 900C 4147      BCC FD6      ;BR IF NOT APLHA
                                4148 ;
0EF1 E000 4149 FD5  CPX #0      ;IF FIRST CHAR NOT
0EF3 F012 4150      BEQ FNDERR   ;ALPHA THEN ERROR
                                4151 ;
0EF5 C930 4152      CMP #30     ;IS CHAR NUMERIC
0EF7 900A 4153      BCC FDEND     ;BR NOT NUMERIC (END OF NAME)
0EF9 C93A 4154      CMP #3A     ;TEST NUMERIC HI
0EFB B006 4155      BCS FDEND     ; BR NO NUMBER
                                4156 ;
0EPD 200A0F 4157 FD6 JSR FDSCHAR  ;STORE THE CHAR
0EP0 4CC30E 4158    JMP FD1      ;AND CONTINUE WITH NEXT
                                4159 ;
0F03 AE0113 4160 FDEND LDX CURFCB  ;RESTORE X REG
0F06 60 4161      RTS
                                4162 ;
0F07 4CC512 4163 FNDERR JMP ERRFN   ;INDICATE FILENAME ERROR

```

FMS - 128/256 BYTE SECTOR (2.0S)

FILE NAME DECODE

```

0F0A 4164      .PAGE
                                4165 ;
                                4166 ; FDSCHAR - STORE FILENAME CHAR
                                4167 ;
                                4168 ; ON ENTRY
                                4169 ; A = CHAR
                                4170 ; X = NEXT FN POSITION
                                4171 ;
                                4172 ; ON EXIT
                                4173 ; CARRY - SET IF FIELD FULL
                                4174 ; MINUS - IF START OF EXECUTION
                                4175 ; PLUS - IF END OF EXECUTION
                                4176 ;
                                4177 FDSCHAR
0F0A E008 4178      CPX #8      ;AT EXECUTION
0F0C 900D 4179      BCC FDSC2   ;BR IF NOT
0F0E F005 4180      BEQ FDSC1   ;BR IF 1ST CHAR OF
                                4181 ;
0F10 E00C 4182      CPX #12     ;AT END OF EXIT
0F12 9007 4183      BCC FDSC2   ;BR NOT AT END

```

ATARI DOS 2.0S

```

0F14 60      4184      RTS
              4185 ;
0F15 2C0C13 4186 FDSC1 BIT  EXTSW      ;DO NOT STORE CHAR UNLESS
0F18 3001    4187      BMI  FDSC2      ;PERIOD WAS SEEN
0F1A 60      4188      RTS
              4189 ;
0F1B 9D5913 4190 FDSC2 STA  FNAME,X    ;SET CHAR INTO NAME
0F1E E8      4191      INX              ;INC TO NEXT CHAR
0F1F 18      4192      CLC
0F20 60      4193      RTS

```

DIRECTORY SEARCH

```

0F21          4194          .PAGE "DIRECTORY SEARCH"
              4195 ;
              4196 ; SFDIR - SEARCH FILE DIRECTORY
              4197 ; CSFDIR - FILE DIRECTORY SEARCH
              4198 ;
              4199 ; THE FILE DIRECTORY IS SEARCHED FOR THE
              4200 ; FILENAME IN FNAME. THE SEARCH STARTS
              4201 ; AT THE CENTRAL SECTOR+1 AND WILL CONTINUE
              4202 ; FOR UP TO A TOTAL OF 8 SECTORS. WHEN
              4203 ; TESTING FOR FNAME MATCH, '?' FNAME
              4204 ; CHARS WILL ALWAYS MATCH THE CORESPONDING
              4205 ; DIR FILENAME CHAR. IF A MATCH IS FOUND
              4206 ; CDIRS CONTAINS THE RELATIVE DIRECTORY SECTOR
              4207 ; NUMBER (0-7) AND CDIRD (AND THE Y REG)
              4208 ; CONTAINS THE DISPLACEMENT OF THE ENTRY.
              4209 ; AFTER A MATCH HAS BEEN FOUND, THE DIRECTORY CAN
              4210 ; BE SEARCHED FOR ANOTHER MATCH VIA THE CSFDIR
              4211 ; ENTRY POINT. IF A MATCH HAS NOT BEEN FOUND
              4212 ; THEN DHOLES AND DHOLED WILL POINT TO A
              4213 ; DIRECTORY HOLE THAT CAN BE USED.
              4214 ; IF DHOLED = FF THEN THE DIRECTORY IS FULL.
              4215 ; THE CARRY IS RETURNED CLEAR IF FILE FOUND,
              4216 ; SET IF FILE NOT FOUND.
              4217 ;
              4218 SFDIR

0F21 A9FF    4219          LDA  $FF      ;INIT TO -1
0F23 8D0213 4220          STA  DHOLES   ;DIR HOLE SECTOR
0F26 8D0613 4221          STA  CDIRS    ;CUR DIR SECTOR
0F29 8D0713 4222          STA  SFNUM    ;FILE NUMBER
0F2C A970    4223          LDA  #$70     ;INIT TO -16 (-ENTRY LENGTH)
0F2E 8D0513 4224          STA  CDIRD    ;CUR DIR DISPL
              4225 ;
              4226 CSFDIR

0F31 EE0713 4227          INC  SFNUM
0F34 18      4228          CLC
0F35 AD0513 4229          LDA  CDIRD    ;CDIRD=CDIRD+ENTRY LEN
0F38 6910    4230          ADC  #DFDELN
0F3A 1011    4231          BPL  SFD2     ;IF RESULT <128 THEN BR
              4232 ; ELSE AT END OF DIR SECT

0F3C EE0613 4233          INC  CDIRS    ;INC TO NEXT DIR SECTOR
0F3F A908    4234          LDA  #8      ;TEST END OF DIR
0F41 CD0613 4235          CMP  CDIRS
0F44 9002    4236          BCC  SFD1     ;BR NOT END
0F46 F048    4237          BEQ  SDRTN
              4238 ;
0F48 206E10 4239 SFD1    JSR  RDDIR    ;READ THE NEXT DIR RECORD
0F4B A900    4240          LDA  #0      ;SET DIR DISPL = 0
              4241 ;
0F4D 8D0513 4242 SFD2    STA  CDIRD    ;SET NEW DIR DISPL
0F50 A8      4243          TAY      ;PUT DISPL IN Y AS INDEX
              4244 ;

```

ATARI DOS 2.0S

DIRECTORY SEARCH

```

0F51 B90114 4245      LDA  FILDIR+DFDFLL,Y ;GET FLAG 1
0F54 F01D  4246      BEQ  SFDSH           ;BR IF UNUSED (END OF USED
                        ENTRIES)
0F56 301B  4247      BMI  SFDSH           ;BR IF DELETED
0F58 2901  4248      AND  #DFDOUT        ;IF OPEN OUTPUT
0F5A D0D5  4249      BNE  CSFDIR         ;DON'T FIND IT
                        4250 ;
                        4251 ; ENTRY IN USE, TEST FOR MATCH
0F5C A200  4252      LDX  #0             ;TEST MATCH ON 12 CHARS
0F5E BD5913 4253 SFD3 LDA  FNAME,X           ;FILE NAME CHAR
0F61 C93F  4254      CMP  #'?           ;IS FNC WILD CARD
0F63 F005  4255      BEQ  SFD4           ;THEN IT MATCHES
0F65 D90614 4256     CMP  FILDIR+DFDFPN,Y ;ELSE IT MUST MATCH FOR
                        REAC
0F68 D0C7  4257      BNE  CSFDIR         ;IF NOT MATCH THEN TRY NEXT
0F6A E8 4258 SFD4 INX                ;INC CHAR CNT
0F6E C8 4259      INY
0F6C E00B  4260      CPX  #11           ;TEST ALL
0F6E D0EE  4261      BNE  SFD3           ;AND CONTINUE CHECK
                        4262 ;
0F70 18 4263      CLC                ;WE HAVE A MATCH
0F71 901D  4264      BCC  SDRTN
                        4265 ;
                        4266 SFD3H
0F73 AD0213 4267     LDA  DHOLES           ;IF DHOLES NOT MINUS
0F76 1012  4268     BPL  SFDSH1        ;THEN ALREADY HAVE A GOOD HOLE
                        4269 ;
                        4270 ; ELSE
                        4271 ;
0F7E AD0613 4272     LDA  CDIRS           ;MOVE CURR DISPL SECTOR
0F7E 8D0213 4273     STA  DHOLES        ;AND CURRENT DIR DISPL
0F7E AD0513 4274     LDA  CDIRD           ;TO HOLE SECTOR AND DISPL
0F81 8D0313 4275     STA  DHOLED
0F84 AD0713 4276     LDA  SFNUM           ;SAVE HOLE
0F87 8D0413 4277     STA  DHFNUM        ;FILE NUMBER
                        4278 ;
0F8A B90114 4279 SFD3H1 LDA FILDIR+DFDFLL,Y ;IF HOLE WAS A DELETED
0F8D 30A2  4280     BMI  CSFDIR         ;ENTRY THEN CONTINUE
                        4281 ;
                        4282 ; ELSE WE ARE AT END OF
                        4283 ;
0F8F 38 4284      SEC                ;USED ENTRIES THUS FILE NOT
                        FOUND
0F90 AE0113 4285 SDRTN LDX  CURFCB        ;RESTORE X REG
0F93 60 4286      RTS

```

WRITE DATA SECTOR

```

0F94 4287      .PAGE "WRITE DATA SECTOR"
                        4288 ;
                        4289 ; WRTNXS - WRITE NEXT SECTOR
                        4290 ;
                        4291 WRTNXS
0F94 BD8513 4292     LDA  FCBFLG,X       ;IF ACQUIRING SECTORS
0F97 300F  4293     BMI  WRTN1        ;THEN NOT UPDATE
                        4294 ;
0F99 0A 4295      ASL  A                ;IF SECTOR NOT MODIFIED
0F9A 1009  4296     BPL  WRU1        ;THEN DON'T IT
                        4297 ;
0F9C 0A 4298      ASL  A
0F9D 9D8513 4299     STA  FCBFLG,X       ;TURN OFF FLAG BITS
0FA2 20F80F 4300     JSR  WRCSIO        ;WRITE CURRENT SECTOR
0FA3 3024  4301     BMI  WRNERR        ;BR IF BAD I/O
0FA5 4CF10 4302 WRU1 JMP  RDNXTS        ;ELSE READ NEXT SECTOR
                        4303 ;

```

ATARI DOS 2.0S

```

0FAB 200611 4304 WRTN1 JSR GETSECTOR ;GET A NEW SECTOR
                    4305 ;
0FAB BD8713 4306 WRTLSEC LDA FCBDLN,X ;GET DATA LEN
0FAE ACFB12 4307 WRTLS1 LDY DRVLBT ;INTO LAST BYTE
0FB1 9147 4308 STA (ZSBA),Y ;OF SECTOR
                    4309 ;
0FB3 BD8C13 4310 WRTN2 LDA FCBLSN+1,X ;MOVE LINK SECTOR
0FB6 1D8113 4311 ORA FCBFNO,X ;PLUS FILE NUM
0FB9 ACF812 4312 LDY DRVMDL ;TO BYTES 126,127
0FBC 9147 4313 STA (ZSBA),Y ;OF SECTOR BUFF
0FBE C8 4314 INY
0FBF BD8B13 4315 LDA FCBLSN,X
0FC2 9147 4316 STA (ZSBA),Y
                    4317 ;
0FC4 20F80F 4318 JSR WRCSIO ;WRITE SECTOR
0FC7 1011 4319 BPL WRTN5 ;BR NOT ERROR
                    4320 ;
0FC9 AD0303 4321 WRNERR LDA DCBSTA ;SAVE ERROR STATUS
0FCC 8D0F13 4322 STA TEMP4
0FCF A900 4323 LDA #0 ;CLOSE FILE
0FD1 9D8213 4324 STA FCBOTC,X
0FD4 AD0F13 4325 LDA TEMP4 ;RECOVER ERROR CODE
0FD7 4CD312 4326 JMP RETURN
                    4327 ;
                    4328 WRTN5
0FDA FE8F13 4329 INC FCBCNT,X ;INC SECTOR CNT
0FD3 D003 4330 BNE WRTN6
0FD7 FE9013 4331 INC FCBCNT+1,X
                    4332 WRTN6
0FE2 200210 4333 JSR MVLSN ;LINK TO CUR
0FE5 A900 4334 LDA #0
0FE7 9D8B13 4335 STA FCBLSN,X ;LINK = 0
0FEA 9D8C13 4336 STA FCBLSN+1,X
0FED 9D8713 4337 STA FCBDLN,X ;DLN = 0

```

WRITE DATA SECTOR

```

0FF0 ADF812 4338 LDA DRVMDL
0FF3 9D8613 4339 STA FCBMLN,X
                    4340 WRNRTS
0FF5 18 4341 CLC
0FF7 60 4342 RTS
                    4343 ;
0FFB 38 4344 WRCSIO SEC ;WRITE CUR SECTOR
0FF9 BD8A13 4345 RWCSIO LDA FCBCSN+1,X
0FFC BC8913 4346 LDY FCBCSN,X
0FF7 4CF711 4347 JMP DSIO
                    4348 ;
1002 BD8B13 4349 MVLSN LDA FCBLSN,X ;MOVE LINK
1005 9D8913 4350 STA FCBCSN,X
1008 BD8C13 4351 LDA FCBLSN+1,X
100B 9D8A13 4352 STA FCBCSN+1,X
100E 60 4353 RTS
                    4354 ;
100F 45 .INCLUDE #E:
100F 50 .INCLUDE #D:ATFMS4.SRC

```

READ DATA SECTOR

```

100F 5000 .PAGE "READ DATA SECTOR"
                    5001 ;
                    5002 ; RDNXTS - READ NEXT SECTOR
                    5003 ;
                    5004 RDNXTS
100F BD8513 5005 LDA FCBFLG,X ;IF NOT UPD MODE
1012 F003 5006 BEQ RDNSO ;BR

```


ATARI DOS 2.0S

```

1014 4C940F 5007      JMP  WRNXXS      ;ELSE WRITE FIRST
1017              5008 RDNSO  =      *
1017 BD8B13 5009      LDA  FCBLSN,X   ;IF LSN NOT
101A 1D8C13 5010      ORA  FCBLSN+1,X ;ZERO
101D D002 5011      BNE  RDNS1    ;BR
101F 38 5012      SEC      ;ELSE EOF
1020 60 5013      RTS
1021 200210 5014 RDNS1 JSR  MVLSN    ;MOVE LINK TO CURRENT
1024 18 5015      CLC      ;READ
1025 20F90F 5016      JSR  RWCSIO   ;CURRENT SECTOR
1028 3035 5017      BMI  RDIOER   ;BR IF OK READ
              5018 ;
              5019 ; ELSE GOTO I/O ERROR
              5020 ;
102A ACF812 5021      LDY  DRVMDL
102D B147 5022      LDA  (ZSBA),Y ;TEST FOR SAME
102F 29FC 5023      AND  #$FC    ;FILE NO
1031 DD8113 5024      CMP  FCBFNO,X
1034 D02C 5025      BNE  RDFNMM   ;IF NOT THEN ERROR
              5026 ;
1036 B147 5027      LDA  (ZSBA),Y ;MOVE LINK SECTOR
1038 2903 5028      AND  #$03
103A 9D8C13 5029      STA  FCBLSN+1,X
103D C8 5030      INY
103E B147 5031      LDA  (ZSBA),Y
1040 9D8B13 5032      STA  FCBLSN,X
              5033 ;
1043 C8 5034      INY      ;INC TO LEN BYTE
1044 B147 5035      LDA  (ZSBA),Y ;GET LEN BYTE
1046 48 5036      PHA      ;SAVE IT
1047 BD8413 5037      LDA  FCBSLT,X ;GET SECTOR LEN TYPE
104A D008 5038      BNE  RDNS3    ;BR IF NEW TYPE
              5039 ;
104C 68 5040      PLA      ;GET LEN
104D 3002 5041      BMI  RDNS2    ;BR IF OLD SHORT SECTOR
104F A97D 5042      LDA  #125   ;ELSE SET FULL SECTOR
1051 297F 5043 RDNS2 AND  #$7F   ;TURN OFF MSB
1053 48 5044      PHA      ;BALANCE STACK
              5045 ;
1054 68 5046 RDNS3 PLA      ;
1055 9D8613 5047      STA  FCBMLN,X ;SET MAX LEN
              5048 ;
1058 A900 5049      LDA  #0     ;SET CUR DATA LEN = 0
105A 9D8713 5050      STA  FCBDLN,X

```

READ DATA SECTOR

```

105D 18 5051      CLC
105E 60 5052      RTS
105F 20E512 5053 RDIOER JSR  ERRIO   ;I/O ERROR
1062 5054 RDFNMM =      *      ;FILE NUMBER MISMATCH
1062 BD4203 5055      LDA  ICCOM,X
1065 C921 5056      CMP  #$21   ;WAS THIS DELETE
1067 F003 5057      BEQ  RDDELE   ;BR IF DELETE
1069 20C712 5058      JSR  ERFNMM   ;BR NOT DELETE
106C 38 5059 RDDELE SEC      ;INDICATE EOF TO DELETE
106D 60 5060      RTS
              5061 ;

```

READ/WRITE DIR

```

106E 5062      .PAGE "READ/WRITE DIR"
              5063 ;
              5064 ; RDDIR/WRDIR READ/WRITE DIRECTORY
              5065 ;
106E 18 5066 RDDIR CLC      ;SET READ

```

ATARI DOS 2.0S

```

106F 9001 5067      BCC DIRIO
                    5068 ;
1071 38    5069 WRTDIR SEC          ;SET WRITE
                    5070 ;
1072 08    5071 DIRIO PHP          ;SAVE READ WRITE
1073 A914  5072 LDA #FILDIR/256 ;MOVE BUF ADDR
1075 8D0503 5073 STA DCBBUF+1 ;TO DCB
1078 A901  5074 LDA #FILDIR&255
107A 8D0403 5075 STA DCBBUF
                    5076 ;
107D 18    5077 CLC
107E AD0613 5078 LDA CDIRS      ;CDIRS+
1081 6969  5079 ADC #$69      ;((40*18)/2)+1
1083 A8    5080 TAY          ;INTO A,Y
1084 A901  5081 LDA #1        ;IS DIR SECTOR NUMBER
1086 6900  5082 ADC #0
                    5083 ;
1088 4CAB10 5084 JMP DSYSIO   ;GO DO SYSTEM I/O
                    5085 ;

```

READ/WRITE VTOC

```

108B      5086      .PAGE "READ/WRITE VTOC"
                    5087 ;
                    5088 ; RDVTOC/WRCTOC - READ/WRITE VTOC
                    5089 ;
                    5090 RDVTOC
108B A005  5091 LDY #DVDWRQ   ;IF WRITE REQD
108D B145  5092 LDA (ZDRVA),Y
108F F001  5093 BEQ RDVGO
1091 60    5094 RTS
1092 18    5095 RDVGO CLC      ;SET READ
1093 9007  5096 BCC VTIO
                    5097 ;
                    5098 WRTVTOC
1095 A005  5099 WRVTOC LDY #DVDWRQ ;TURN OFF
1097 A900  5100 LDA #0      ;WRITE READ
1099 9145  5101 STA (ZDRVA),Y
109B 38    5102 SEC
                    5103 ;
                    5104 ;
109C 08    5105 VTIO PHP      ;SAVE R/W
109D A546  5106 LDA ZDRVA+1 ;MOVE BUF ADDR
109F 8D0503 5107 STA DCBBUF+1 ;TO DCB
10A2 A545  5108 LDA ZDRVA
10A4 8D0403 5109 STA DCBBUF
                    5110 ;
10A7 A068  5111 LDY #$68   ;READ SECTOR
10A9 A901  5112 LDA #1     ;(40*18)/2
                    5113 ;
                    5114 DSYSIO
10AB 28    5115 PLP
                    5116 DSYSIA
10AC AEFE12 5117 LDX DRVTYP  ;LOAD DRIVE TYPE
10AF 206C07 5118 JSR BSIO   ;GO DO I/O
10B2 3001  5119 BMI DSIOER ;BR IF ERROR
10B4 60    5120 RTS      ;RETURN
                    5121 ;
                    5122 ;
10B5 C983  5123 DSIOER CMP #DCBDER ;WAS IT DATA ERROR
10B7 F003  5124 BEQ DEAD  ;BR IF WAS
10B9 4CE512 5125 JMP ERRIO  ;ELSE USER PROBLEM
                    5126 ;
10BC 4CC912 5127 DEAD JMP ERRSYS ;FATAL ERROR
                    5128 ;
                    5129 ; OPEN VTOC
                    5130 ;

```

ATARI DOS 2.0S

```
5131 OPVTOC
10BF 208B10 5132 JSR RDVTOC ;READ IT
10C2 4C9510 5133 JMP WRTVTOC ;THEN WRITE IT
5134 ;
5135 ; INSURES NOT PROTECTED
5136 ;
```

FREE SECTOR

```
10C5 5137 .PAGE "FREE SECTOR"
5138 ;
5139 ; FRESECT - FREE CURRENT SECTOR
5140 ;
5141 FRESECT
10C5 BD8913 5142 LDA FCBCSN,X
10C8 1D8A13 5143 ORA FCBCSN+1,X
10CB F038 5144 BEQ FSRTS
10CD A900 5145 LDA #0
10CF A003 5146 LDY #3 ;DIVIDE SECTOR #
10D1 5E8A13 5147 FS1 LSR FCBCSN+1,X ;BY 3 TO GET BYTE NO
10D4 7E8913 5148 ROR FCBCSN,X ;WITH REM IN ACU
10D7 6A 5149 ROR A
10D8 88 5150 DEY
10D9 D0F6 5151 BNE FS1
5152 ;
10DB A005 5153 LDY #5
10DD 6A 5154 FS2 ROR A ;TO FOR BYT BIT NO
10DE 88 5155 DEY
10DF D0FC 5156 BNE FS2
5157 ;
10E1 A8 5158 TAY ;BIT NO (0-7) INTO Y
10E2 A900 5159 LDA #0
10E4 38 5160 SEC ;SHIFT IN A BIT
10E5 6A 5161 FS3 ROR A ;TO PROPER LOCATION
10E6 88 5162 DEY
10E7 10FC 5163 BPL FS3
10E9 48 5164 PHA ;SAVE MASK
10EA BD8913 5165 LDA FCBCSN,X ;GET BYTE NO
10ED 690A 5166 ADC #DVDSMP ;ADD OFFSET TO SMAP
10EF A8 5167 TAY ;RESULT IS VTOC INDEX
5168 ;
10F0 68 5169 PLA ;GET BIT MASK
10F1 1145 5170 ORA (ZDRVA),Y ;OF BIT TO BIT MAP
10F3 9145 5171 STA (ZDRVA),Y ;AND SET RESULTS
5172 ;
10F5 A003 5173 LDY #DVNSA ;INC NO OF SECTORS AVAIL
10F7 B145 5174 LDA (ZDRVA),Y
10F9 18 5175 CLC
10FA 6901 5176 ADC #1
10FC 9145 5177 STA (ZDRVA),Y
10FE C8 5178 INY
10FF B145 5179 LDA (ZDRVA),Y
1101 6900 5180 ADC #0
1103 9145 5181 STA (ZDRVA),Y
5182 ;
1105 5183 FSRTS = *
1106 60 5184 RTS
5185 ;
```

GET SECTOR

```
11106 5186 .PAGE "GET SECTOR"
5187 ;
5188 ; GET SECTOR - GET A FREE SECTOR FOR
5189 ; USE IN FCB AT X REG. THE SECTOR
5190 ;NUMBER IS PLACED IN FCBLSN
```

ATARI DOS 2.0S

```

5191 ;
5192 ; THE SEARCH FOR A FREE SECTOR STARTS
5193 ; AT THE DVDSMP BYTE. SECTORS ARE
5194 ; NUMBERED SEQUENTIALLY FROM ZERO TO
5195 ; MAXSM WITH THE LEFT BIT OF THE DVDSMP
5196 ; BEING WITH ZERO.
5197 ;
5198 GETSECTOR
1106 A009 5199 LDY #DVDSMP-1 ;SET Y TO START MAP-1
5200 ;
1108 C8 5201 GS1 INY ;INC SMAP INDEX
1109 C064 5202 CPY #90+DVDSMP ;AT END OF MAP?
110B B054 5203 BCS GSEERR ;BR IF AT END
110D B145 5204 LDA (ZDRVA),Y ;GET A MAP BYTE
110F F0F7 5205 BEQ GS1 ;BR NO FREE SECTOR IN BYTE
5206 ;
1111 8C0C13 5207 STY TEMP1 ;SAVE MAP INDEX
1114 48 5208 PHA ;DEC NO OF SECTORS AVAIL
1115 38 5209 SEC
1116 A003 5210 LDY #DVNSA
1118 B145 5211 LDA (ZDRVA),Y
111A E901 5212 SBC #1
111C 9145 5213 STA (ZDRVA),Y
111E C8 5214 INY
111F B145 5215 LDA (ZDRVA),Y
1121 E900 5216 SBC #0
1123 9145 5217 STA (ZDRVA),Y
5218 ;
1125 C8 5219 INY ;SET READ REQD
1126 A9FF 5220 LDA #$FF
1128 9145 5221 STA (ZDRVA),Y
5222 ;
112A 68 5223 PLA
112B A0FF 5224 LDY #$FF ;SET BIT COUNTER ==-1
5225 ;
112D C8 5226 GS2 INY ;SHIFT MAP BYTE
112E 0A 5227 ASL A ;UNTIL A FREE SECTOR
112F 90FC 5228 BCC GS2 ;FOUND
1131 8C0D13 5229 STY TEMP2 ;SAVE BIT NUMBER
1134 4A 5230 LSR A ;AND SHIFT BYTE
1135 88 5231 DEY ;BACKS TO ITS ORIGINAL
1136 10FC 5232 BPL GS3 ;POSITION AND PUT IT
1138 AC0C13 5233 LDY TEMP1 ;BACK INTO THE MAP
113B 9145 5234 STA (ZDRVA),Y
5235 ;
5236 ;

GET SECTOR
113D 38 5237 SEC ;SECTOR NAP BYTE
113E AD0C13 5238 LDA TEMP1 ;=DISPL-DVDSMP
1141 E90A 5239 SBC #DVDSMP
5240 ;
1143 A000 5241 LDY #0
1145 8C0C13 5242 STY TEMP1 ;CLEAR SECT NO HI
5243 ;
1148 0A 5244 GS4 ASL A ;MULT REL SECTOR MAP
1149 2E0C13 5245 ROL TEMP1
114C C8 5246 INY
114D C003 5247 CPY #3
114F 90F7 5248 BCC GS4
5249 ;
1151 18 5250 CLC
1152 6D0D13 5251 ADC TEMP2 ;ADD BIT NO TO
1155 9D8B13 5252 STA FCBLNS,X ;SECTOR #
1158 AD0C13 5253 LDA TEMP1 ;AND PUT INTO
115B 6900 5254 ADC #0 ;FCBLNS

```

ATARI DOS 2.0S

```

115D 9D8C13 5255      STA  FCBLSN+1,X
                    5256 ;
1160 60          5257      RTS
                    5258 ;
1161 4CCB12 5259 GSERR JMP  ERRNSA      ;NO SECTOR AVAIL
                    5260 ;

```

SETUP ROUTINE

```

1164          5261      .PAGE "SETUP ROUTINE"
                    5262 ;
                    5263 ; SETUP - A ROUTINE USED FOR ALL COMMANDS
                    5264 ; TO SET UP FMS CONTROLL CELLS
                    5265 ; TO ACCESS A PARTICULAR FILE.
                    5266 ;
                    5267 SETUP
1164 A99F 5268      LDA  #$9F      ;INIT ERROR CODE
1166 8549 5269      STA  ERRNO      ;TO ZERO
1168 8E0113 5270     STX  CURFCB     ;SAVE FCB
                    5271 ;
116B BA     5272     TSX
116C E8     5273     INX
116D E8     5274     INX
116E 8E0013 5275     STX  ENTSTK
                    5276 ;
1171 AE0113 5277     LDX  CURFCB     ;GET CURRENT FCB
1174 A421 5278     LDY  ICDNOZ     ;MOVE DRIVE NO
1176 8C0103 5279     STY  DCBDRV     ;TO DCB
1179 88     5280     DEY
                    5281 ;
117A B92913 5281     LDA  DBUFAL,Y   ;MOVE WRITE BUFFER
117D 8545 5282     STA  ZDRVA      ;ADD TO ZERO PAGE PTR
117F B93113 5283     LDA  DBUFAH,Y
1182 8546 5284     STA  ZDRVA+1
                    5285 ;
1184 B91113 5286     LDA  DRVTBL,Y   ;GET DRIVE TYPE
1187 F052 5287     BEQ  DERR1      ;BR IF NOT EXISTS
1189 8DFE12 5288     STA  DRV Typ     ;SAVE TYPE
                    5289 ;
118C A8     5290     TAY
                    5291 ;
118D B9F812 5291     LDA  DRVMDL,Y   ;AND LAST SECTOR BYTE
1190 8DF812 5292     STA  DRVMDL     ;DISPL TO LAST OF
1193 B9FB12 5293     LDA  DRVLBT,Y   ;TABLES
1196 8DFB12 5294     STA  DRVLBT
                    5295 ;
1199 BC8813 5296     LDY  FCBBUF,X   ;GET SECTOR BUF #
119C 88     5297     DEY
                    5298 ;
119D 1031 5298     BPL  SSBA      ;BR IF ONE IS ALLOCATED
                    5299 ;
119F A000 5300     LDY  #0
11A1 B91913 5301     LDA  SECTBL,Y   ;TRY TO FIND ONE
11A4 F008 5302     BEQ  GSB4      ;BR ONE FOUND
11A6 C8     5303     GSB2
11A7 C010 5304     CPY  #16
11A9 90F6 5305     BCC  GSB1      ;BR MORE TO TRY
                    5306 ;
11AB 4CCD12 5307     GSB3 JMP  ERRNSB     ;NO SECTOR BUFFERS AVAIL
                    5308 ;
11AE AD FE12 5309     GSB4 LDA  DRV Typ     ;FOUND ONE IF 256 BYTES
11B1 4A     5310     LSR  A
11B2 B010 5311     BCS  GSB5      ;DRIVE NEEDED TO CONT
                    ;BR NOT 256 BYTES

```

SETUP ROUTINE

```

11B4 C8     5312     INY
11B5 C010 5313     CPY  #16
11B7 B0F2 5314     BCS  GSB3
11B9 B91913 5315     LDA  SECTBL,Y   ;ELSE TRY NEXT CONTIG
                    ;TEST END OF BUFFERS
                    ;AND BR IF NO MORE
                    ;ELSE SEE IF ITS THREE

```

ATARI DOS 2.0S

```

11BC D0E8 5316 BNE GSB2 ;BR NOT FREE
11BE 88 5317 DEY
5318 ;
11BF A980 5319 LDA #S80 ;ALLOCATE SECOND OF 2
11C1 991A13 5320 STA SECTBL+1,Y
5321 ;
11C4 A980 5322 GSB5 LDA #S80 ;ALLOCATE FIRST OR ONLY
11C6 991913 5323 STA SECTBL,Y
11C9 98 5324 TYA
11CA 9D8813 5325 STA FCBBUF,X ;PUT BUF NO INTO FCB
11CD FE8813 5326 INC FCBBUF,X ;INC BUF NO SO NOT ZERO
5327 ;
11D0 B93913 5328 SSBA LDA SABUFL,Y ;MOVE BUFFER ADDR
11D3 8547 5329 STA ZSBA ;TO ZERO PAGE PTR
11D5 B94913 5330 LDA SABUFH,Y
11D8 8548 5331 STA ZSBA+1
5332 ;
5333 ;
11DA 60 5334 RTS
5335 ;
11DB 4CCF12 5336 DERR1 JMP ERRDNO ;BAD DRIVE NO

```

SETUP ROUTINE

```

11DE 5337 .PAGE
5338 ;
5339 ; FREE SECTOR BUFFERS
5340 ;
11DE 5341 PRESBUF = *
11DE BC8813 5342 LDY FCBBUF,X ;GET BUF NO
11E1 F013 5343 BEQ FSBR ;BR IF NONE
11E3 88 5344 DEY ;DEC FOR TBL ACCESS
11E4 A900 5345 LDA #0 ;FREE
11E6 9D8813 5346 STA FCBBUF,X ;IN FCR
11E9 991913 5347 STA SECTBL,Y ;AND TABLE
11EC ADFE12 5348 LDA DRVTTY ;IF 128 BYTES
11EF 4A 5349 LSR A ;DRIVE
11F0 B004 5350 BCS FSBR ;FREE ONLY ONE
11F2 4A 5351 LSR A ;ELSE
11F3 991A13 5352 STA SECTBL+1,Y ;FREE 2
11F6 60 5353 FSBR RTS
5354 ;

```

DATA SECTOR I/O

```

11F7 5355 .PAGE "DATA SECTOR I/O"
5356 ;
5357 ; DSIO - DATA SECTOR I/O
5358 ;
5359 DSIO
11F7 48 5360 PHA ;SAVE ACU DATA
11F8 A547 5361 LDA ZSBA ;WRITE SECTOR BUF
11FA 8D0403 5362 STA DCBBUF ;ADR MOVED TO
11FD A548 5363 LDA ZSBA+1 ;DCB
11FF 8D0503 5364 STA DCBBUF+1
1202 68 5365 PLA ;RESTORE ACU
5366 ;
1203 AEF12 5367 LDX DRVTTY
1206 206C07 5368 JSR BSIO ;DO THE I/O
1209 60 5369 RTS
5370 ;

```

WRITE DOS

```

120A 5371 .PAGE "WRITE DOS"
5372 ;

```

ATARI DOS 2.0S

```

5373 ; WRTDOS - WRITE DOS TO DISK
5374 ;
5375 WRTDOS
120A BC8913 5376 LDY FCBCSN,X ;MOVE START ADDR
120D BD8A13 5377 LDA FCBCSN+1,X
1210 205312 5378 JSR SETDSO ;WRITE SECTOR 0
1213 206712 5379 JSR WD0 ;WRITE DOS
1216 4CF012 5380 JMP GREAT
5381 ;
5382 DELDOS
1219 A900 5383 LDA #0 ;SET FILE NOT EXISTS
5384 DD1
121B 8D0E07 5385 STA DFSFLG
5386 ;
5387 WRTSCO
121E A907 5388 LDA #FMSORG/256 ;MOVE FMS START
1220 8D0503 5389 STA DCBBUF+1 ;ADDR TO DCB
1223 A900 5390 LDA #FMSORG&255
1225 8D0403 5391 STA DCBBUF
5392 ;
1228 A900 5393 LDA #0 ;CLEAR SECTOR NO TO ZERO
122A 8D0A03 5394 STA DCBSEC
122D 8D0B03 5395 STA DCBSEC+1
5396 ;
1230 EE0A03 5397 WRNBS INC DCBSEC ;INC SECTOR NO
1233 A201 5398 LDX #1 ;GET DRIVE TYPE
1235 38 5399 SEC
1236 207207 5400 JSR BSIOR ;DO THE WRITE
5401 ;
5402 ;
1239 18 5403 CLC
123A AD0403 5404 LDA DCBBUF ;INC SECT ADDR
123D 6980 5405 ADC #128
123F 8D0403 5406 STA DCBBUF
1242 AD0503 5407 LDA DCBBUF+1
1245 6900 5408 ADC #0
1247 8D0503 5409 STA DCBBUF+1
5410 ;
124A AD0A03 5411 LDA DCBSEC ;TEST FOR WRITE
124D CD0107 5412 CMP BRCNT ;OF ALL BOOT SECTORS
1250 D0DE 5413 BNE WRNBS ;BR NOT ALL
5414 ;
1252 60 5415 RTS
5416 ;
1253 8C0F07 5417 SETDSO STY DFLINK ;SET LINK START
1255 8D1007 5418 STA DFLINK+1
1259 ADFE12 5419 LDA DRV Typ
125C 8D0E07 5420 STA DFSFLG
125F ACF812 5421 LDY DRVM DL

WRITE DOS

1262 8C1107 5422 STY BLDISP
1265 D0B4 5423 BNE DD1 ;GO WRITE SECTOR 0
5424 ;

WRITE DOS

1267 5425 .PAGE "
1267 AD1207 5426 WD0 LDA DFLADR ;MOVE FILE START ADDR
126A 8543 5427 STA ZBUFF ;TO ZBUFF
126C AD1307 5428 LDA DFLADR+1
126F 8544 5429 STA ZBUFF+1
5430 ;
1271 A000 5431 WD1 LDY #0 ;MOVE 125
1273 B143 5432 WD2 LDA (ZBUFF),Y ;BYTES OF DOS

```

ATARI DOS 2.0S

```

1275 9147 5433 STA (ZSBA),Y ;TO SECTOR BUFFER
1277 C8 5434 INY
1278 CCF812 5435 CPY DRVMDL
127B 90F6 5436 BCC WD2
127D 98 5437 TYA
127E 9D8713 5438 STA FCBDLN,X ;SET DATA LEN
5439 ;
1281 205707 5440 JSR INCBA ;INC ZBUFF BY 125
1284 CD0D07 5441 CMP SASA+1 ;IF NOT END OR
1287 900B 5442 BCC WD3 ;PAST END OF DOS
1289 D00F 5443 BNE WD4 ;THEN WRTNXS
128B A543 5444 LDA ZBUFF ;ELSE
128D CD0C07 5445 CMP SASA ; DONE
1290 9002 5446 BCC WD3
1292 D006 5447 BNE WD4
5448 ;
1294 20940F 5449 WD3 JSR WRTNXS ;WRITE NEXT SECTOR
1297 4C7112 5450 JMP WD1
5451 ;
129A 60 5452 WD4 RTS ;RETURN, CLOSE WILL WRITE
FINAL SECTOR
5453 ; AND RETURN
5454 ;

```

TEST DOS FILE NAME

```

129B 5455 .PAGE "TEST DOS FILE NAME"
5456 ;
5457 ; TSTDOS - TEST FOR DOS SYS FILE NAME;
5458 ;
5459 TSTDOS
129B A00B 5460 LDY #11 ;LOOK AT 12 CHARS
129D B95813 5461 TDF1 LDA FNAME-1,Y ;TEST DECODE FILENAME CHAR
12A0 D9A812 5462 CMP DFN-1,Y ;WITH DOS FILENAME CHAR
12A3 D003 5463 BNE TDFR ;BR NOT MATCH
12A5 88 5464 DEY
12A6 D0F5 5465 BNE TDF1 ;BR IF MORE, ELSE RTN EQ
12A8 60 5466 TDFR RTS
5467 ;
12A9 44 5468 DFN .BYTE "DOS SYS "
12AA 4F
12AB 53
12AC 20
12AD 20
12AE 20
12AF 20
12B0 20
12B1 53
12B2 59
12B3 53
12B4 20
5469 ;
5470 ; ERROR ROUTINES
5471 ;
12B5 E649 5472 ERDBAD INC ERRNO ;BAD SECTOR AT FORMAT TIME
12B7 E649 5473 ERAPO INC ERRNO ;ATTEMPT APPEND TO OLD TYPE
FILE
12B9 E649 5474 ERRPOT INC ERRNO ;POINT INVALID
12BB E649 5475 ERFNF INC ERRNO ;FILE NOT FOUND
12BD E649 5476 ERDFULL INC ERRNO ;DIRECTORY FULL
12BF E649 5477 ERDVDC INC ERRNO ;DEVICE COMMAND INVALID
12C1 E649 5478 ERFLOCK INC ERRNO ;FILE LOCKED
12C3 E649 5479 ERFPDL INC ERRNO ;POINT DATA LENGTH
12C5 E649 5480 ERFNF INC ERRNO ;FILE NAME ERROR
12C7 E649 5481 ERFNMM INC ERRNO ;FILE NUMBER MISMATCH
12C9 E649 5482 ERRSYS INC ERRNO ;FATAL SYS DATA I/O ERROR

```


ATARI DOS 2.0S

```

12CB E649 5483 ERRNSA INC ERRNO ;NO SECTOR AVAIL
12CD E649 5484 ERRNSB INC ERRNO ;NO SECTOR BUFFERS AVAIL
12CF E649 5485 ERRDNO INC ERRNO ;DRIVE NO ERROR
      5486 ;
12D1 A549 5487 LDA ERRNO ;GET ERROR NUMBER
12D3 AE0113 5488 RETURN LDX CURFCB ;GET CUR FCB NO
12D6 9D4303 5489 STA ICSTA,X ;PUT IN FCB
12D9 AE0013 5490 LDX ENTSTK ;GET ENTRY STACK PTR
12DC 9A 5491 TXS ;AND RESTORE
12DD AE0113 5492 LDX CURFCB
12E0 A8 5493 TAY
12E1 AD0B13 5494 LDA SVDBYT ;GET SAVED DATA BYTE

```

TEST DOS FILE NAME

```

12E4 60 5495 RTS
      5496 ;
12E5 AD0303 5497 ERRIO LDA DCBSTA ;GET I/O ERROR CODE
12E8 30E9 5498 BMI RETURN
      5499 ;
12EA AE0113 5500 FGREAT LDX CURFCB
12ED 20DE11 5501 JSR FRESBUF ;FREE SECTOR BUFFER
12F0 A901 5502 GREAT LDA #01 ;SET ALL OK
12F2 D0DF 5503 BNE RETURN
12F4 A988 5504 ERREOF LDA #$88 ;SET EOF CODE
12F6 30DB 5505 BMI RETURN
      5506 ;

```

MISC STORAGE

```

12F8 5507 .PAGE "MISC STORAGE"
      5508 ;
      5509 ; MISC NON ZERO PAGE STORAGE AREA
      5510 ;
12F8 00 5511 DRVMDL .BYTE 0 ;MAX DATA LEN
12F9 7D 5512 .BYTE 125 ;128 BYTE SECTOR
12FA FD 5513 .BYTE 253 ;256 BYTE SECTOR
      5514 ;
12FB 00 5515 DRVLBT .BYTE 0 ;DISPL TO LAST SECTOR BYTE
12FC 7F 5516 .BYTE 127 ;128 BYTE SECTOR
12FD FF 5517 .BYTE 255 ;256 BYTE SECTOR
12FE 5518 DRVTYP *= +1 ;DRIVE TYPE
12FF 5519 RETRY *= +1 ;I/O RETRY COUNTER
1300 5520 ENTSTK *= +1 ;ENTRY STACK LEVEL
1301 5521 CURFCB *= +1 ;CURRENT FCB (IOCB ALSO)
1302 5522 DHOLES *= +1 ;DIR HOLE SECTOR
1303 5523 DHOLED *= +1 ;DIR HOLE DISPL
1304 5524 DHFNUM *= +1 DIR HOLE FILE NO
1305 5525 CDIRD *= +1 ;CURRENT DIR DISPL
1306 5526 CDIRS *= +1 ;CURRENT DIR SECTOR
1307 5527 SFNUM *= +1 ;FILE NUMBER
1308 5528 SVDBYT *= +1 ;SAVED OUTPUT DATA BYTE
1309 5529 SVD1 *= +1 ;SAVE DATA BYTES
130A 5530 SVD2 *= +1 ;FOR WRITE BURST
130B 5531 SVD3 *= +1
      5532 EXTSW
130C 5533 TEMP1 *= +1 ;TEMP1
130D 5534 TEMP2 *= +1 ;TEMP2
130E 5535 TEMP3 *= +1 ;TEMP3
130F 5536 TEMP4 *= +1 ;TEMP4
1310 5537 BURTPY *= +1 ;BURST I/O TYPE
      5538 ;
1311 5539 DRVTBL *= +8 ;DRIVE TABLE
1319 5540 SECTBL *= +16 ;
1329 5541 DBUFAL *= +8 ;VTOC BUFFER
1331 5542 DBUFAH *= +8 ;PTR FOR DRIVE N

```

ATARI DOS 2.0S

```

1339      5543 SABUFL *=    *+16      ;SECTOR BUFFER
1349      5544 SABUFH *=    *+16      ;FOR SECTOR N
1359      5545 FNAME  *=    *+12      ;FILE NAME
1365      5546 AFNAME *=    *+12      ;AUXILLARY FILE NAME
        5547 ;
1371      5548 MDRV   *=    *+1        ;MAX DR NO
        5549 ;
1372      5550 Z     =     *          ;PUT ON SAME BOUNDRY AS
        ;                               PRODUCTION
1372      5551      *=    $1381      ;VERSION
    
```

FILE CONTROL BLOCKS

```

1381      5552      .PAGE "FILE CONTROL BLOCKS"
        5553 ;
        5554 ; FILE CONTROL BLOCK
        5555 ; ONE FILE CONTROL BLOCK IS USED FOR EACH
        5556 ; OPEN FILE. THE RELATIVE FCB USED
        5557 ; RELATES DIRECTLY TO THE IOCB #
        5558 ; THAT OPENED THE FILE. THUS THERE ARE
        5559 ; 8 FCBS. THE FCB ARE (CONVIENTLY)
        5560 ; THE SAME SIZE AS IOCBS. EACH FCB
        5561 ; CONTAINS ALL THE INFORMATION REQUIRED
        5562 ; TO CONTROL THE PROCESSING ON AN
        5563 ; OPEN FILE
        5564 ;
        5565 FCB
1381      5566 FCBFNO *=    *+1        ;FILE # LEFT JUSTIFIED
1382      5567 FCBOTC *=    *+1        ;OPEN TYPE CODE
1383      5568      *=    *+1        ;SPARE
1384      5569 FCBSLT *=    *+1        ;FLAG FOR NEW SECTOR LEN TYPE
1385      5570 FCBFLG *=    *+1        ;WORKING FLAG
1386      5571 FCBMLN *=    *+1        ;MAX SECTOR DATA LEN
1387      5572 FCBDLN *=    *+1        ;CUR SECTOR BUF DATA LEN
1388      5573 FCBBUF *=    *+1        ;SECTOR BUF NO
1389      5574 FCBCSN *=    *+2        ;CUR SECTOR #
138B      5575 FCBLSN *=    *+2        ;LINK/ALLOCATE SECTOR #
138D      5576 FCBSSN *=    *+2        ;CUR FILE RELATIVE SECTOR #
        5577 FCBCRS
138F      5578 FCBCNT *=    *+2        ;SECTOR COUNT
0010     5579 FCBLEN =    *-FCB      ;FCB LEN
        5580 ;
1391      5581      *=    FCBLEN*7+* ;ALLOCATE 7 MORE FCBS
        5582 ;
        5583 ; OPEN CODE BITS
        5584 ; USED IN IOCB AUX1
        5585 ; - AND FCBOTC
        5586 ;
0004     5587 OPIN  =    $04        ;INPUT
0008     5588 OPOUT =    $08        ;OUTPUT
0002     5589 OPDIR =    $02        ;LIST DIRECTORY
0001     5590 OPAPND =    $01        ;APPEND
        5591 ;
0080     5592 FCBFAS =    $80        ;FCBFLG - ACQ SECTORS
0040     5593 FCBFSM =    $40        ;FCBFLG - SECTOR MODIFIED
    
```

FILE DIRECTORY

```

1401      5594      .PAGE "FILE DIRECTORY"
        5595 ;
        5596 ; DISK FILE DIRECTORY
        5597 ; THE FILE DIRECTORY OCCUPIES 8
        5598 ; CONSECTUIVE SECTORS STARTING AT THE
        5599 ; CENTRAL SECTOR+1. EACH FILE DIRECTORY
        5600 ; SECTOR CONTAINS 8 ENTRIES. THERE
        5601 ; IS 1 ENTRY FOR EACH NAMED FILE. THE
    
```

ATARI DOS 2.0S

```
5602 ; THERE ARE A TOTAL OF 64 NAMED FILES
5603 ; PER VOLUME
5604 ;
5605 ; THE FILE NUMBER IS USED THROUGH THE
5606 ; THE SYSTEM IS THE RELATIVE (TO ONE)
5607 ; FILE DIRECTORY ENTRY NUMBER.
5608 ;
5609 ; THE EQUATES BELOW ARE FOR A SINCE NAMED
5610 ; FILE ENTRY
5611 ;
0000 5612 DFDFL1 = 0 ;FLAG1 (1)
0001 5613 DFDCNT = 1 ;SECTOR COUNTER (LOW)
0003 5614 DFDCSSN = 3 ;START SECTOR NO (2)
0005 5615 DFDPFN = 5 ;PRIMARY FILE NAME (8)
000D 5616 DFDFXFN = 13 ;EXTENDED FILE NAME (4)
0010 5617 DFDELN = 16 ;ENTRY LENGTH
5618 ;
5619 ; DFDFL1 VALUE EQUATES
5620 ;
0000 5621 DFDEUU = 0 ;ENTRY UNUSED
0080 5622 DFDEDE = $80 ;ENTRY DELETED
0040 5623 DFIDINU = $40 ;ENTRY IN USE
0001 5624 DFDOUO = $01 ;FILE OPEN FOR OUTPUT
0020 5625 DFDFLOC = $20 ;ENTRY LOCKED
0002 5626 DFDFNLD = $02 ;FILE HAS NEW TYPE SECTOR LEN
; BYTE
5627 ;
1401 5628 FILDIR *= *+256 ;RESUME FILE DIR SPACE
5629 ;
```

VOLUME DIRECTORY

```
1501 5630 .PAGE "VOLUME DIRECTORY"
5631 ;
5632 ; DISK VOLUME DIRECTORY
5633 ; THE VOLUME DIRECTORY OCCUPIES THE CENTRAL
5634 ; VOLUME SECTOR. THE VOLUME DIRECTORY
5635 ; CONTAINS INFORMATION PERTAINING TO
5636 ; THE ENTIRE DISKETTE VOLUME.
5637 ;
5638 ; THE LABELS BELOW, MAP THE VOLUME
5639 ; DIRECTORY SECTOR.
5640 ;
0000 5641 DVDTCD = 0 ;VOLUME DIRECTORY TYEP CODE )1)
5642 ;
5643 ; USED TO DELINATE MAJOR (1)
5644 ; FMS SYSTEM FORMAT CHANGES
5645 ;
0001 5646 DVDMNSN = 1 ;MAX SECTOR NUMBER (1)
0003 5647 DVDNSA = 3 ;NO SECTORS AVAIL
5648 ;
0005 5649 DVDWRQ = 5 ;WRITE REQUIRED
000A 5650 DVDSMP = 10 ;SECTOR MAP START
5651 ;
5652 ; EACH BIT REPRESENTS A SECTOR
5653 ; IF THE BIT IS ON THEN THE SECTOR
5654 ; IS FREE AND AVAILABLE. IF THE
5655 ; BIT IS OFF, THE SECTOR IS IN
5656 ; USE OR BAD. THE MOST SIGNIFICANT
5657 ; BIT OF THE FIRST BYTE IS SECTOR ZERO.
```

END OF FMS

```
1501 5658 .PAGE "END OF FMS"
5659 ;
1501 5660 ENDFMS = *
1501 60 .END
```

ATARI DOS 2.0S

END OF FMS

=0700 FMSORG	=0043 FMSZPG	=0340 IOCBORG	=0003 LMASK
=0300 DCBORG	=E453 DHADR	=009B EOL	=031A DEVTAB
=0020 ZICB	=02E7 LMADR	=1540 DUPINIT	=0102 STAK
=00DF OSBTM	=0246 DSKTIM	=000F TIMOUT	0340 IOCB
0340 ICHID	0341 ICDNO	0342 ICCOM	0343 ICSTA
0344 ICBAL	0345 ICBAH	0346 ICPUT	0348 ICBLL
0349 ICBLLH	034A ICAUX1	034B ICAUX2	034C ICAUX3
034D ICAUX4	034E ICAUX5	034F ICAUX6	=0010 ICLEN
=0001 ICOIN	=0002 ICOOUT	=0003 ICIO	=0004 ICGBR
=0005 ICGTR	=0006 ICGBC	=0007 ICGTC	=0008 ICPBR
=0009 ICPTR	=000A ICPBC	=000B ICPTC	=000C ICCLOSE
=000D ICSTAT	=000E ICDDC	=000E ICMAX	=000F ICFREE
=0001 ICOK	=0002 ICSTR	=0003 ICSEOF	=0080 ICSBRK
=0081 ICSDNR	=0082 ICSDNER	=0083 ICSDER	=0084 ICSIVC
=0085 ICSDNOP	=0086 ICSDIVN	=0087 ICSDWPC	=0021 ICDNOZ
=0028 ICBLLZ	=0029 ICBLLHZ	=0024 ICBALZ	=0025 ICBHAZ
=0022 ICCOMZ	=0026 ICPUTZ	0300 DCB	0300 DCBSBI
0301 DCBDRV	0302 DCBCMD	0303 DCBSTA	0304 DCBBUF
0306 DCBTO	0308 DCBCNT	030A DCBSEC	=0052 DCBCRS
=0050 DCBCWS	=0053 DCBCST	=0021 DCBCFD	=0001 DCBSOK
=0081 DCBDNR	=0082 DCBCNR	=0083 DCBDR	=0084 DCBIVC
=0087 DCBWPR	0043 ZBUF	0045 ZDRVA	0047 ZSBA
0049 ERRNO	0700 BFLG	0701 BRCNT	0702 BLDADR
0704 BINTADR	0706 BCNT	0714 XBCONT	0709 SABYTE
070A DRVBYT	070B SAFBFW	070C SASA	=1501 ENDFMS
070E DFDFLG	070F DFLINK	0711 BLDISP	0712 DFLDR
070C DFMSDH	074F BFAIL	072F XBC1	=076C BSIO
0753 BGOOD	0757 INCBA	0754 XBRTN	0772 BSIOR
077C DSIO1	0786 DSIO2	12FF RETRY	079C DSIO3
07A2 DSIO4	07C4 DSIO5	07BE STRTYP	1301 CURFCB
08AB DFMOPN	0B15 DFMCLS	=0ABF DFMGET	09CC DFMPT
0B01 DFMSTA	0BA7 DFMDDC	=07E0 DINIT	130C TEMP1
07F2 DIA	130D TEMP2	0807 DIHAVE	1311 DRVFTBL
1329 DBUFAL	1331 DBUFAH	083D DIDDEC	=0005 DVDWRQ
0823 DI256	0870 DINCBP	0845 DIXNXTS	084B DISETS
1319 SECTBL	085E DISNI	1339 SABUFL	1349 SABUFH
=087E CLRFCB	0882 CFCBX	1381 FCB	088A ADI1
089B ADI2	1164 SETUP	0E9E FNDCODE	1382 FCBOFC
=0002 OPDIR	088E OPN1	0DAD LISTDIR	0F21 SFDIR
=0004 OPIN	=08D8 DFOIN	=0008 OPOUT	=0911 DFOOUT
=08DD DFROUPD	=0001 OPAPND	=08EC DFOAPN	12BF ERDVC
08E9 OPNER1	=08E3 DFOUI	0CAC TSTLOCK	09AE FDRDSU
12F0 GREAT	12BB ERNF	1305 CDIRD	1401 FILDIR
=0000 DFDFL1	=0002 DFDNLD	090E APOER	10BF OPVTOC
1106 GETSECTOR	138D FCBSLN	138B FCBSLN	=097C DHFOX2
12B7 ERAPO	=091D DFOX1	0C53 XDEL0	=0948 OPN1A
1302 DHOLES	0992 OPNER2	1306 CDIRS	106E RDDIR
1303 DHOLED	1304 DHFNUM	1307 SFNUM	093E OPN1B
=0005 DFDFPN	=0003 DFDSSN	=0040 DFDINU	=0001 DFDOUT
=0001 DFDCNT	0966 OPN2	1359 FNAME	=0970 OPN2A
1071 WRTDIR	=0995 SETFCB	0FE2 WRTN6	0982 OPN3
=0080 FCBFAS	1385 FCBFLG	129B TSTDOS	=098F DHFOX3
120A WRTDOS	12BD ERDFULL	099A OPNF1	1381 FCBFNO
1387 FCBDLN	138F FCBCNT	1384 FCBSLT	=1017 RDNSO
1308 SVDBYT	1300 ENTSTK	09E5 FRMCIO	0A19 PUTER
1386 FCBMLN	0A06 PUT1	0F94 WRTNXS	0A1C PEOF
0A1F WTBUR	=0040 FCBFSM	12F4 ERROF	0A4A NOBURST
0A28 TBURST	0A26 RTBUR	1310 BURTTY	=0AAE TBLEN
0A3E NXTBUR	0A4C WRBUR	100F RDNXTS	0A7B BBINC
=0A9D BUREOF	12F8 DRVMDL	1309 SVD1	130A SVD2
130B SVD3	1388 FCBBUF	11D0 SSB	0AAC BURST
12FE DRVTYP	0AB9 TBL256	0ACC GET1	0DB9 GDCHAR
0ADF GET2	=0ADC GEOF	=0AEA EFLOOK	0AFE GET3
12D3 RETURN	0B12 SFNF	0B6D CLDONE	=0B75 CLUPDT
0FAB WRTLSEC	=0B80 RRDIR	0B50 CLOUT	0B3C APP1

ATARI DOS 2.0S

0FB3 WRTN2	1095 WRTVTOC	12EA FGREAT	0FF8 WRCSIO
0B9B FNSHFT	0B9D FNSHF1	0B9F FNSHF2	0BD6 XFV
=0027 MAXDDC	0BD3 DVDCER	0BC5 DVDCVT	0BD9 XRENAME
0C32 XDELETE	0C7C XLOCK	0C83 XUNLOCK	0CBA XPOINT
0D03 XNOTE	0D18 XFORMAT	0BE7 XRN1	0BF2 XRN1A
1219 DELDOS	0EB4 FNDCNX	0C0C XRN1B	1253 SETDSO
0C11 XRN2	0C1B XRN3	0F31 CSFDIR	0C79 DFNF
=0C3A XDELX	0C45 XDELY	0C45 XDEL3	0C56 XDEL1
=0080 DFDEDE	=0C6C XDEL2A	0C67 XDEL2	=0C72 XDEL4
10C5 FRESECT	=0020 DFDLOC	130F TEMP4	0C88 XLCOM
0C93 XLC1	0CB7 TLF	12C1 ERFLOCK	0D00 PERR1
1389 FCBCSN	0CCF XP1	0CED XP2	=0CDC XP1A
=0CF7 XPERR	0CFA XP3	12C3 ERRPDL	12B9 ERRPOT
0D52 XF0	0D4F XFERR	=0D3D TSTFMT	0D4C XFBAD
12B5 ERBAD	0D55 XF1	=000A DVDSMP	0D76 XF2
0D94 XF3	0D9F XF4	0DE3 LDENT1	0DE9 LDCNT
0E11 LDDONE	0DD6 GDCRTN	0DD9 LDENT	0E21 FDBENT
108B RDVTOC	=0003 DVDNSA	0E57 CVDX	=000D FSCML
0DFD MVFSCM	0E14 FSCM	0E67 CVDY	0E35 LD1
0E3B LD2	0E71 CVDIGIT	0E8D STDIGIT	130E TEMP3
0E76 CVD1	0EAA FD0A	0F07 FNDERR	0EB3 FDB0B
0EB8 FD0	130C EXTSW	0EC3 FD1	0ED5 FD3
0ECA FD2	0F0A FDSCHAR	0F03 FDEND	0EE5 FD4
0EFD FD6	0EF1 FD5	12C5 ERRFN	0F1B FDS02
0F15 FDSC1	=0010 DFDELN	0F4D SFD2	0F48 SFD1
0F90 SDRTN	0F73 SFDSH	0F5E SFD3	0F6A SFD4
0F8A SFDSH1	0FAB WRTN1	0FA5 WRU1	0FC9 WRNERR
0FAE WRTLS1	12FB DRVLBT	0FDA WRTN5	1002 MVLSN
0FF6 WRNRTS	0FF9 RWCSIO	11F7 DSIO	1021 RDNS1
105F RDIOER	=1062 RDFNMM	1054 RDNS3	1051 RDNS2
12E5 ERRIO	106C RDDELE	12C7 ERFNMM	1072 DIRIO
10AB DSYSIO	1092 RDVGO	109C VTIO	1095 WRVTOC
10AC DSYSIA	10B5 DSIOER	10BC DEAD	12C9 ERRSYS
=1105 FSRTS	10D1 FS1	10DD FS2	10E5 FS3
1108 GS1	1161 GSERR	112D GS2	1134 GS3
1148 GS4	12CB ERRNSA	11DB DERR1	11A1 GSB1
11AE GSB4	11A6 GSB2	11AB GSB3	12CD ERRNSB
11C4 GSB5	12CF ERRDNO	=11DE FRESBUF	11F6 FSBR
1267 WD0	121B DD1	121E WRTSCO	1230 WRNBS
1271 WD1	1273 WD2	1294 WD3	129A WD4
129D TDF1	12A9 DFN	12A8 TDFR	1365 AFNAME
1371 MDRV	=1372 Z	138F FCBCRS	=0010 FCBLN
=000D DFDXFN	=0000 DFDEUU	=0000 DVDTCD	=0001 DVDSMN



Appendix A

AN INTERMEDIATE USER'S GUIDE TO THIS BOOK

If you are familiar with machine language, commented source code, and hexadecimal numbers, you probably won't need to read this appendix. On the other hand, if you don't know or are new to machine language – perhaps some of the information here will help.

A knowledge of machine language is important to grasping the sense of the DOS since it is written in machine language. However, we will briefly cover some of the fundamentals, as they relate to the book, in the hope that this might be a starting point. One of the functions of this book is to reveal the inner workings of Atari DOS. A benefit of knowing how it works is that you are able to change it to suit yourself, to customize it.

First we'll examine the meaning of the various fields of information which are in the source code (page 59 on). Then, after a brief look at how to deal with hexadecimal numbers, we can make a modification to DOS step-by-step to show how it's done.

The book is divided into two sections: roughly the first half is a series of descriptions of the major subroutines of the disk operating system. The latter half is a *commented source code* of the DOS. In order to better understand what you can accomplish with all this information, we can set up a problem and solve it using the book.

What's "Commented Source Code"?

We'll change the DOS so that we could type in a disk command using lowercase letters. Unfortunately, the D: must be in uppercase, the program which makes this decision is in ROM and we can't get at it and change it. The rest of the command can be in lowercase, though, after we make our change to the DOS in RAM. After fixing it, any routine that uses the disk will accept lowercase as in D: open.

Before getting into the details of the modification there is some important preliminary information. What, for example, is “commented source code?”

Machine language differs in several respects from BASIC. When you write a program in BASIC, you never see how it looks to the computer. Instead you see something like this:

```
10 FOR I= 1 TO 100
20 NEXT I
```

This delay loop just creates a brief pause in a program. If you RUN the above, the computer handles the problem of translating the BASIC words into machine language. Anything the computer does must be translated into machine language (ML). Translating (or *interpreting*) a BASIC program takes place *during* the RUN of the program – that’s why BASIC is so slow compared to ML.

By contrast, ML is translated *before* it is RUN. Programming ML is done in two stages: 1. writing the source code and then 2. assembling it into *object* code. The computer does most of the drudgery of this because most ML is written by using a program called an assembler which handles many of the details. Some assemblers are so complex that using them can seem almost like programming in BASIC.

Here is how you might program the above example delay loop when using an assembler:

```
1000          LDY #64      ; SET COUNTER TO 100
1001 LOOP     DEY
1002          BNE LOOP
```

Probably the most peculiar thing about this, to the beginner, is how 64 stands for 100 (it’s hex, we’ll get to it in a minute). The line numbers could be BASIC, but the instructions are 6502 mnemonics (memory aids). LDY means to load the Y register with 100 (decimal). The next line is named (labelled) “loop” because assemblers don’t say GOTO 1001. Instead, they use convenient names. In any event, the Y register is decremented by DEY, it’s lowered by one. So each time the program cycles through the LOOP address, it will lower the counter one. Finally, the instruction at 1002 says, Branch if Not Equal (to zero). In other words, GOTO LOOP if Y hasn’t yet counted down to zero. When Y reaches zero, the program will continue on, following whatever instruction is in line 1003.

After the above program is written, though, it still cannot be RUN. There is the second step, the creation of object code (executable), the assembly process.

You tell the assembler to assemble this program. The result of

APPENDIX A

that is an additional two "fields" (zones). Above, we have five fields: line number, label, mnemonic (instruction), operand (the #64), and a comment field which is the equivalent of BASIC REM statements. There will soon be a total of seven fields.

After assembly, the two new fields are the addresses and the object code (expressed as hex bytes). By the way, BASIC always assigns its programs a starting address in memory, but, in ML, the programmer must make this known to the assembler. It's not the computer's decision. Assume the computer were told to assemble the above example at address \$2000 (this would be 8192, in decimal). The dollar sign means that a number is a hex number. The labels, mnemonics, and operands would be translated into object code and put into the computer's memory. As you'll see in the second half of this book, a printout of completed assembly looks like this:

```
2000 A000 1000          LDY #64 ;SET COUNTER TO 100
2002 88      1001 LOOP  DEY
2003 D0FF 1002          BNE LOOP
```

Hex

Before concluding this brief overview of some fundamentals of machine language, we should explain how to read the numbers in the source code listings.

```
100 DIM H$(23),N$(9):OPEN#1,4,0,"K:"
130 GRAPHICS 0
140 PRINT "PLEASE CHOOSE:
150 PRINT "1 - Input HEX & get decimal back
."
160 PRINT "2 - Input DECIMAL to get hex bac
k."
170 PRINT:PRINT "=>";:GET#1,K
180 IF K<49 OR K>50 THEN 170
190 PRINT CHR$(K):ON K-48 GOTO 300,400
300 H$="@ABCDEFGHI!!!!!!JKLMNO"
310 PRINT "HEX";:INPUT N$:N=0
320 FOR I=1 TO LEN(N$)
330 N=N*16+ASC(H$(ASC(N$(I))-47))-64:NEXT I
350 PRINT "$";N$;"=";N:PRINT:PRINT:GOTO 140
400 H$="0123456789ABCDEF"
410 PRINT "DECIMAL";:INPUT N:M=4096
420 PRINT N;"=$";
```

```
430 FOR I=1 TO 4:J=INT(N/M)
440 PRINT H$(J+1,J+1);:N=N-M*J:M=M/16
450 NEXT I:PRINT:PRINT:GOTO 140
```

This program will turn a decimal number into hex or vice versa. Hexadecimal is a base 16 number system, where decimal is base ten. This means that you count from zero to fifteen before going to the next column. For example, you count up zero one two...until you reach nine in decimal. Then you go to the next column and have a one-zero (10) to show that there is one in the "ten's column" and zero in the "one's column."

In hex, what was a "ten's column" becomes a "sixteen's column." In other words, the symbol "10" means that there is one sixteen and zero "ones." So, the decimal number 17 would be written in hex, as \$11 (one sixteen plus one one). The decimal number 15 would, in hex, be \$0F. After nine, we run out of digits, so the first few letters of the alphabet are used: A = 10, B = 11, C = 12, D = 13, E = 14, and F = 15.

This explains how to "read" hex numbers if you don't want the program above to do it for you. The number \$64 is decimal 100 because there are six 16's and four one's. $6 \times 16 + 4 = 100$.

Addresses can be larger than two digits, up to a maximum of four. You might see an address such as \$11F7 in the listings. The third column is the 256's and the fourth column is the 4096's. So to find out what this address is in decimal, you can multiply 7×1 , 15×16 , 1×256 , and 1×4096 . And add them all together.

A quicker way is to find out the first two, ($15 \times 16 + 7 = 247$) and then multiply the second two by 256. It comes out the same. The second two would be \$11 (17 in decimal) so $17 \times 256 + 247 = 4599$. It might be easier to just use the BASIC program to make the translations until hex becomes more familiar.

Making A Modification

Now that you have the entire source listing of DOS 2.0S, you can customize it to fit your needs.

You may have felt restricted by the limitations on file names. A file name can consist of eleven characters: up to eight characters plus an optional three-character extension. The first character must be from A-Z; subsequent characters can be from A-Z or 0-9. That's it. No punctuation. No imbedded spaces. No lowercase.

By changing only two locations in the file name decode section of DOS, many more characters are permitted. We will modify DOS to

APPENDIX A

accept any ASCII characters in a file name except character graphics and inverse video. Additionally, the filename can start with a number (e.g. "D:3-D"). Unfortunately, there is no foolproof way to allow imbedded spaces such as "D:TIME OUT".

The following fragment of code checks to see that a character of the file name falls in the range of A-Z. If the character is less than (carry clear) 65 [ASC("A")] or greater than or equal to (carry set) 91 [ASC("Z") + 1], then the test fails. All we do is change the check for "A" to a check for "!" (its number in the code is one greater than "space"), and the check for "Z" + 1 to "z" + 1 (lowercase z).

Included in this range of 90 characters are the numbers (48-57) and all punctuation. Since we start with 33, "space" is excluded. It is possible to permit imbedded spaces, but the file would then be inaccessible in certain situations where a space is used as a delimiter. You can allow it at your discretion, or even permit the entire (almost) ATASCII character set to be used by changing the limits to 0 and 255.

```
CMP  #'A
BCC  FD5
CMP  #5B
BCC  FD6
```

We change this to:

```
CMP  #'!
BCC  FD5
CMP  #7B
BCC  FD6
```

The changes can be made in BASIC with POKE 3818,33:POKE 3822,123 or change hex locations \$0EEA to \$21 and \$0EEE to \$7B. The section of code we're modifying is located between source line numbers 4072 through 4193. Remember to rewrite the modified DOS to disk with WRITE DOS FILES (Menu selection "H") if you want your change to be permanent.

Other equally simple changes are also possible. You could change the wild-card character ("*") to any other character by changing location \$0EC7 to the desired character. A more ambitious task would be to increase the maximum file name length.

This brings up a final point – software compatibility. For example, if you changed the wild card character to "@," you couldn't run any previous programs that assume "*" as the wild card character. Our change is less dangerous – if you allow lowercase file names, the unmodified DOS won't be able to access it, although it will look fine on the directory. This change has not been exhaustively tested for

conflicts, so we can't guarantee its usage. Nevertheless, it seems quite useful and shows that some customizing can be accomplished with a few simple changes.

When experimenting, always keep a backup copy of your valuable disks in case something should go awry.

```
100 REM CHANGE DOS PROGRAM
110 REM FOR DOS 2.0S ONLY
120 REM CHANGE LOW RANGE CHECK FROM
130 REM 65 TO 33. THIS ALLOWS
140 REM ANY CHARACTER (EXCEPT
150 REM GRAPHICS AND INVERSE VIDEO)
160 REM TO START A FILENAME, INSTEAD
170 REM OF ONLY A THROUGH Z.
180 REM 0EE9 C941 CMP #'A
190 REM 0EE9 C921 CMP #'!
200 POKE 3818,33
210 REM CHANGE HIGH RANGE TO EXTEND
220 REM UP TO ASCII "z"
230 REM (LOWERCASE Z)
240 REM 0EED C95B CMP # $5B
250 REM 0EED C97B CMP # $7B
260 REM POKE 3822,123
270 REM NO NEED TO CHANGE NUMERIC
280 REM CHECK SINCE IT IS NO
290 REM LONGER EXECUTED, THANKS
300 REM TO THE ABOVE CODE.
```

Some Cautions

Care is necessary when making customizations. Only make the changes to a *copy* of your DOS – not the original “system master.” (You shouldn't be able to do this anyway, since the disk is “write-protected,” but better safe than sorry.) Remember that any files SAVED with your custom DOS will probably not be compatible with the original, unchanged DOS. Alternation of the DOS can have unpredictable effects; we urge caution and cannot accept any liability for software or hardware damage incurred through the use of this book.

Things To Look Out For

These modifications could make a customized DOS incompatible with the original, unmodified DOS 2.0S:

- 1) File name changes (such as allowing lowercase, or increasing

APPENDIX A

the length)

2) Changes to DOS file structure (such as using a different "linking" system)

3) Removing error-checks. These built-in traps insure disk integrity and reliability. When you alter one, you could risk muddling one or more files. For example, if you allow an automatic "wild-card" feature, where an asterisk is assumed at the end of a file, it could cause havoc when performing a SCRATCH, RENAME, or UPDATE operation. Another example is removing some of the qualifications for "burst-I/O." Remember that a lot of thought went into each design consideration.

Keeping these suggestions in mind, here are some ideas for modifications. You may need to type in and re-assemble (with your insertions) the entire DOS when making certain modifications.

1) Adding a STATUS check before a disk access. Have you ever noticed how long the drive will grind away when no disk is inserted? You can query the disk for its status, and even add a "Drive not ready" error message if the drive door is not closed or a disk is not inserted. Check your DOS manual for details.

2) Adding Disk Utility commands. These would be additional functions performed by the FMS, keyed to the "special command." Some of the tasks performed by the Disk Utility Package could be a part of the DOS kernal, such as LOAD and SAVE binary files. You could even implement new commands such as "relative file" support, where you only give the DOS a "record number" to randomly access a file. The file could be divided into records of any length.

3) Allocate more sectors for the directory, thereby extending the maximum amount of directory entries.

4) Add a disk name and/or disk I.D. number (serial number?) to the disk (maybe on sector 720). It could even print out with the directory.

5) Given the extra "unused" bytes in the file name, add a byte for file type, such as program, data, object code, etc., and have it printed out with the directory, making it easy to identify files without having to use the extension. This would be hard to interface with software, however.

Remember that some of this is risky business. Keep backup disks for any disk you are "experimenting" with. That way, you should lose no important files.

The publishers and authors of this book disclaim any responsibility for errors or problems caused by modification of Atari DOS 2.0S.

NOTES

NOTES

NOTES

NOTES

COMPUTE! Books

P.O. Box 5406 Greensboro, NC 27403

Ask your retailer for these **COMPUTE! Books**. If he or she has sold out, order directly from **COMPUTE!**

For Fastest Service
Call Our **TOLL FREE US Order Line**

800-334-0868

In NC call 919-275-9809

Quantity	Title	Price	Total
_____	The Beginner's Guide To Buying A Personal Computer (Add \$1.00 shipping and handling. Outside US add \$4.00 air mail; \$2.00 surface mail.)	\$ 3.95	_____
_____	COMPUTE!'s First Book of Atari (Add \$2.00 shipping and handling. Outside US add \$4.00 air mail; \$2.00 surface mail.)	\$12.95	_____
_____	Inside Atari DOS (Add \$2.00 shipping and handling. Outside US add \$4.00 air mail; \$2.00 surface mail.)	\$19.95	_____
_____	COMPUTE!'s First Book of PET/CBM (Add \$2.00 shipping and handling. Outside US add \$4.00 air mail; \$2.00 surface mail.)	\$12.95	_____
_____	Programming the PET/CBM (Add \$3.00 shipping and handling. Outside US add \$9.00 air mail; \$3.00 surface mail.)	\$24.95	_____
_____	Every Kid's First Book of Robots and Computers (Add \$1.00 shipping and handling. Outside US add \$4.00 air mail; \$2.00 surface mail.)	\$ 4.95	_____
_____	COMPUTE!'s Second Book of Atari (Add \$2.00 shipping and handling. Outside US add \$4.00 air mail; \$2.00 surface mail.)	\$12.95	_____
_____	COMPUTE!'s First Book of VIC (Add \$2.00 shipping and handling. Outside US add \$4.00 air mail; \$2.00 surface mail.)	\$12.95	_____

All orders must be prepaid (money order, check, or charge). All payments must be in US funds. NC residents add 4% sales tax.

Payment enclosed Please charge my: VISA MasterCard
 American Express Acc't. No. _____ Expires ____ / ____

Name _____

Address _____

City _____

State _____

Zip _____

Country _____

Allow 4-5 weeks for delivery.



If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!** Magazine. Use this form to order your subscription to **COMPUTE!**

For Fastest Service,
Call Our **Toll-Free** US Order Line

800-334-0868
In NC call **919-275-9809**

COMPUTE!

P.O. Box 5406
Greensboro, NC 27403

My Computer Is:

PET Apple Atari VIC Other _____ Don't yet have one...

- \$20.00 One Year US Subscription
 \$36.00 Two Year US Subscription
 \$54.00 Three Year US Subscription

Subscription rates outside the US:

- \$25.00 Canada F=2
 \$38.00 Europe/Air Delivery FI=3
 \$48.00 Middle East, North Africa, Central America/Air Mail FI=5
 \$88.00 South America, South Africa, Australasia/Air Mail FI=7
 \$25.00 International Surface Mail (lengthy, unreliable delivery) FI=4,6,8

Name _____

Address _____

City _____

State _____

Zip _____

Country _____

Payment must be in US Funds drawn on a US Bank; International Money Order, or charge card.

Payment Enclosed

VISA

MasterCard

American Express

Acc't. No. _____

Expires _____ / _____

