31336 VIA COLINAS, WESTLAKE VILLAGE, CA 91361

SMOKE SIGNAL BROADCASTING

DISK SYSTEMS

INTRODUCTION

How much do you want to read? If you are like most people, you would rather start up the system right away and read the instructions later. This manual is designed to get you up and running with a minimum of reading; however, to become proficient at system operation, you will need to read the entire manual thoroughly.

If you have a Chieftain Microcomputer System, you may skip right to the section "Getting It Up".

If you have another type of microcomputer, you should first read the section "Installation Instructions" which includes information on hardware requirements and minimum CPU clock frequency.

SSB DISK SYSTEM REFERENCE MANUAL

[redacted]

PART 1: INSTALLATION INSTRUCTIONS

UNPACKING

Carefully remove the disk system from its shipping container. Remove the disk controller board. Remove the protective packing material wrapped around the board. Inspect both disk system and controller board for any shipping damage. Any damage should be reported to the shipping agent.

HARDWARE REQUIREMENTS

In order to use your disk system you will need:

1. An operational SS-50 BUS 6809 computer or its functional equivalent. The system should contain:
   a) 8K of RAM at $C000 thru $DFFF. In addition, a minimum of 16K of RAM from $0000 through $3FFF will be required to run certain system programs.
   b) Nothing that would interfere with the controller PIA at $F77C through $F77F.

MINIMUM CLOCK FREQUENCY

The minimum processor clock frequency for operation of the minifloppy systems is 0.70 MHz. For the 8" systems, the minimum clock frequency is 1.40 MHz. Most 6809 computers now in use operate with a clock frequency near 1.0 MHz. Thus, you may have to modify your computer to operate at a higher clock frequency when using an 8" disk system. This is easily done and results in your programs running proportionately faster.

NOTE: The SWTP MP-16 will not operate above approximately 1.05 MHz because of the method used to refresh its dynamic memory chips. The SSB M-16X uses static memory and every board is tested at 2 MHz. If you have a MP-16, check with your local dealer to determine his policy on a trade-in allowance towards an M-16X.

TO INSTALL IN YOUR SYSTEM

1. MAKE SURE ALL POWER IS REMOVED FROM THE COMPUTER SYSTEM.
2. Read the hardware requirements section of the installation instructions and make certain your system has not been modified to be incompatible with the SSB disk system.
3. Install the controller card in any of the large slots so that the component side faces forward.
4. Install the disk interface cable. (Use J-2 for 5" systems or J-3 for 8" systems). When adding additional drives, refer to the section on "Installing Additional Drives".

Be sure that none of the drives contains a diskette. Apply power to the disk system by plugging it into the 115 volt power source and pressing the power switch. The power switch is on the front panel of the BFD-68 and on the rear panel of the LFD-68 and DFD-68.

6. Power may now be reapplied to the computer.

!!! CAUTION !!!
Never turn power on or off to the disk system or to the computer to which the disk system is attached while a diskette is installed in any of the drive units. During power on or off, a false write could occur which may destroy the data stored on the diskette.

!!! IMPORTANT !!!
Read the limited warrantee and software license information in this manual prior to using the system.

This completes the hardware installation. To get the software going, read the section "GETTING IT UP".

A WORD ABOUT DISK DRIVES

Smoke Signal Broadcasting uses two types of minifloppy drives in the disk system: Shugart Associates SA400 drives and Microperipherals Inc. B51 drives. These drives are functionally equivalent but have two differences. One is physical; The SA400 has a different front panel from the B51. Both operate essentially the same. Diskettes are inserted into the drives with the label facing the left side of the system. The edge of the diskette with the long oval cutout should be the first edge of the diskette to enter the drive. The other difference has to do with the speed at which at which the stepper motor moves the head. For the SA400, it takes about 40 milliseconds per step; the B51 takes about 12 milliseconds. DOS09 will work properly with either (or both) types of drive, and the controller board supplied with the 5" system will work with either (or both) types of drive. See the SSTEPR command description for additional information.

Shugart Associates SA800 drives are used in the single sided 8" disk systems and SA850 drives are used in the double sided 8" systems. The controller supplied with the 8" system will operate with either type drive without modification, as will DOS09.

PART 2: GETTING IT UP

CHECK YOUR SYSTEM

Before you run DOS09, you must make sure that your system is up and running under your ROM monitor. Prior to loading DOS09 into memory, you should be familiar with the disk bootstrap operation. If you are already familiar with this, skip the explanation and continue with "BOOTING DOS09".

The initial loading of software into a computer with little or no permanently resident software involves a process called bootstrapping. Bootstrapping is the use of a typically small, dumb program to load a larger, smarter loader which in turn can load the desired program (usually the operating system) into the computer.

The MON09 monitor EPROM contains a bootstrap loader capable of reading and executing a second loader located in sector 0 of track 0 from a disk mounted in drive 0. The contents of this sector is first initialized by the FORMAT program, then LINK'ED to contain information to cause DOS09 to be loaded and executed. "LINKING" the disk is a process whereby the boot routine is told what file to load and run when the disk is booted. (See the LINK command description.)

BOOTING DOS09

Now locate the disk with DOS09 version 1. To boot DOS09, place the system disk into drive 0 and transfer control to the ROM/EPROM bootstrap loader. MON09 users can initiate the boot with the MON09 Q command. Booting in this manner is referred to as "COLD STARTING" because all monitor and DFM temporaries are initialized. (Once DOS09 has been loaded into memory, it is possible to WARM START the monitor, that is, to restart the monitor without initializing everything by transferring control to the warmstart address of DOS09. Pefer to APPENDIX A for the warmstart address of the DOS.)

INSTALLING AND CONFIGURING DOS09

DOS09, when booted or cold started, will attempt to run the command:

```
EXEC,START.UP
```

This requires the utility EXEC.$ and an exclusive file named START.UP found on drive 0. The START.UP file is a procedure file containing a series of DOS09 commands which can be selected to set system parameters as you would like them. For your convenience, DOS09 is supplied with a procedure file which will provide simple system configuration at boot time.

SSB has designed DOS09 to work with the MON09 monitor.

Therefore, if you are using some other monitor, you should read the technical sections of the manual to determine the changes that might be required for your monitor.

Initially, EXEC.$ does not exist, so the first time you boot up DOS09, the system will respond "EXEC? CMD NOT FOUND". The first thing you should do is to select one of the EXECn.$ files and rename it to "EXEC.$". Then create a START.UP file for your system (see the build command), or use the one supplied with the system, and re-boot the system. You should now be fully up and running. If you will be using a printer with the system, read the section "Installing DOS09 Printer Drivers". Otherwise, you are ready to run any command or program. For example, to invoke the formatter, simply type the command: FORMAT

At this time it is a good idea to make a backup copy of your system disk and place it in permanent storage. Formatting and disk Backup are explained in Part 3 of the manual.

To configure DOS09 to meet your individual requirements at boot time, you may want to create a custom START.UP file to contain the commands you need. Be sure to read the description of the SET command to fully understand how to set the system parameters. We would suggest experimenting with your system parameter settings as commands from the terminal, and then using the BUILD command, generate trial procedure files to execute with the EXEC command before renaming them to bear the name "START.UP".

INSTALLING DOS09 PRINTER DRIVERS

DOS09 is supplied with both source and load modules for both serial and parallel printer drivers. The serial driver module is named SPRINT.SYS, and the parallel driver is named PPRINT.SYS. You should select the appropriate driver for your type of printer, and use the RUN command to load and initialize the driver. As an example:

```
RUN,PPRINT.SYS
```

will load and initialize the parallel printer driver PPRINT.SYS. SSB suggests for your convenience that you include a RUN command for the driver module you select within the START.UP file so that you will automatically be set up for hardcopy when you boot up.

Since the source code for both drivers is supplied, you are free to examine and perhaps modifiy the drivers to accomodate any special hardware requirements. Adequate memory space has been reserved for customizing the drivers; see Appendix A for its location.

Unless the HARDCOPY location has been changed using the SET command, PPRINT.SYS will expect to use a parallel card located at port 6. SPRINT.SYS will default to a serial card in port 3.

TROUBLESHOOTING BOOT PROBLEMS

If you cannot get DOS09 to boot in, you may have hardware problems. Some commonly encountered problems are:

1. The head loads and unloads without DOS09 printing its banner. This may be due to no memory or faulty memory between $C000 and $DFFF. Read the section on hardware requirements.

2. Disk seems to be working, but no results. Two possibilites:
   a) Since all the ICs are in sockets, some may have been dislodged in shipping. To make sure, push in all the ICs and try again.
   b) The head carriage assembly on the disk drive may have been stuck or mispositioned during shipping. To free it, power down the disk system and, with the eraser end of the pencil or a finger, move the head assembly back and forth to make sure it is free. Then position the carriage so that the stylus is in the groove on the stepper motor cam. Then try booting the system again. This problem is confined to the Shugart SA400 5" disk drive.

[redacted]

PART 3: OPERATING DOS09 VERSION 1

INTRODUCTION:

The DOS09 command set consists of both memory resident and disk resident (Transient) commands. Memory resident commands reside in memory as part of DOS09. Transient commands reside on a disk as files with names bearing a "$" filename extension. Disk resident commands are loaded into memory and executed when the command is invoked. This facility offers two distinct advantages: 1) the resident monitor is smaller because all the command processors need not be resident in memory, and 2) the user is easily able to implement new monitor commands without modification of the monitor. For these reasons, the majority of the DOS09 command set has been implemented as disk resident command processors.

Moreover, the transient command scheme offers the users of multi-disk systems the ability to specify that the command is to be found on a specific disk drive. This feature is particularly useful when invoking transient commands which might conflict with a command found on another disk. To invoke a transient command residing on a specific drive, the command is entered by preceeding the command name with the drive number followed by a colon, ':'. For example, 2:LIST,1 will retrieve the LIST command processor from disk drive 2 to list the directory for drive 1. All commands will default to be called from the system drive unless specified as above. Any target file will default to be retrieved from the work drive. (See the SET parameter command description for the method of defining the system and work drive numbers).

COMMAND DESCRIPTIONS:

DOS09 indicates its readiness to accept a command by displaying "DOS: " on the terminal. At this time a new command line may be entered.

ENTERING COMMANDS

Commands given to DOS09 are entered on a line input basis. This means that the entire line is typed in before DOS09 begins to process it. Using such line input gives the user a chance to easily correct typing errors or to completely cancel the line before DOS09 starts any processing. The previously entered character can be deleted by typing the backspace (BS) character (control H by default). To delete the entire line, type the delete line (DL) character which defaults to control X. It is important to note that the character definitions for the DL and BS characters are user definable using the SET command, so you can redefine the backspace (BS) or delete line (DL) character definitions to meet your specific system requirements.

RE-EXECUTING COMMANDS

DOS09 provides you two methods for recalling the last command for re-execution. First, to recall the last entered command for re-execution, type a control D. DOS will immediately re-display and re-execute the command.

Second, to recall the last command to either display it, or to display it and modify it, enter a control A. DOS09 will display the previously executed command, leaving the line buffer pointer and your terminal's cursor pointing at the termination character of the line. You can now edit the line by backspacing and entering the changes. Typing a carriage return will cause the command line to be executed. The DL (delete line) character will cancel the line.

DOS09 is supplied with the following commands:

	COMMAND NAME	FUNCTION:
	APPEND	Splices two files together to form one file
	BACKUP	Makes a backup copy of an entire disk
	BUILD	Build command control file
	CONCAT	Allows merging text files into a new file
	COPY	Allows files to be copied from disk to disk
	DEL	Remove a file from a disk
	EXEC	Execute command control file
*	EXIT	Exit to ROM resident monitor
	FIND	Type map of file address information
	FORMAT	Format a blank disk
*	GET	Load a binary object file into memory
*	GOTO	Go to hex specified address
	LINK	Set up information to boot the monitor
	LIST	List the disk file directory
*	P	Initialize for terminal output to printer
*	REENT	Re-enter the last transient program loaded
	REPAIR	Repair (edit) a sector on the disk
	REN	Change the name of a file
*	RUN	Load a file into memory and begin execution
	SAVE	Save memory into a file
	SAVET	Save transient area memory into a file
	SDC	Single disk drive copy
	SET	Set system parameters
	SSTEPR	Set disk drive step rates
	VIEW	Type contents of an editor text file

(* indicates a memory resident command)

The following conventions are used to describe files under DOS09:

The disk drive unit numbers are 0, 1, 2 and 3.

Angle brackets, < >, are used to enclose a string of characters to indicate that the string indicates one item. For example: <UNIT NUMBER> is used below to represent the disk unit number for a file.

If a field in a command is optional, the optional portion is bracketed in square brackets. For example: [,<UNIT NUMBER>] means that the comma followed by a unit number is optional.

Many operations require the specification of a file; <FILE SPEC> will be used as an abbreviation for the following:

```
    [<UNIT NUMBER>:]<FILE NAME>[.<EXTENSION>]
```

where <FILE NAME> may be one to six alphanumeric characters and <EXTENSION> may be up to three alphanumerics.

Some legal file specifications might be:

1:FILE.ONE	A file called "FILE.ONE" on unit number 1
ABC.1	A file called "ABC.1" assumed to be on work unit
FNAME	A file called "FNAME" assumed to be on work unit
0:FILE.ONE	Note that 0:FILE.ONE is a different file from 1:FILE.ONE

RESIDENT COMMAND DESCRIPTIONS:

This section describes those DOS09 commands which are memory resident. It is not possible to specify a disk unit number for retrieving these commands since these commands are not transient commands.

EXIT   EXIT:

The Exit command returns control to the resident ROM monitor. The exit is made by using a jump table entry whose address is specified in the "DOS09 MEMORY MAP" table. The address of the ROM monitor entry point can be changed by changing this table entry.

GET   GET,<FILE SPEC>[,<OFFSET>]:

The GEt command loads the file specified into memory and returns to the monitor. <OFFSET> is an optional hex value which when entered is added to the load address of the file.

For example:

```
    GET,GOING           Load the file "GOING" from the work drive
    GET,1:XYZ           Load the file "XYZ" into memory from drive 1
    GET,PROG.BIN,1000   Load "PROG.BIN" $1000 locations above
                            the specifed load addresses
```

GOTO   GOTO,<HEX ADDRESS>:

The GOto command is provided for convenience. It is used to start execution of a program already loaded into computer memory. <HEX ADDRESS> is a 1 to 4 digit hex number representing the address where program execution is to commence. The primary reason for using the GOTO is if there is a program already loaded into memory and you do not wish to load it off of the disk again. Caution must be exercised that the program really exists and has not been damaged since it was last loaded.

For example:

```
    GOTO,100     Transfer computer control to location $0100
```

P   P,<COMMAND LIST>:

The P command is unlike any others currently in DOS09. P signals the system I/O drivers to route the output of any command through the system printer driver. This is very useful for getting printed copies of the disk directory via LIST or to print the contents of a text file with the VIEW command. Proper P command operation requires that an appropriate printer driver like SPRINT.SYS or PPRINT.SYS has been initialized with the RUN command prior to using P. The ideal time to initialize the driver is at Boot time with a RUN command for the driver included in the START.UP file so the necessary vectors in the SYSTEM

PARAMETER table will already be set up. The required vectors are the port address of the printer, and the initialization and character output addresses of the driver. If these vectors have not been initialized, the P command will not enable printer output but will continue to output characters to the system terminal.

For example:

P,LIST	Print directory of drive 0 on printer device
P,VIEW,LAST.MSG	Print file "LAST.MSG" located on the work drive on the system printer

RUN

RUN,<FILE SPEC>[,<OFFSET>]:

The RUn command performs the same function as the GET command except that if a transfer address was given when the file was saved, this address will be transferred to once the file has been loaded. NOTE: the RUN command has the same optional offset load capability as the GET command. The offset will also be added into the transfer address. It is the user's responsibility to know if a file can be run when loaded with an offset.

EXAMPLES:

RUN,2:GAME.1	Run the file "GAME.1" on disk 2
RUN,1:PROG.REL,1000	Load with a $1000 offset and run "PROG.REL"

REENT

REENT:

The REEnt command checks the reentry status bit in the monitor, and, if set, transfers control to 3 plus the transfer address of the last program loaded. It is the user's responsibility to insure that executable code exists at this location. Typically, a user program would be written this way:

```
      ORG $100
START JMP RUN         MAIN ENTRY POINT
      JMP RERUN       REENTRY POINT
      ...
RUN   JSR INIT        INITIALIZE THIS PROGRAM
      LDA YMSTAT      SET 'REENTRY ALLOWED' BIT
      ORA #1
      STA YMSTAT
RERUN ...             CONTINUE
```

Typing REENT after the program has been executed will cause it to execute again, but without calling the subroutine INIT.

TRANSIENT COMMAND DESCRIPTIONS:

This section describes the disk resident commands supplied with DOS09. DOS09 can be told to search a specific disk for any of the following commands by preceeding the command name with a unit specifier such as "1:" to indicate unit 1. If the unit number is omitted, the system drive unit number will be assumed (i.e., unit number defaults to system drive).

APPEND

APPEND,<THIS FILE>,<THAT FILE>:

The APPEND command allows two files to be combined into one file. The file specified by <THIS FILE> is appended to <THAT FILE> where both files are assumed to reside on the disk specified by the file specification of <THAT FILE>. Once appended, the file name of <THIS FILE> is removed from the disk directory. Note that if the drive number of <THAT FILE> is not specified that the work drive will be assumed as the default drive.

FOR EXAMPLE:

```
    APPEND,FILE.ONE,2:FILE.TWO
```

This causes the file "FILE.ONE" on disk drive 2 to be appended to file "FILE.TWO" on disk drive 2. The file "FILE.ONE" will cease to exist on disk 2.

BACKUP

BACKUP:

THE BACKUP command allows the user to make a complete image copy of a disk mounted in drive 0 to a disk in drive 1.

BACKUP makes a complete disk image of the master disk by transferring all the data, on a track by track basis. Therefore, if the files on the master disk are physically scattered, the image copy of the master disk will in turn reflect the same physical file layout. To duplicate a disk to ensure a "contiguous" file layout, use the COPY utility with a freshly formatted diskette. Since BACKUP transfers the LINK'ed boot information, system disks duplicated with BACKUP do not require re-LINKing.

When invoked, BACKUP will prompt the user with its banner. Then the user is prompted to place the master disk (the disk to be copied from) in drive 0.

Make sure that the disk you intend to be the new copy is mounted in drive 1, as BACKUP will write on every track and sector of the disk, effectively eliminating all the files which existed prior to executing BACKUP.

When the disks are mounted and you are ready to proceed, type carriage return. The BACKUP operation will then begin. A successful disk BACKUP operation requires that the disk in drive 1 does not contain any bad sectors. * Do not attempt to intermix single sided and double sided diskettes when using BACKUP. SSB suggests using COPY where any questions exist.

BACKUP has a number of error messages that report the various error conditions that may occur during the backup operation. These error messages are self explanatory.

BUILD   BUILD,<FILE SPEC>:

The BUILD command is provided for those desiring to create a small text file quickly (such as START.UP files or EXEC files). The main purpose for BUILD is to generate short text files for use by the EXEC command.

BUILD has the general command form:

```
     BUILD,<FILE NAME>
```

where <FILE NAME> is the name of the file you wish to create. The default extension for the file is Type 4, which is initially 'CTL'. The unit number in which the file will be created is that of the work drive. If the output file has already been created, the prompt "DELETE OLD FILE? " is issued. If the file is OK to delete the user is expected to respond with 'Y'es. Failure to respond with Yes will terminate the build session since the file currently existing cannot be used for writing.

After you are in the BUILD mode, the terminal will respond with the '#:' prompt. To enter your text simply type on the terminal the desired text, keeping in mind that once the carriage return is entered the line is in the file and may not be changed. Any time before the carriage return is entered the functions defined for line input editing are available for text editing. To terminate the build session, entry of a carriage return (null line) will write the remainder of the file to the disk file, close the file and return to DOS09.

CONCAT   CONCAT:

The CONCAT command concatenates text files together by reading input files and transferring their contents to an output file. At the conclusion of an input file transfer, the output file remains open for additional input file transfers. When the user no longer wishes to append another file, the output file is closed. The contents of the output file reflect the order in which the input files are specified.

CONCAT allows the user to specify input files located on any drive regardless of the drive on which the output file is assigned. For example, a file on drive 0 and another file on drive 1 can be CONCATenated to form an output file on drive 2.

CONCAT first prompts the user by asking for the output file. After an output file has been specified, CONCAT then prompts the user for the name of the first input file. After the first input file has been specified, the contents of the input file are transferred to the output file. Next, CONCAT will ask if you want to append a file. Respond with a Y or N. If Y, then CONCAT will ask for the next input file name. Once entered, the next input file will be read and transferred to the output file. At the conclusion of this and subsequent input file transfers, CONCAT will continue to ask you if you want to append another file until you enter an N response, at which time the output file will be closed. As part of the output file closing process, CONCAT will mark the end of file with an EOF character (control Z). This is done to ensure text file compatibility with other software.

COPY

COPY,<FROM>,<TO>:

The COPY command provides a means for copying files. COPY has the general command form:

```
COPY,<FROM>,<TO>[<SWITCHES>]
```

Where <FROM> specifies the source of information to be copied, <TO> specifies the destination, and <SWITCHES> represents the specification of various options. COPY will use the WORKING DRIVE value for <FROM> and <TO> if either is omitted from the command line (refer to SET for further explanation).

Either <FROM> or <TO> may reference "$"-files (e.g. LIST.$), and either may be an ambiguous file specification. An ambiguous file reference uses the characters "?" and "*" as part of the file name to reference a family of files as opposed to a specific file reference. "?" is a wild-card character which is used to mean "any character in this position". "*" is a wild card field which when used in a file reference means "treat this field as if it were filled with "?"s.

Examples:

.	means all files
*.OBJ	means all files with an extension of "OBJ"
SAM.*	means all files with a file name of "SAM"
*.$	means all command (i.e. $-sign) files
A?????.BIN	means all files with an extension of "BIN" where the file name starts with the letter "A"

Z?.*	means all files where the file name is two characters long and starts with "Z"

NOTE: "1:" is treated the same as "1:*.*"

The function of the COPY command is to copy all files specified by <FROM> to files specified by <TO>. The only restrictions as to what may be done are restrictions of common sense. For example: COPY,1:*.$,2:ABC makes no sense since this tries to copy all command files on drive 1 to a single file, ABC, on drive 2.

COPY,1:*.OBJ,2:*.BIN is a valid statement which says to copy all files on drive 1 with an extension of "OBJ" to drive 2, but in the process changing "OBJ" to "BIN" for each file copied.

COPY,1:A?????.*,2:Z?????.* says to copy all files on drive 1 which start with the letter "A" to drive 2 while changing the "A" to a "Z".

COPY 1:?A????.* 2:?Z????.* says to copy all files from drive 1 which have an "A" as the second letter to drive 2 while changing the second letter to a "Z".

COPY has one other feature which allows the user to abort the COPY command by typing the break character (control C). COPY uses the control port status routine of DOS09 to determine if the user has attempted to abort the command. COPY returns control to DOS09 after the present file which is being copied is finished. (Refer to SET command for further explanation).

<SWITCHES> represents the specification of 1 or more of the following options:

"/NV"	No Verify. The default mode of operation is to verify all files copied by reading back the output file and comparing it to the input file data.
"/C"	Confirm. COPY will print the file name of the next file to be copied and then wait for the operator to say yes ("Y") or no ("N") to the copying of that file when the confirm option is specified.
"/L=n"	Memory limit. The memory limit option is used to limit the amount of memory used by COPY. COPY normally assumes that 16K of RAM starting at 0 is available for use. The "n" specified represents the high order digit of the next 4K of memory NOT available for use by copy (the default is /L=4). "n" may be the numbers 2 through 7 for 8 through 28K.
"/P"	Partial. The partial option forces COPY to copy as much possible of a file which has a read error that prevents the entire file from being read. Otherwise, the remainder of the last buffer read will not be written when the read error occurs.

Examples of using COPY, including using switches.

COPY 0:,1: Copy all of disk 0 to disk 1; the COPY command will attempt to copy all files into contiguous segments which will speed the access time of DOS09 considerably.

COPY 0:,1:/L=6 Same as above except use all 24K for buffer to copy all of disk 0 to disk 1

COPY,0:,1:/L=6/P/NV/C Use all 24K for buffer to copy all of disk 0 to disk 1, without performing verification, but copying as much of the file as possible while asking for confirmation on each file to be copied.

COPY,*.$,1:,/C/L=2 Use 8K as memory limit to copy all command files from disk 0 to 1 with confirmation

NOTE: The number of switches specified and the order in which they are specified is not important.

DEL DEL,<FILE SPEC>,[<FILE SPEC2>,,<FILE SPECN>]

The DELete command removes a specified file or list of files from the directory of the specified unit. A number of file specifications can be included in the command line in order to delete a list of files. The number of files that can be deleted in one command line is limited only by the number of file specs that can be entered in the command line. After a file is deleted, the sectors used by the file are now available for reuse. Disk repacking is never required when files are deleted.

Examples:

DEL,AB.123
This deletes file "AB.123" from the work drive.

DEL,2:FILE.ONE.
This deletes a file called "FILE.ONE" from the directory of disk drive 2.

DEL,1:REN.$
This deletes from disk drive 1 the transient command file for the REName command.

DEL,AB.123,2:FILE.ONE,1:REN.$
This deletes all of the above files as though each had been entered by separate commands. Any number of files that may be entered on the command line may be deleted.

EXEC EXEC,<FILE NAME>:

The EXECute command is used to process a text file as a list of commands, just as if they had been typed from the keyboard. This is a powerful feature in that it allows very complex procedures to be built up as a procedure file. When it is desirable to run the procedures, it is only necessary to type EXEC followed by the file name of the procedure file. Essentially all EXEC does is to replace the keyboard entry routine with a routine which reads a character from the procedure file each time the keyboard routine would have been called. The system routines have no idea that the characters of input are coming from a file instead of the terminal.

EXEC has the general command form:

```
    EXEC,<FILE NAME>
```

where <FILE NAME> is the name of the procedure file. The default extension is type 4, which is initially 'CTL', although it can be redefined using SET. An example will give some idea on how EXEC can be used to create a system start up file. The BUILD utility should be used to create this file. The creation of this file might go as follows:

```
    DOS: BUILD,START.UP<CR>     (<CR> means type carriage return)
    #:SSTEPR,F,F,S,S<CR>
    #:RUN,PPRINT.SYS<CR>
    #:SET,CRT=E008,WD=0,DP=0<CR>
    #:<CR>
    DOS:
```

The first line of the example tells DOS09 we wish to BUILD a file called START.UP. Next the command lines are typed in just as they would be typed into DOS09. The first command will execute the file named SSTEPR which will set the stepping rates used with each disk drive; in this example, drives 0 and 1 will step fast, while drives 2 and 3 will step slow. Then PPRINT.SYS will enter the information to drive the system printer. Next the SET file will be executed to establish the desired system parameters for the control port address at $E008, page WIDTH and DEPTH disabled.

* SSB has supplied EXEC at several locations on each DOS disk. If you are using the EXEC command to do batch processing, you may wish to use EXEC at another location depending on the memory configuration in your machine. Following is a table showing where EXEC is located:

EXEC	EXEC ADDRESS
EXEC3	$3E00-$3FFF
EXEC5	$5E00-$5FFF
EXEC7	$7E00-$7FFF
EXECB	$BE00-$BFFF

You should rename one of these to "EXEC.$" for general use.

FIND

FIND,<FILE SPEC>:

FIND is a command used to determine where an object file would load and what its transfer address would be.

General command format:

```
    FIND,<FILE SPEC>
```

Where <FILE SPEC> specifies an object file in the binary record format used by the "SAVE" command and the SSB ASSEMBLER.

Example: Find out where FIND loads.

```
    FIND,FIND.$
     LOADS FROM CD80 THRU CEB2
    TRANSFER ADD = CD80
```

FORMAT

FORMAT is a command for the initialization of blank diskettes for use with DOS09. FORMAT must be used to prepare every disk to be used by DOS09. FORMAT is invoked by either of the following two command syntaxes:

Example 1:

```
          FORMAT,<Format Parameters><cr>
```

Example 2:

```
          FORMAT<cr>
```

FORMAT will prompt the user with the following message:

```
** SSB FORMATTER V1.0 **

     /# PRINT SWITCH SETTINGS
     /? HELP
```

After a drive is specified, FORMAT will ask if you really want to format on the specified drive. Answering with any character other than Y will cause a return to DOS09. Answering Y will cause FORMAT to ask "HONEST?". Respond with H if you really do want to format on that drive. Any other character will cause a return to DOS09.

The first command example will cause FORMAT to format the disk according to the specified format parameters. At the conclusion of the formatting operation, control is returned to DOS09.

The second example will cause FORMAT to be loaded and initiated, indicating its operational readiness by its "FMT:" prompt. The user may now enter the desired formatting parameters. After the command line is terminated, the FORMAT program will format the specified disk. After the disk is formatted, the FORMAT program will retain control and display its "FMT:" prompt again. The user can repeat the last format operation, or alter the switch settings as required.

If the command is repeated, the FORMAT switch settings used during the previous formatting operation remain in effect in the next formatting operation unless otherwise specified by the user.

To return to DOS09, type a BREAK character or a carriage return. After the FMT: prompt, control will return to DOS09.

The general format of the commands following the "FMT:" prompt is as follows:

```
[<unit #>]/[<options>]
```

If a unit number is not entered, FORMAT will assume drive 0. If a unit number is supplied, it must be one of the numbers 0, 1, 2 or 3. Options, if used, are of the form: /<option>/<option>.

OPTION SYMBOL	FUNCTION:
#	Print the options currently in effect
?	Print a summary of available options
F	Fast Mode; do not verify
Q	Quiet Mode; do not print track # or bad sector messages during verification
D	Format a double sided disk
S	Format a single sided disk
8	Format an 8" disk
5	Format a 5" disk
A	Copy DFM09 overlay files onto new disk

The options D, S, 8 and 5 have default values corresponding to the type of disk on which FORMAT is distributed and normally need not be changed. The A, F and Q options may be turned off by preceeding the command symbol by a minus sign.

Options remain in effect until either FORMAT is terminated or the option is overridden. For example: /-F overrides /F and vice versa. To determine what the defaults are, type "FORMAT/#".

There are certain errors which are considered fatal by FORMAT. The printout of these messages are preceeded by "FE: ". Fatal errors cause the current process to be aborted. Such a fatal error might be:

```
FE: BOOT ERROR: XX
```

which means the sector for the boot program is bad. "XX" represents a hardware error code which indicates the cause of the error.

The /A option allows the user to copy the DFM09 overlay files onto the newly formatted disk. If the user prefers not to have the overlay files copied over to the new disk, the option switch /-A must be used.

LINK   LINK,<FILE SPEC>:

The LINK command tells the bootstrapping routine which program to load and run when the rom bootstrap routine is executed.

For example: LINK,MON causes the linkages on the work drive to be set up such that the file "MON" will be run when this disk is booted in. LINK,2:NEWMON similarly links the boot on disk two to "NEWMON". In this case, in order to then boot "NEWMON", you must move the disk with "NEWMON" on it down to drive zero before the ROM boot is executed.

LIST   LIST[,<UNIT>]:

The LIST command types the contents of the directory for the disk unit number specified. The "disk directory" is the list of file names contained on a given disk. If no unit is specified, unit zero is assumed. An optional title line may be printed by specifying the unit number followed by a statement of any length. Examples:

```
          LIST                  List directory of disk 0
          LIST,0                List directory of disk 0
          LIST,2                List directory of disk 2
          LIST,0, 1 A DIRECTORY List directory of disk 0 with added
                                title line "1 A DIRECTORY"
```

The format of the directory listing is as follows:

```
    LIST,2, 2 DIRECTORY

                FILES DISK NO. 2 DIRECTORY

FILE NAME  FTFS BTBS ETES NSEC   FILE NAME  FTFS BTBS ETES NSEC
ABC    .1  0200 8042 8043    2   FILE   .ONE 0100 8044 8047    4

 FREE SECTORS=1915       COMMANDS

FILE NAME  FTFS BTBS ETES NSEC   FILE NAME  FTFS BTBS ETES NSEC
APPEND.$   0200 804A 804A    1   BASIC  .$  0200 804B 8749   78

 FREE SECTORS=1915
```

The file names are under the heading "FILE NAME" followed by information about the file. "FT" is the file type (e.g. 02 for a sequential file and 04 for a random file). "FS" is file status and is non-zero only when the file is active (i.e. non-zero indicates the disk linkages are in a questionable state from the file not being closed properly). "BTBS" is the starting disk address of the file, "ETES" is the ending disk address for the file, "NSEC" is the number of sectors being used by the file.

Disk addresses are given by a four digit hex number. The first two digits are the track number beginning with $80 representing track #0. The second two digits are the sector number beginning with $40 representing sector #0. The number of sectors used and the number of free sectors are given in decimal.

The LIST command will first list all the non-transient files on the disk and then will list all of the monitor transient commands resident on the disk. The command file names are listed with the '$' extension in the "FILE NAME" column.

REN REN,<THIS>[.$],<THAT>[.$]:

The REName command changes the name of "THIS" file to "THAT".

For example:

```
    REN,FILE.BIN,FILE.$
       On the work drive the file "FILE.BIN" is renamed to the
       command file name of "FILE".

    REN,FILE.ONE,FILE.1
       On work drive the file "FILE.ONE" is renamed to "FILE.1"

    REN,2:THIS,THAT
       On disk drive 2, the file "THIS" is renamed to "THAT".

    REN,LIST.$,LI.$
       On the work drive, renames the command file "LIST" to
       command file "LI".
```

REPAIR REPAIR,<UNIT NUMBER>:

REPAIR is a command designed to facilitate disk patching for the recovery of files accidentally deleted or files which have a sector that cannot be read.

This documentation is restricted to explaining what REPAIR will do for you and makes no attempt to explain the magical things which can be done by using REPAIR. These magical things are accomplished by understanding what all the information means in the SSB DISK SYSTEMS MANUAL and in the "DOS09 SYSTEM PROGRAMMERS GUIDE".

General command format:

```
                    REPAIR,<unit number>
```

This loads the REPAIR command and prepares it to work on the disk in drive <unit number>. REPAIR will prompt with:

```
                    REP:
```

when it is ready to accept a command.

REPAIR executes the following commands:

R	Read
D	Display
C	Change sector buffer
N	Read next sector
P	Read previous sector
W	Write
U	Change disk unit number
X	Exit to DOS09

All input to REPAIR is done through the use of the monitor line input routine which makes the usual command editing features available. Note that the commas in the following command descriptions may be replaced by any valid delimiter (in particular, a space).

COMMAND DESCRIPTIONS:

R READ

Format: REP: R,<track number>,<sector number>

The specified sector is read into the sector buffer. REPAIR masks the track number with $7F and the sector number with $3F allowing these to be input either way as shown below:

```
REP: R,83,51    (Read track 3, sector $11)
```

or equivalently,

```
REP: R,3,11
```

D DISPLAY

Format: REP: D

REPAIR prints the current sector address followed by the contents of the sector buffer displayed both in HEX and in ASCII on the terminal. ASCII characters less than $20, or equal to $7F, are displayed as a period. (NOTE: REPAIR was written to produce an easily read display on a SOROC IQ 120 terminal with a 24 by 80 character display. The display may need to be modified for different terminals.)

The display is formatted with 16 bytes per line by 8 lines. Each byte is identified by its offset into the buffer which is shown as row and column headings. Thus the 17th byte of the buffer (offset of $10) is in row 1 column 0.

C CHANGE

Format: REP: C,<starting byte offset>,<new contents list>

Change allows for modifications of the sector buffer for updating the disk. For example: to change byte $31 (see D description for location this byte) to a value of $53, the following command is used:

```
                    REP: C 31 53
```

To convince yourself that the change has been made, use the D command. Note: this only changes memory, the disk has not yet been updated; this is done with the write (W) command.

Several consecutive bytes may be changed as shown in the following example:

```
                    REP 31 53 54 55
```

This example changes 3 bytes starting at byte 31.

N   READ NEXT
Format: REP: N

The sector specified by the forward linkages (in the sector buffer) is read into the sector buffer and the previous, current, and next sector addresses are listed.

P   READ PREVIOUS

Format: REP: P

The sector specified by the backward linkages (in the sector buffer) is read into the sector buffer and the previous, current, and next sector addresses are listed.

W   WRITE

Format: REP: W

The contents of the sector buffer are written to the current sector on the disk (the current sector address is shown with each disk read and when the D command is used).

U   CHANGE UNIT

Format: REP: U <unit number>
The current disk unit number is changed to <unit number> where <unit number> may be 0, 1, 2, or 3. The contents of the sector buffer are unaffected.

X   EXIT

Format: REP: X

REPAIR exits to DOS09.

SAVE SAVE,<FILE SPEC>,<SA>,<EA>[,<TA>]

The region of memory specified as starting at <SA> through location <EA>, inclusive, is saved in binary format into the file specified. Multiple regions of memory may be specified by repeating the <SA> and <EA> sequence for the number of regions required. <TA> is an optional starting address for use by the run command. If <TA> is not specified, no transfer address is written to the file. When the SAVE command is used to save transient monitor commands by using the '$' extension, a <TA> is required to be included.

For example:

```
    SAVE,1:MEMORY.LOW,0,1000
      $0 thru $1000 is saved as "MEMORY.LOW" on disk 1

    SAVE,2:SPOPN.$,0100,2000,0100
      A command file is saved on disk 2.

    SAVE,CMD.$,0-7F,200-37E,200
      A command file is saved on the work drive in two blocks
      with a transfer address. (NOTE: The '-' character may be
      used to aid in readability only. Any non-hex character
      will denote address separation.)
```

SAVET SAVET,<FILE SPEC>,<SA>,<EA>[,<TA>

The SAVET command is used exactly like the SAVE command. The file is loaded at $0100 rather than within the Transient Command Area thus allowing the saving of command files that are currently loaded within the transient area.

```
    SAVET,2:CMD.$,D080,D180,D080
      A transient command file is saved on disk 2.

    SAVET,CMD.$,D080-D180,D280-D380,D080
      A transient command file is saved on the work drive in two
      blocks with a transfer address.
```

SDC SDC,<FILE SPEC>[,MEM MAX]:

The SDC (single drive copy) command allows users of single disk drive systems to copy files from one disk to another disk by reading the entire file into memory, prompting the operator to change the disk, and then writing the memory image to the new disk. Thus the user is limited to copying files which will fit into memory.

The SDC command assumes that the user has 16K of RAM origined at 0, and that the disk drive in the system is unit zero. The upper limit of memory may be changed by specifing a value for <MEM MAX> in hexadecimal if more than 16K of RAM is available.

SET SET,<PARAMETER LIST>:

The SET Parameters command is provided so the user may control the characteristics of the terminal. With this command, the action of the terminal input and the display output may be controlled.

The general command format is:

```
     SET,<PARAMETER LIST>
```

where the parameter list is a list of parameter names, each followed by an equals ('='), and then by the value being assigned. Each parameter must be separated by a comma.

The default number base for numerical values is the base most appropriate to the parameter. In the descriptions that follow, 'hh' is used for parameters whose default base is hexadecimal, 'dd' is used for those whose default base is decimal.

The following fully describes all of the SET Parameters available to the user. Their initial values are defined, as well as any special characteristics they may possess.

BS=hh Back Space character

This sets the 'backspace' character to the character having the ASCII hexadecimal value of 'hh'. This character is initially a 'control H'(hex 08), but may be defined to any ASCII character. The action of the backspace character is to delete the last character typed from the terminal. Setting BS=0 will disable the backspace feature.

DL=hh Delete Line

This sets the 'DELETE LINE' character to the hex value 'hh'. This character is initially a 'CONTROL X' (hex 16). The action of the delete line is to 'erase' the current input line before it is accepted into the computer for execution. Setting DL=0 will disable the line delete feature.

DP=dd DePth

The DePth parameter specifies the number of lines to be displayed before an end-of-page pause occurrs. The occurrence of an end-of-page pause requires the WAIT to be ON. As an example, with DP=24 and WAIT=ON, 24 lines will be output and an end-of-page pause will occur. To resume the output, the CONT character must be typed. If DP=0, then the DePth control is disabled.

WD=dd WiDth

The WD parameter specifies the number of characters to be displayed on a physical line at the control terminal. Lines of text longer than the value will be broken at every multiple of WD characters. The WD parameter is initialized at 80 characters. If WD is set to 0 the width feature will be disabled, and any number of characters will be permitted on a physical line.

NULLS=dd   Null count

This parameter sets the number of non-printing (Null) 'pad' characters to be sent to the terminal at the end of each line. These pad characters are used so the terminal carriage return has enough time to return to the left margin before the next printable character is sent. The initial count is 0. Users of non-CRT terminals may want to set NULLS since pad characters are sometimes required on these types of terminals.

EJ=dd   EJect count

This parameter is used to specify the number of 'eject lines' to be sent to the terminal at the bottom of each page. If the value is 0 (which is the default value), no 'eject lines' are issued. An eject line is simply a blank line (carriage return/line feed) sent to the terminal. This feature is especially useful for terminals or printers with fan-fold paper so as to skip over the fold.

WAIT=ON or WAIT=OFF   Pause control

This parameter enables(WAIT=ON) or disables(WAIT=OFF) the end-of-page pause feature. If WAIT is on and DP (depth) is set to some non-zero value, the output display is automatically suspended at the end of each page. The output may be restarted by typing the "CONT" character. If WAIT is disabled, there will be no end-of-page pausing. This feature is useful for those using high-speed CRT terminals to suspend output long enough to read the page of text.

STOP=hh   STOP character

The STOP character is used to stop output from being displayed. The STOP character, whose ASCII hex value is 'hh', is initially $1B, the ESC character. You can use the STOP character, for example, to temporarily halt the listing of a file being VIEWed, a directory LIST, or an assembler listing. If the STOP character is typed before the end of a line is reached, DOS09 will proceed to display the remaining portion of the line and then stop prior to issuing a carriage return -line feed sequence. Once the output has stopped, the output can be resumed on receipt of a CONT character. Note: STOP is disabled until "CRT=hhhh" has been set. See "CRT=".

CONT=hh   CONTinue character

The CONTinue character is the functional complement ot the STOP character. The CONT character, whose ASCII hex value is 'hh', is initially $1B, the ESC character. The CONT char. is used to continue output to be displayed once it has been stopped using the STOP char. Since the STOP and CONT characters are initially set to the same ASCII code, $1B (ESC), the effect of the ESC key will be that of a switch which toggles the output off and on with each ESC keystroke. If you prefer separate codes for STOP and

CONT, simply SET STOP or CONT or both to the ASCII codes you prefer.

BREAK=hh   BREAK character

The BREAK character whose ASCII hex value is 'hh', is defined to be the system BREAK character. The initial value of BREAK is $03, which is a control C. The BREAK character is used to abort a command while its output is being displayed, after entering a STOP character, or when an end of page pause is encountered. For example, if you just started VIEWing a long file, and you suddenly discover that you entered the wrong file name, just type a BREAK character and you'll get back to DOS fast. You can now correct the situation and try again.

EXT=d,<EXT> Define the default extension of an extension Type.

The decimal EXTension Type 'd' is defined to be the character string of <EXT> of up to 3 characters in length. The default extension definition feature of SET gives you the capability of defining the extension used as the default extension by programs which implement default extension assignments. Programs which offer default extensions, or which assign extensions, do so according to an extension type. The table below enumerates each of the extension types, along with the initial definition, and its use. By assigning default extensions to suit your particular file naming conventions, you can truly have it "your way".

TYPE	INITIAL DEFINITION	USE
0	BIN	Default extension for formatted binary object files created by the Assembler.
1	TXT	Default extension for Editor files and Assembler input files.
2	SRC	A source file.
3	BAS	A BASIC program.
4	CTL	Default extension for control or procedure files that are processed by EXEC, and procedure files that are created by BUILD.
5	BAK	Reserved extension to be used only by the Editor for generating backup files.
6	DAT	A data file.
7	FOR	A FORTRAN program.
8	TMP	Reserved extension to be used only by the Editor for naming temporary work files.
9	null	Spare - user definable.

UCT   User Command Table clear

This parameter will clear a User Command Table extension that may currently be in effect. This will clear the extension without need of re-bootstrapping DOS09 to disable the User Command Table extension.

CRT=hhhh   CRT control port address

This parameter will allow changing the default control port address. The port at this address is checked for the entry of the BREAK and STOP characters and does not change the port specification of the system ROM monitor character input routine. The CRT address must specify the address of a compatible serial interface to which the system terminal is connected. Compatible interfaces are the SER-2 and MPS types which use the ACIA (6850) device. The CRT address must specify the status register of the ACIA. If the CRT address is set to an interface not connected to the system terminal, an incompatible interface, or an incorrect address, the BREAK and STOP features will be effectively disabled. * Note, the "port address" utilized by CRT refers to the lower address of the A or B side of the interface.

HC=hhhh   Hard-Copy (printer) port address

This parameter will allow changing the hardcopy port address to that of another compatible interface. The selection of a port address to which a compatible interface is connected is required for the correct functioning of the 'PRINT.SYS' drivers. If multiple hard-copy devices are available but are interfaced using identical I/O cards located at different port addresses, changing the device address will allow utilization of each device without the need for loading a different driver. * Note, the "port address" utilized by HC refers to the lower address of the A or B side of the interface.

SY=h and WK=h   Setting SYstem and WorK drives

These parameters allow specification of the SYstem and Work drives. The system drive is used by DOS09 as the default for command names or in general, the first name on a command line. The work drive is used as the default on all other file specifications within a command line. When the system is initialized, both the system and working drives are set to drive 0.

DATE=<date string>   define date

The system date can be specified by a character string of up to 15 characters in length. Any ASCII character, excluding control characters and commas, can be included in the string. The comma serves as the delimiting character for separating the last character in the string from the next parameter definition on the command line. The format of the date string is left to the user's discretion. However, SSB suggests using the format:

```
DAY MTH DD YYYY
```

LC=ON or LC=OFF SET LOWER CASE LOCK ON OR OFF

The Lower Case lock parameter controls the representation of lower case characters on the system terminal. The LC parameter internally controls the lower to upper case character conversion within the generalized character output routine. If the LC parameter is on, a lower case character output to the system terminal or printer will appear as an uppercase character. Moreover, if LC=ON, lower case characters received from the system terminal will be echoed by DOS09 as upper case, although they are internally stored as lower case. If you are using a terminal which does not respond gracefully to lower case characters, SSB suggests that the LC be set ON. If LC=OFF, the lower to upper case conversion is inhibited, allowing lower case characters to be displayed as such.

MEMAX=hhhh DEFINE MEMORY ALLOCATION LIMIT

The MEMAX parameter is an address defining the upper limit for workspace allocation by system or user programs. Programs which utilize the MEMAX parameter will not attempt to utilize memory beyond MEMAX if MEMAX has been set. (If MEMAX=0000 it is not set.) Thus, if you wish to "protect" a region of memory, set MEMAX to an address just below that of the region of interest. Similarly, you can set MEMAX equal to the address of where your "user" memory ends to tell DOS09 and other programs how much memory is available to them. The MEMAX parameter is accessable to user programs.

SSTEPR SSTEPR,<PARAMETER LIST>:

The SSTEPR command is provided to allow the user to configure the step rate table in DFM09 to match the stepping rate characteristics of his disk drives. It is a transient command that executes from the TCA; it takes its arguments optionally from either the command line, or, if none are found there, it prompts for the required data and accepts it from the terminal.

DFM09 is supplied with the step rate table set to the slowest rate for all drives. This allows any type of drive to successfully boot the system. Once the system has been booted, then SSTEPR should be executed to select the appropriate step rates for your system.

The program may be used as part of the START.UP execution file; when doing this, you should supply the parameters as part of the command line. Example: SSTEPR,F,F,S,S will cause DFM09 to use the fast step rate for drives 0 and 1, while the slow step rate is used for drives 2 and 3.

When using SSTEPR in the command line mode, the only characters that may appear in the command line after the SSTEPR are upper case F, upper case S, and delimiters. The presence of any other character, or a misplaced delimiter, will cause a syntax error to be reported, and the program will not alter the step rate table. If fewer than 4 drives are on the system, then only the specs for the available drives need be entered. Example: for a system with two SA400 drives in positions 0 and 1, use SSTEPR,S,S.

When using the prompting mode of SSTEPR, answer either S or F to the question for each drive present on the system. Where a drive does not exist on the system, respond with carriage return. After all the step rates have been entered, the program will list them back, and ask for confirmation. Answering Yes will accept the data as it is and alter the step rate table, then return to DOS09. Answering No will allow for reentry of all the rates, and confirmation again. This sequence may be repeated indefinitely.

The step rates available with SSTEPR are those that correspond to the two least significant bits of the FDC control register being equal to 0 or 3, corresponding to F or S. These correspond to 6/12 ms and 20/40 ms, respectively. The only drives being supplied at the present time that require the slow step rate are the SA400 type; all others should use the fast rate.

If an error is made such that the drives in the system are unable to respond to the step rates specified, then recovery is easily accomplished by rebooting and again running SSTEPR. The booting process always loads a fresh copy of DFM09, with its default lowest speed step rates which work with all drives.

VIEW VIEW,<FILE NAME>[,<LINE COUNT>]:

VIEW is a command for typing the contents of a text file. An optional line count is provided for CRT users and if specified will cause the typeout to pause every <LINE COUNT> number of lines.

General command format:

```
    VIEW,<FILE NAME>[,<LINE COUNT>]
```

If <LINE COUNT>, a hex number, is zero or not specified, the entire file will be typed without pausing. In response to a pause, a carriage return will terminate the listing; all other characters will cause the listing to continue. Note that the pause feature of the ESCAPE character within the system parameter will also function to halt the file viewing whether or not a line count is specified.

DOS09 COMMAND ERROR MESSAGES:

DOS09 prints only its prompt "DOS: " unless an error condition occurs. The following are error messages which can be generated by DOS09:

CMD NOT FOUND. DOS09 could not find the command as being memory resident or disk resident.

? DOS09 does not understand the syntax of the command entered. Correct the syntax and try again.

ILL FILE NAME. A file name was entered incorrectly. Try typing the line again.

NOT HEX NUM. An invalid digit was encountered in a hexadecimal number. Check the value and try again.

NO TA. No transfer address was found on the transient command or the file to be RUN. The file was loaded but DOS09 does not know where to begin execution.

CS ERR: XXXX. A checksum error has occured during the reading of a binary file. XXXX is the address of the object record being loaded. The file has been written on (most likely by someone trying to patch the file), or the file is not a binary file.

CLOSE ERR: XXXX. DOS09 has attempted to close a file left open by some program but the information in the File Control Block (FCB) needed to determine how to close the file is not valid, thus DFM09 cannot close the file. This is usually caused by a program corrupting the contents of the FCB. The only cure for this error is to cold start DOS09 (it may be advisable to reboot DOS09 since part of DOS09 may also have been corrupted by the offending program.

DISK ERR: XX. An error has been detected by the Disk File Manager (DFM). XX is the DFM error code indicating the nature of the error. Refer to the ERROR CODES in the DFM Programming Tables to interpret the error code.

WRITE PROTECT:

Each 5" diskette has a small rectangular cutout on one edge. Covering the cutout with a piece of OPAQUE TAPE will write protect the diskette from an accidental write. An attempt to execute any command that would write to the diskette will return an error message.

Some 8" diskettes also have a write-protect feature, however, its operation is just opposite that of the 5" diskettes. If the cutout is NOT covered, the diskette is write-protected. You must cover the cutout to have the disk operate normally. Not all 8" disks have a cutout. If the disk has a cutout, it is located towards the left on the edge of the diskette first inserted into the disk drive.

CREATING NEW DISKS:

This section describes how to build backup disks.

It should be noted that it is not necessary to have a copy of the DOS09 operating system on every disk used by DOS09. it is only necessary to have the system present on the system disk, typically the disk in drive 0.

STEP 1: FORMAT A DISK

The first step in building a new disk is to format a new disk. DOS09 is supplied with a command called "FORMAT". FORMAT is a program which initializes blank soft sectored diskettes (see the section on formatting new disks). Warning: FORMAT will completely destroy any information previously recorded on the diskette. Be sure that you want to format the diskette before doing so.

STEP 2: COPY ALL DESIRED FILES TO THE NEW DISK.

SINGLE DRIVE SYSTEMS:

To copy files between diskettes on a single drive disk system, a transient monitor command called "SDC" (Single Disk Copy) has been provided. SDC allows for the reading of a file into memory, giving the operator time to change diskettes, and writing the file back out to the new disk (see the transient command description of "SDC").

Repeatedly use SDC to copy all desired files from the old disk to the new disk.

Single disk system owners must have the DFM09 overlay files on all diskettes if the user is planning to use any of the random file features of DOS09, or the APPEND, DELete, or REName functions.

MULTIPLE DRIVE SYSTEMS:

On multiple drive systems the transient monitor command "COPY" is used to copy files from disk to disk. Alternatively, the BACKUP command will copy all files plus the boot-time linkage information to the copy of the master disk.

Assuming the newly formatted disk is in drive one and the old system disk is in drive zero, the following command will copy all the files from disk zero to disk one:

```
    COPY 0: 1:      or      BACKUP
```

NOTE: PREPARING SYSTEM DISKS

A "system disk" is a disk which contains at least the DFM overlay files, and possibly a LINKed copy of DOS09. The non-presence of the DFM overlays will generate a "non system disk" error if DFM requires a disk resident function handler.

If the disk being prepared is to be used to boot from a cold start, the disk must contain a copy of DOS09 and the disk must be "LINKED" by executing the following monitor command:

```
    LINK,1:DOS09.10     (Assuming the new disk is in drive one)
```

(the LINK command serves to tell the booting program which file is to be loaded and executed when the ROM booting routine is used.)

START UP:

The start up sequence is initialized during the "COLD START" of DOS09. The user may execute the start up sequence at any time by use of the procedures outlined below.

The general command sequence is:

```
     EXEC,START.UP
```

where EXEC is the command file processor and START.UP is the name of a text file containing the sequential commands necessary to configure the software to the system hardware.

DOS09 will present this command to the command interpreter each time the system is cold started. This implies that the utility EXEC and a file named START.UP exist on the system disk from which DOS09 is bootstrapped. If either file is missing, an error message will be displayed, but the system will still boot.

An example of a START UP file that will initialize the system was constructed utilizing the BUILD utility. The command sequence to build this follows:

```
     DOS: BUILD,0:START.UP
     #: RUN,PPRINT.SYS
```

```
#: SET,CRT=E008,DP=0,WD=0,EJ=0
#: (cr)
```

The BUILD will cause the file named START.UP to be created on the disk drive #0. The remaining commands perform:

1. RUN,PPRINT.SYS
   This command runs the file PPRINT.SYS on the work drive which will activate the system printer vectors for use by the P command. Since this file is run at system initialization the work drive is assumed to be drive #0.

2. SET,CRT=E008,DP=0,WD=0,EJ=0
   This command defines the CRT control port to be located at $E008 and disables all CRT formatting controls.

   More exotic START.UP command files may be devised at the user's option and installed on the system disk.

PART 4: DOS09 SYSTEM PROGRAMMER'S GUIDE:

INTRODUCTION

The Smoke Signal Broadcasting disk operating system, DOS09, consists of two distinct programs:

(1) The monitor portion (TMON), and
(2) The Disk File Management portion (DFM09).

The monitor portion of DOS09 handles all other functions of DOS09 not handled by DFM09.

DFM09 is strictly a Disk File Management system and can be used independently of the monitor portion of DOS09 when the user wishes to manipulate disk files.

The following sections provide the necessary interfacing information for the user to be able to make use of the functions available through DOS09.

SYSTEM EQUATE FILE:

DOS09 is supplied with a general system equate file named SQ.TXT. This file contains all the DOS09 equates necessary for system programs to interface with TMON and DFM09. Included are the equates for the monitor's entry points and parameter table, DFM linkages, DFM programming and return codes, and more. SQ.TXT is intended to be used by the system programmer as an equate file to be included in the file list when assembling your system programs. If you make a hardcopy listing of it, the file serves as a handy reference guide for programming DOS09.

MONITOR SYSTEM:

This section describes the user interface to the DOS09 monitor.

MONITOR ENTRY POINTS:

The first portion of the monitor contains a jump table for accessing several commonly used routines which are present within the monitor. The layout for the table is as follows:

ENTRY ADDR	ENTRY NAME	FUNCTION:
$D280	ZCOLDS	Monitor cold start
$D283	ZWARMS	Monitor warm start
$D286	ZOUTEE	Character output routine
$D289	ZINCH	Character input routine
$D28C	ZMON	JMP to ROM monitor
$D291	ZFLSPC	Get a file specification
$D294	ZGCHAR	Get current character from the line buffer
$D297	ZGNCHR	Get the next character from the line buffer
$D29A	ZANCHK	Check for alphanumeric
$D29D	ZDIE	Print command string, error message, and exit
$D2A0	ZGETHN	Get a hex value from the line buffer
$D2A3	ZADDX	Add the B register to the index
$D2A6	ZOUTST	Print a string
$D2A9	ZTYPDE	Type the disk error message
$D2AC	ZOUTHX	Print a byte in hex
$D2AF	ZOUTHA	Print an address in hex
$D2B5	ZLINEI	Input edited line from the terminal
$D2B8	ZLP	FORTRAN Line printer output vector
$D2BB	ZPEEK	Peek ahead at next char. in line buffer.
$D2BE	ZOUTCH	User alterable output vector.
$D2C1	ZPUTCH	Directed output vector.
$D2C4	ZGETCH	Input directed vector.
$D2C7	ZSTAT	Terminal input status
$D2CA	ZRESTR	Restore I/O vectors
$D2CD	DCMDLN	Call DO processor.
$D2D0	ZEXCMD	Execute command
$D2D3	ZLOAD	LOAD - File loader
$D2D9	ZNAMEJ	Decode name and jump.
$D2DC	ZCRLF	Print carriage return and line feed
$D2DF	ZSTEXT	Enter user default file extension.

ZCOLDS - Monitor cold start:

This entry to DOS09 resets the processor's stack and all internal status of both the monitor and DFM. The banner:

```
        DOS09 VX.Y
```

is printed on the terminal where VX.Y represents the version number. After initialization, the START.UP file is executed. DOS09 then proceeds to do a warm start. Note that when executing a cold start bootstrap, the boot routine in track 0, sector 0 transfers control indirectly through the transfer address; thus, DOS09's transfer address is ZCOLDS+1.

ZWARMS - Monitor warm start:

This entry to DOS09 resets the processor's stack, sets $DFBA to the address of 'ZWARMS' for subsequent restarting, sets the NMI vector to ZWARMS, closes any files that may have been left open, and then prompts the operator for a new command by typing the prompt 'DOS: '.

ZOUTEE - Character output to the control terminal:

This JMP is used for output to the user's terminal. This JMP is always set to use "OUTEEE" at [$F80A] within the resident ROM. It is assumed that the output routine preserves the D register and the X register.

ZINCH - Character input from the control terminal:

This JMP is used for input from the user's control terminal. This JMP is set to use "IN", [$F804], within the resident ROM. It is not possible to patch this address to refer to some other routine. User programs should use the ZGETCH routine, rather than calling ZINCH, because ZINCH does not check the SYSTEM PARAMETERS. It is assumed that the B and X registers are preserved and that the character input is returned in the A register. DOS09 assumes that "IN" does not automatically echo input back to the terminal.

ZMON - Jump to ROM monitor:

The monitor "EXIT" command uses this jump to give control to some other resident monitor. This jump is set to [$F802] to cause entry into the resident ROM, after setting up the stack.

ZFLSPC - Get a file specification:

"FILE SPEC" is a routine which is used to pick up a unit number and file name from the input buffer in the form:

[<UNIT NUMBER>:]<FILE NAME>[.<EXTENSION>]

The line buffer pointer is assumed to be pointing to the delimiter of the previous field. To use ZFLSPC, the X register must contain the address of a File Control Block (FCB) in which the unit and file name is to be put (see FCB format description). NOTE: ZFLSPC clears the FCB starting at FCB+0 through FCB+37 before picking up the file specification.

ZFLSPC returns with the carry set if an error occurs. If no error occurs, the FCB will have the properly set up unit number, file name and file extension if specified. If no drive number is specified the working drive number will be used. The A register will return with the delimiting character of the file name. If the file specification includes the command extension '$', the '$' extension is positioned within the FCB file extension, the next character/terminator is retrieved and the file name expansion is terminated.

The Y and U registers are preserved. The line buffer pointer is left pointing to the delimiting character.

ZGCHAR - Get current character:

This routine returns the character currently being pointed to by the line input buffer pointer.

NOTE: When control is given to a program, the line buffer pointer is pointing to the delimiter of the command name. Therefore, this routine must not be called until either ZGNCHR, ZFLSPC, or ZGETHN has been used since the line buffer pointer is initialized to point to the character preceding the line buffer.

ZGNCHR - Get the next character:

This routine advances the line buffer pointer by one and returns the character being pointed to. Once a carriage return character is returned, the pointer will no longer be advanced and carriage returns will be returned with each call. This routine exits through the ZANCHK routine thus returning the character classification in the carry bit.

ZANCHK - Alphanumeric check:

The character in the A register is checked for being $30-$39, $41-$5A, or $61-$7A. If the character is not within these inclusive limits, the carry bit will be set on return. Otherwise the carry will be cleared. The A, B, X, Y, and U registers are preserved.

ZDIE - Abort command and give error:

This routine prints the contents of the line buffer to the left of the line buffer pointer, followed by '?', followed by the text of an error message pointed to by the X register (printing stops when a null is found in the error message). After printing the error message, control will return to the monitor through the warm start entry.

ZGETHN - Get a hex number:

This routine returns the value of a HEX number found in the line buffer. ZGETHN starts by doing a ZGNCHR and continues to collect hex digits until a non-hex character is found. Upon return, the line buffer pointer is left pointing to the delimiting character, the value is returned as a 16 bit value in the X register, and if the carry is not set, indicating no error occured, the A register will contain the terminating character, and the B register will be non-zero if any hex digits were found.

ZADDX - Add the B register to the index register:

The 8 bit unsigned value in the B register is added to the value in the X register and the sum is returned in the X register. All registers except CC and X are preserved. This routine is included only for downward compatibility, since the ABX instruction performs this operation directly.

ZOUTST - Output a string:

The character string pointed to by the X register is output to the terminal. The output stops when a null, $00, is encountered. The X register is left pointing to the null, A contains 0, and all other registers (except CC) are preserved upon return.

ZTYPDE - Type disk error:

This routine is used to print the message "DISK ERROR: XX". The X register is assumed to be pointing to an FCB whose error status will be printed as the DFM error code "XX".

ZOUTHX - Output a byte in hex:

This routine prints two hex digits corresponding to the byte of memory pointed to by the X register. The A register is destroyed; all other registers (except CC) are preserved.

ZOUTHA - Type two bytes in hex:

This routine prints four hex digits corresponding to the two bytes of memory pointed to by the X register. The A register is destroyed, the B register is preserved, and the X register is returned advanced by one to point to the low order value printed. All other registers (except CC) are preserved.

ZLINEI - Line input routine:

This routine accepts a line of data from the terminal. The following characters have special meaning to the input routine:

CARRIAGE RETURN: Terminates the input
DELETE LINE (YDLINE): Restart line input
BACK SPACE (YBSCHR): Delete the previous input character

The edited information is put into the line input buffer and the buffer pointer is reset to point to the character position preceding the line buffer. A carriage return line feed pair is echoed upon receipt of the carriage return ending the input.

ZLP - FORTRAN Line printer output vector.

This entry point is reserved for the SSB FORTRAN line printer output vector. No other programs should use this vector.

ZPEEK - Peek ahead at next character.

This routine "peeks" ahead at the next character pointed to in the line input buffer, and returns it in the A register without causing the line buffer pointer to be advanced. Once a carriage return is returned, subsequent ZPEEK calls will continue to yield carriage returns.

ZOUTCH - Output character

The ZOUTCH routine usually does the same as ZOUTEE, however, ZOUTCH may be changed by programs to refer to some other output routine. For example, ZOUTCH may be changed to drive a line printer. ZOUTCH is called by ZPUTCH routine if the parameter 'YOSWT' is equal to 0. A warm start entry or a call to ZRESTR resets the ZOUTCH vector to the ZOUTEE vector and sets the YOSWT switch to zero.

ZPUTCH - Output character

This routine outputs a character in the A register to an output device, honoring system parameters as necessary. The column count is checked, and a new line is started if the current line is full. If an ACIA is being used to control the terminal it is checked for the entry of a BREAK character. If a BREAK character is detected, then the routine will jump to the address specified as the abort vector by the parameter 'YABORT'. If a BREAK character is not detected, then the terminal is checked for the entry of a STOP character. If a STOP character is detected, then the output will pause at the end of the line. During the end of line pause, the terminal is checked for the entry of the CONTinue character. If the CONTinue character is detected, then the output will be resumed. If the 'YOSWT' parameter is non-zero, the routine ZOUTEE is called to process the character (i.e., forces output to control terminal). If the 'YOSWT' parameter is zero, the routine ZOUTCH is called. Normally, ZOUTCH sends the character to the terminal via ZOUTEE. The user program may, however, change the address portion of the ZOUTCH entry point to go to another character output routine, such as a printer driver, etc.

ZGETCH - Get character

This routine gets a single character from the control terminal, and returns it in the A register to the calling program. The current line counter is cleared. Because this routine honors system parameter controls its use is preferred to ZINCH.

ZSTAT - Get terminal input status:

This routine checks the control terminal ACIA for data ready and returns the Data Ready status in the carry register. If the carry is set then data ready is set, if the carry is clear data is not ready. Note: if the parameter YCPORT in the system parameter table is zero, this routine will return a not ready status.

ZRESTR - Restore output vectors:

This routine forces the ZOUTCH jump vectors to point to ZOUTEE. The output switch 'YOSWT' is set to zero. The A, X, and CC registers are changed by this routine. All other registers are preserved.

DCMDLN - Do Command Line entry:

This routine is a reserved vector used by DOS09 for EXEC file control. No application program is authorized use of this vector location.

ZEXCMD - Execute command:

This entry point allows any user written program to pass a command string to DOS09 for processing. DOS09 will return control to the user program on completion only if the programs called by the command execute an RTS to exit. Otherwise, control may be returned to DOS09 as is normally done at completion of most utilities. The command string may be placed anywhere in memory and the line pointer (YLINAD) and reset pointer (YLINPT) established to point to the location preceding the first character of the command string. The command string must terminate with a carriage return and null ($0D00).

ZLOAD - Load file:

On entry, the X register must contain the address of the FCB which has been opened for reading the desired file. This routine is used to load binary files only, not text files. The file is read from the disk and stored in memory, normally at the load addresses specified in the binary file itself. It is possible to load a binary file into a different memory area by using the load address (YOFSET) location. The 16-bit value in the load address (YOFSET) location is added to the addresses read from the binary file. Any carry generated out of the most significant bit of the address is lost. The transfer address, if any is encountered, is not modified by the load address (YOFSET). Note that setting a value in the load address (YOFSET) does not modify any part of the content of the binary file. It does not act as a program relocator in that it does not change any addresses in the program itself only the location of the program in memory. On exit, the transfer address flag (YTAFLG) is zero if no transfer address was found. This flag is non-zero if a transfer address was encountered. The transfer address (YTADDR) location contains the last transfer address encountered. The file is closed on exit. If a disk error is encountered, an error message is issued and control is returned to DOS09 at the warm start entry.

ZNAMEJ - Decode command name and jump

This routine, when called with the address of a properly formatted user command table in the index register, will search the table for a match against a command name in the DOS FCB (DOSFCB). If a match is found, ZNAMEJ will call the command as a subroutine, and return with the carry bit clear. If a match is not found in the command table, ZNAMEJ will return with the carry bit set. You should refer to the section entitled USER COMMAND TABLE for the command table format that ZNAMEJ expects.

ZCRLF - Print a carriage return line feed:

This routine, in addition to outputting a carriage return and line feed, checks and responds to several parameter conditions. On entry, the routine checks for a STOP character having been entered while the line was being printed. If so, the routine waits for a BREAK character or a CONTinue character to be entered. If a BREAK character is entered, the routine jumps to the address contained in the YABORT vector of the parameter table. Unless changed by the user program, the YABORT vector is set to the DOS09 Warm Start entry point (ZWARMS).

However, if a CONTinue character is entered or it wasn't necessary to wait for one, the line count is checked. If the line count equals the last line of a page and the wait on end of page (WAIT) is on, the routine waits for a BREAK character or a CONTinue character, as above. Note that all waiting is done before the carriage return/line feed is printed. The carriage return/line feed are printed, followed by the number of nulls specified by null count (NULLS). If the end of page is encountered on entry to this routine, an 'eject' is performed by issuing additional carriage return, line feeds and nulls until the total number of blank lines specified by eject count (YEJECT) is reached.

ZSTEXT - Set default file name extension:

On entry the X register should contain the address of the FCB into which the default extension is to be stored if there is not an extension already in the FCB. The B register, on entry, should contain a numeric code indicating the extension type to be used. The codes and extension values set are described below. If there is already an extension in the FCB, this routine returns to the calling program immediately. The extension types, as referenced by the value in the B register, are:

0 - BIN	5 - BAK
1 - TXT	6 - DAT
2 - SRC	7 - FOR
3 - BAS	8 - TMP
4 - CTL	9 - null (spare)

It is important to note that (1.) Any extension as referenced by its extension type is alterable using the SET command, so if an extension has been re-defined, ZSTEXT will set the default extension as defined per extension type, and (2.) Any extension type other than the values shown above are ignored with the routine returning without setting an extension. All registers except CC are preserved.

USER COMMAND TABLE:

When a command line is processed by DOS09, the monitor resident command table is checked first. If the command is not found, then the user command table is checked. If the command is still not found, the disk directory is checked for the command. If still not found, "CMD NOT FOUND" is output to the terminal.

At UCTBL in the system parameter table are two locations which are normally zero indicating that there are no user resident commands present in memory. If these locations are not zero they are assumed to be the address of the user command table.

The format of the user command table is as follows:

```
START   FCB     5         Length of command (must be 1 - 6)
        FCB     2         Minimum number of characters which
                          must be entered by the operator
                          for a match
        FDB     CMDADR    Address of user command
        FCC     /MYCMD/   Text of command name
        FCB     0         0 means end of table
                          (table may contain any number of
                          entries)

        ORG     UCTBL     Tell DOS09 about this table
        FDB     START
```

In the above example, DOS09 will transfer to location 'CMDADR' if the operator types in any one of the following:

```
MY
MYC
MYCM
MYCMD
```

When control is passed to the user command, the input line buffer pointer is left pointing to the character delimiting the command name so that the user may request the monitor to pick up parameters from the command line (see the monitor jump table descriptions).

The user should exit his command processor by doing a jump to the monitor warm start entry point.

CREATING TRANSIENT MONITOR COMMANDS:

The file "SAVET" contains a program to facilitate the creating of new transient monitor commands.

Since the monitor 'SAVE' command is a transient program, it cannot be used to save a new transient routine (if the new routine resides in the Transient Command Area) since the SAVE command itself will be called into the Transient Command Area (TCA) destroying the program to be saved. Hence, 'SAVET' is a resident version of the transient SAVE command.

SAVET is used identically to the SAVE command. The only difference is that the SAVET program will reside at location $0100 rather than within the Transient Command Area.

NOTES ON MODIFYING DOS09 OR TRANSIENTS:

Executable object files are stored on the disk in a binary record format. This implies that if it is desired to patch an object file (such as DOS09 or the transient commands) the user should load the program into memory, make the changes, and then write a new file out to the disk. The user can directly modify the disk only if the checksum for the record being changed is also updated. In order to be able to do this the user must first read and understand the Motorola MINIBUG-II binary object record format since this is the format used by DOS09.

An alternative method of patching an object file is to append to the file a file containing object records which load the patches into place. When the object file is loaded, the original file will be loaded, the patches will be loaded, and once the end of file has been reached, DOS09 will then transfer to the starting address.

DISK FILE MANAGMENT SYSTEM - DFM09

This section is directed toward how to use DFM09 and how to interpret the disk structure used.

INTRODUCTION:

DFM09 is a disk file management program written for Motorola 6809 based microcomputers using Smoke Signal Broadcasting's disk controller.

DFM09 provides the interface between user programs and the disk hardware by maintaining the information necessary to allow the user to transmit data to and from disk files on a character-by-character basis.

By providing this interface, the user program need not be concerned with:

1) The actual mechanics of reading and writing the disk,
2) What files are on the disk and where they are located,
3) Allocation and de-allocation of disk space.

The user need only be concerned with:

1) The name of the file to be operated upon,
2) The operation to perform (e.g., read or write)
3) The physical drive upon which the file resides.

DFM STRUCTURE:

The user need not be concerned with the internal structure of DFM other than to understand its effect upon the operation of the system. To the user, DFM consists of three parts:

1) The kernel. The kernel portion of DFM interfaces user requests with the appropriate function processor. The kernel is always present in memory and contains common subroutines used by several request processors.

2) The resident function handlers. The frequently used function processors, and certain special function processors, are kept in memory for faster access.

3) The Disk-resident function handlers. Infrequently used function processors are kept in DFM overlay files on the system disk. These DFM overlay files are expected to exist on the currently "logged" system drive (see QLOGD and QLOGE functions; also see Appendix A for the system file names and which functions they contain). These system files need not be present on all disks; the system files need only be present on the current system drive when one of the functions they contain is to be used.

USING DFM:

All requests for services are communicated to DFM by means of a file control block (FCB). The FCB is a table in RAM memory which contains information such as the file name, operation to perform, unit number of disk for the file, and the disk I/O buffer space.

In order to operate on a file the file must must be "OPENED". Opening the file establishes the linkages to be able to transfer data to or from the file. Subsequent data transfers are then made by passing a byte of data through the A register. After all data transfers are complete, the file must be "CLOSED". Closing the file updates all information on the disk regarding the file.

FILE TYPES:

DFM supports sequential and random access file structures. Each file type has two subtypes. A sequential file may be either a binary file or an ASCII text file. Random access files may be either byte addressable or record addressable.

SEQUENTIAL FILE OVERVIEW:

The two types of sequential files are binary and ASCII text. DFM makes no assumptions as to the file's contents in binary mode and treats the file as a stream of 8-bit bytes. In ASCII text mode, however, DFM assumes that all characters are 7-bit ASCII characters and DFM will provide transparent blank compression thereby possibly reducing the overall file size for a typical text file.

RANDOM FILE OVERVIEW:

The two types of random files are byte addressable and record addressable. These two types exist to facilitate the access of data depending upon the application program.

Random files are treated as an ordered collection of bytes. In byte mode, the user can specify which byte of the file is be operated upon next by suppling a byte address (a number from 0 to one less than the file size). In record mode, the user specifies which fixed length record is to be operated upon next.

FILE CONTROL BLOCK FORMAT:

This section describes the entries within the File Control Block (FCB) used to access disk files and to communicate with DFM.

The FCB is a table 166 ($A6) bytes in length for sequential files and 320 ($140) bytes in length for random access files.

The user must allocate one FCB for each file being operated on at any one moment. There is no restriction upon how many FCBs may be in use at any time thus allowing the user to operate on as many files as desired from within any single program.

SEQUENTIAL FILE CONTROL BLOCK STRUCTURE

The format of the FCB for sequential files is as follows:

NAME		LOCATION		USAGE:
XFC	34	FCB+0		Function code
XES	35	FCB+1		Error status returned to caller
XUN	36	FCB+2		Unit number for operation
XFN	37	FCB+3		1st character of file name
	38	FCB+4		2nd character
	39	FCB+5		3rd character
	3A	FCB+6	+3	4th character
	3B	FCB+7		5th character
	3C	FCB+8		6th character of file name
	3D	FCB+9	+6	1st character of the file extension
	3E	FCB+10	A	2nd character of the extension
	3F	FCB+11	B	3rd character of the extension
XFT	C0	FCB+12	C	File type
XFS		FCB+13	D	File status
XFSU		FCB+14	E	Track # of first sector used by the file
		FCB+15	F	Sector # of the first sector used by the file
XLSU		FCB+16	[illegible]	Track # of the last sector used by the file
		FCB+17	[illegible]	Sector # of the last sector used by the file
XSUC		FCB+18	[illegible]	High order count of sectors used by the file
		FCB+19	[illegible]	Low order count of the sectors used by the file
		FCB+20	[illegible]	Reserved
		FCB+21	[illegible]	"
		FCB+22	[illegible]	"
		FCB+23	[illegible]	"
		FCB+24	[illegible]	"
		FCB+25	[illegible]	"
		FCB+26	1A	"
XNFP		FCB+27	1B	High order address of next FCB in active FCB chain
		FCB+28	1C	Low order address of next FCB in active FCB chain
XBI		FCB+29	1D	Index into data buffer
XCT		FCB+30	1E	Track # of current sector on disk
XCS		FCB+31	1F	Sector # of the current sector on the disk
		FCB+32	20	Reserved
		FCB+33	21	"
		FCB+34	22	"
		FCB+35	23	"
		FCB+36	24	"
		FCB+37	25	"
		(Disk I/O buffer begins with the next byte)		
XNT		FCB+38	26	Track # of next sector in the file
XNS		FCB+39	27	Sector # of the next secotr in the file
XPT		FCB+40		Track # of previous sector in the file
XPS		FCB+41		Sector # of the previous sector in the file
XSOD		FCB+42 - FCB+165		Data portion of disk sector

The FCB entries from FCB+3 through FCB+26 are the exact entries to be found in the disk file directory.

RANDOM FILE CONTROL BLOCK STRUCTURE

The format of the FCB for random files is as follows:

NAME	LOCATION	USAGE:
XFC	FCB+0	Function code (See note 1)
XES	FCB+1	Error status
XUN	FCB+2	Unit number
XFN	FCB+3	File name
	...	
	FCB+11	(Last character of file name)
XFT	FCB+12	File type
XFS	FCB+13	File status
XFSU	FCB+14	First sector used (track number)
	FCB+15	First sector used (sector number)
XLSU	FCB+16	Last sector used (track number)
	FCB+17	Last sector used (sector number)
XSUC	FCB+18	High order count of sectors used by file
	FCB+19	Low order count of sectors used by file
XRFS	FCB+20	Random file size high (See note 2)
	FCB+21	Random file size middle
	FCB+22	Random file size low
XRHBW	FCB+23	Highest byte written high (See note 2)
	FCB+24	Highest byte written middle
	FCB+25	Highest byte written low
	+26	Reserved
XNFP	FCB+27	Address of next active FCB
	+28	(low order address)
XRBA	FCB+29	Random file byte address high (See note 2)
	+30	" " " " middle
	+31	" " " " low
XRIM	FCB+32	Random file increment mode (Also called XRIF during random file creation)
XRID	FCB+33	Random file initialization data constant

(The remainder of the FCB is to be used only by DFM)

FCB+319 Last byte of FCB

NOTE 1: The random and sequential file FCBs are identical in function in bytes FCB+0 through FCB+19.

NOTE 2: In byte mode these three bytes are treated as one 24-bit binary logical byte number. In record mode, the high and middle order bytes are used as a 16-bit record number and the low order byte is a 0 through 255 byte offset within the record. This note applies to both file size (XRFS) and byte address (XRBA).

DISK FILE DIRECTORY:

The directory of files present on the disk begins in sector 1 of track zero. The format of the directory is as follows:

BYTE	USAGE:
0	Track # of next directory block (0 if end of directory)
1	Sector # of next directory block
2	Track # of previous directory block
3	Sector # of previous directory block

The four bytes above are then followed by five File Information Blocks (FIBs). A FIB is the information contained in the FCB entries from FCB+3 through FCB+26 with one exception in the case of the first directory block.

The first directory block (track 0 sector 1) only describes four files. The first FIB is used to point to the start and end of the list of available sectors on the disk. The format of this first FIB is as follows:

BYTE	USAGE:
4	MUST BE $FF
5 - 12	Don't cares ($FF)
13	Unused (0)
14	Disk type ←
15	Next available block track number
16	Next available block sector number
17	Last available block track number
18	Last available block sector number
19	High order count of available sectors
20	Low order count of available sectors

INTERFACING WITH DFM:

There are three entry points into DFM09; they are:

1) The DFM09 initialization entry point
2) The DFM09 closing entry point
3) The DFM09 I/O service request entry point

These three entry points correspond to three "JMP" instructions located in the first nine bytes of DFM09 (see the memory map for specific addresses).

DFM INITIALIZATION ENTRY POINT (ODFM):

DFM09 must be initialized before it can be used. Initializing DFM basically tells DFM that there are no files currently in use.

The entry point to initialize DFM is the first of the three jumps located in the beginning of DFM. There are no errors associated with initializing DFM since no disk operations are performed and since the function of this call is to reset all internal status flags within DFM09. Initializing DFM also logs drive 0 as as the system drive (see the QLOGD function description).

DFM CLOSING ENTRY POINT (CDFM):

When no further use is to be made of DFM09, DFM should be "closed". Closing DFM serves to close any open files which may not have been closed.

DFM is closed by calling the second of the three jumps located in the beginning of DFM. Errors are reported as follows: a "BNE" will branch if an error occured. If an error occured, the error type is returned in the A register (see Appendix C for the error types), the error number will be returned in the B register, and on type 1 and 3 errors, the X register will be pointing to the FCB which is in error.

DFM I/O SERVICE REQUEST ENTRY POINT (DFM):

All service requests made to DFM are handled by calling the third of the three jumps located in the beginning of DFM.

Information regarding the functions to be performed by DFM is passed to DFM in the FCB. The address of the FCB is loaded in the X register when DFM is called. All registers (except CC) are preserved by the call unless data is returned to the caller. The condition codes are not preserved; upon return from DFM, a "BNE" will branch if an error occurred. If an error occurs, the error status byte in the FCB will contain the error code, otherwise, the error status byte will be zero.

The following is a desciption of the actions of the various functions available through DFM. The function and error code values can be found in the DFM PROGRAMMING TABLES.

QFREE: REPORT FREE SPACE:

DFM09 will return the count of available sectors for the disk drive requested in the FCB. The high order binary count will be returned in the A register and the low order count in the B register.

QDEL: DELETE A FILE:

The file specified in the FCB is deleted from the disk directory and the sectors used by the file return to the list of available sectors. The same function code is used to delete both random and sequential files.

QREN: RENAME A FILE:

The file specified in the FCB will be renamed to the new file name specified in FCB+45 through FCB+53 unless the new file name consists of all nulls (00). The remaining bytes in the FIB will be changed to the contents of FCB+54 through FCB+68 unless FCB+54 (the new file name XFT) is zero. If the new file name is null and FCB+54 is zero, then the RENAME FUNCTION will not change the existing FIB on the disk, but will copy it into the FCB in the usual place (FCB+3 through FCB+26) effectively implementing a "LOOKUP" function.

The same function code will rename both random and sequential files.

QAPP: APPENDING TWO FILES:

The file whose name is specified in bytes FCB+45 through FCB+53 is appended to the file specified in the FCB and is then deleted from the disk directory. Both files are assumed to reside on the disk drive specified in FCB+XUN. The nulls filling the unused portion of the last block of the first file remain (i.e., the files are not compressed together).

Random files are not allowed to be appended to other random files or to sequential files.

QDIRI: DIRECTORY READ SETUP:

This command causes DFM to open the disk directory specified in FCB+XUN. Subsequent calls using the directory transfer code are then used to to pick up one File Information Block (FIB) at a time until an end-of-file condition occurs.

QDIRT: DIRECTORY TRANSFER:

The directory transfer command is used to retrieve a file name at a time from the directory. This function may only be used following a directory setup command or following another directory transfer command, otherwise unpredictable results will occur. The transfer command causes the next active FIB entry to be copied from the directory into bytes FCB+3 through FCB+26 (see directory format description).

QTYPE: EXAMINE DISK TYPE

The QTYPE command returns the disk type byte for the drive specified in the FCB. The caller need only fill in XFC and XUN; the value returned in accumulator A breaks down as follows:

BIT	IF=0	MEANING	IF=1
0	single sided		double sided
1	5 inch		8 inch
2	single density		double density
3	48 TPI		96 TPI
4		reserved	
5		reserved	
6		reserved	
7	not protected		disk is "MASTER" disk

QRAFC: READ ACTIVE FCB CHAIN:

The QRAFC command provides the user with a means of locating the FCBs that DFM considers to be "active". "Active" can be though of as meaning containing valid status information for accessing a file.

The caller places an ordinal number in FCB+XFN and calls DFM. DFM will return the FCB address in bytes FCB+XFN+1 and FCB+XFN+2. If the ordinal is 0 then DFM returns the first FCB address in the active FCB list; if the ordinal is 1 then DFM returns the second, and so forth. DFM will return an address of zero if the ordinal exceeds the number of active FCBs or if there are no active FCBs.

QLOGD: LOGGING A SYSTEM DRIVE:

DFM must be told on which disk to expect to find its overlay files if the drive is to be other than drive 0. On calling ODFM, DFM logs drive 0 as the default system drive. If it is desired to change the system drive, the user may do so be calling DFM with the contents of FCB+XUN equal to the new system drive number.

QLOGE: EXAMINE SYSTEM DISK LOG:

The user can determine which drive is the logged in as the system drive by the QLOGE function. The QLOGE function will return the system drive number in FCB+XUN.

QSSR: SINGLE SECTOR READ:

The single sector read command causes a specified sector to be read from the disk and placed in the FCB data buffer. NOTE: A 166 ($A6) byte FCB must used with this command.

The track number and sector number to be read is placed in bytes FCB+30 (XCT) and FCB+31 (XCS) respectively; the track number must be a value between 0 and 77 and the sector number a value between 0 and 26.

NOTE: The error code returned from this command is the actual value read back from the WD 1771 chip with the least significant bit being used to indicate a seek error in which case the error bits reflect the type I command error bits.

QSSW: SINGLE SECTOR WRITE:

The single sector write is the counterpart of the single sector read command (see above description).

WARNING: Single sector I/O allows any sector to be read or written. It is up to the user to know what he is modifying.

SEQUENTIAL FILE ACCESS

QSO4W: OPEN A SEQUENTIAL FILE FOR WRITE:

DFM09 creates the file to be written by the name specified in the FCB. An error will occur if the file already exists (a file cannot be automatically overwritten). The caller must specify the file name, unit number, and file type. If the file type (XFT) contains the code for a compressed (i.e. ASCII) text file (a file type of FTCS; See appendix for file type codes) then DFM will perform blank compression otherwise the file will default to the binary sequential file type. WARNING: set only those bits in the file type byte (XFT) which are specified in Appendix B (the other bits will be used in future versions of DOS09).

QSWRIT: WRITE TO A SEQUENTIAL FILE:

The byte of data passed in the A register is written to the file specified in the FCB used to open the file.

QSWC: CLOSE A SEQUENTIAL WRITE FILE:

The file specified in the FCB is closed. That is, the last buffer of data is written to the file and the directory entry for the file is updated. If the last buffer is not full, it will be padded with nulls. When the file is read back, an end-of-file condition will not occur until after the last character of the last buffer has been read.

QSO4R: OPEN A SEQUENTIAL FILE FOR READ:

The file specified in the FCB is opened for read. The caller need not worry whether the file being read is a compressed ASCII file or whether the file is a binary file. DFM will automatically expand the blanks which it compressed when the file was written if the file was written in the compressed mode. The caller need only supply the unit number and file name to open the file. The caller may examine the lower four bits of the file type (XFT) byte after the file has been sucessfully opened if it desired to know whether the file was written in the compressed mode.

QSREAD: READ FROM A FILE:

The next byte is read from the file specified in the FCB and returned to the caller in the A register. Note that the nulls padding the last block of the file will be returned to the caller as if they were originally written to the file. DFM will not inform the caller of an end-of-file condition until after the last byte has been read from the last block. Thus, it is advised to ignore nulls in text files and nulls following records in object files.

Also, note that it is not necessary to write a special end-of-file character out to a disk file. By examining the error status returned by DFM it is possible to determine the end-of-file by continued reading from the file until DFM returns an end-of-file error status code (see appendix for error status codes).

QSRC: CLOSE A SEQUENTIAL READ FILE:

DFM releases the linkages to the file being read. The FCB is now free to be used for another file.

RANDOM FILE ACCESS

QCRF: CREATING A RANDOM FILE:

Random files differ from sequential files in that random files are created by a separate request to DFM other than the request to open the file, and that random files are of fixed size specified by the caller at the time of creation. To create a random file the programmer must supply the following information:

1) The unit number on which to create the file is placed in FCB+XUN.

2) The name of the file is supplied in FCB+XFN and follows the usual DFM file naming rules.

3) The file type is supplied in FCB+XFT and must be either FTRB or FTRR (see DFM PROGRAMMING TABLES for file type codes). The file type will determine whether file positioning will be specified in the byte mode (FTRB) or the record mode (FTRR).

4) The size of the file to be created must be specified in the three bytes starting with FCB+XRFS. If FCB+XFT contains FTRB then the user must supply a three byte binary number representing the desired number of bytes to be in the file. If FCB+XFT contains FTRR then the first two bytes must specify the number of records desired and the third byte must be the record size (1 to 255 characters).

5) The data initialization flag (FCB+XRIF) must be non-zero to force the initialization of the file's contents to the value contained in FCB+XRID. If FCB+XRIF is zero then the contents of the file are indeterminate.

From this information, DFM allocates the file space and builds the structure used to rapidly access the random data file.

NOTE: The random file is not left in an "OPEN" state. The QORF (open random file) file command must be used in order to operate upon the file.

QORF: OPEN A RANDOM FILE:

In order to read or write a random file the file must be opened by this command (QORF). To open the file the user must supply the unit number on which to find the file and the file name. DFM will handle the byte or record mode addressing depending upon the file type information found in the directory for the file. Be aware that the random file may now be read or written and that random files require the use of a 320 ($140) byte long FCB.

The file is opened positioned to byte 0 (or record 0 byte 0) and the byte pointer increment mode is set to auto-incrementing.

The byte pointer incrementing mode (XRIM) determines what happens to the byte position within the file when a byte is read from or written to the file. If XRIM is positive and non-zero then the file pointer will be incremented after the next read or write so that the next byte of the file will be operated upon next. If XRIM is zero then the byte pointer is not modified following reads or writes. If XRIM is negative then the byte pointer is decremented after the byte is read or written.

Each time DFM's byte pointer is automatically incremented or decremented, DFM will update the contents of XRBA, the random file byte address (and it will update it corresponding to the file type, either record mode or byte mode). Thus, by reading XRBA the user can tell where DFM is positioned within the file.

QPRF: POSITIONING A RANDOM FILE:

The current position within the random file can be altered by the QPRF, position random file, command. The caller sets up the byte address, XRBA, to either the byte or record position, depending upon the file type, and then calls DFM to position the file. An end-of-file error (EEOF) will be returned if the byte address does not fall within the file size and the file pointer will not be modified.

Note that in a record addressable file, the position command can be used to position into the middle of a record by setting the third byte of XRBA non-zero to indicate which byte of the record to position to.

Also, note that positioning a file will not cause the disk head to move. Movement of the disk head is postponed until a disk read or write is required.

QRRF and QWRF: READING AND WRITING A RANDOM FILE:

Read (QRRF) and write (QWRF) function identically with the exception of the direction of the data flow and so will both be covered in this description.

The data to be read or written will be passed in the A register on calling or returning from DFM. The byte position of the file to be operated upon corresponds to the current byte address (XRBA). WARNING: If the user changes the contents of XRBA, the user must notify DFM of this change by using the position command to tell DFM to position to the new byte location.

DFM will adjust the current file byte position, and XRBA, as specified by the contents of the increment mode flag XRIM. XRIM positive means auto-increment, XRIM zero means non-incrementing, and XRIM negative means auto-decrementing.

QCLSRF: CLOSING A RANDOM FILE:

Random access files must be closed by the QCLSRF command when the user is finished with the file. Closing the random file will force the last record to be written out if the file has been written to and then the FCB will be released so that it may be reused.

XRHBW: HIGHEST BYTE WRITTEN:

DFM09 keeps track of the highest byte written within a random access file. This address is kept in the directory entry for the file in bytes 20 through 22 of the FIB (Bytes 23 through 25 of the FCB). These bytes contain the value needed to position the file to the highest addressed byte within the file that has previously been written. Note that, for record mode files, bytes 20 and 21 will contain the record number which contains the highest byte written and byte 22 will contain the byte offset into the record for the highest byte written.

During file creation, XRHBW is set to $FF, $FF, $FF to indicate that nothing has been written to the file. As with all other FIB contents, XRHBW is available in the FCB while the file is open.

QERF: RANDOM FILE EXPANSION:

DFM09 contains a convenient means for expanding a random access file once it has been created. The function code is named QERF. To expand the size of a random file, the programmer must supply the following information:

1) FCB+XUN must contain the unit number on which the file exists.

2) FCB+XFN must contain the name of the file to be expanded.

3) The new file size (not the amount of expansion) must be in XRBA (normally the file byte address). The new file size follows the same restrictions as creating a new file except that the record size need not be specified for a record mode file.

4) FCB+XRIF may be set non-zero if it is desired to have the newly allocated disk sectors initialized to a value passed in in FCB+XRID. Note that the previously unused bytes of the original file remain initialized as determined during the file creation, not as during expansion.

From this information, DFM09 allocates the required disk space to the file. The data in the original file is not modified in any manner.

USING THE DISK FILE MANAGEMENT SYSTEM (DFM):

DFM is that portion of DOS09 which relates to the usage of disk files. This section is provided as an introduction as how to use DFM to read and write disk files.

It is recommended that the reader first read through the descriptions of the functions provided by DFM. Then, after having read this section, read through several of the monitor transient command listings to see how the information presented below is implemented in practice.

HOW TO READ FROM A SEQUENTIAL FILE:

Reading from a file is done in three steps:

1) Opening the file for read
2) Reading from the file
3) Closing the file

Opening the file sets up the software linkages within DFM to enable the user to then request data be transfered from the file.

In order to open a file the user must reserve a 166 byte table in his program. This table is referred to as a File Control Block (FCB). The FCB is a table in which the pointers to the file being accessed are kept.

To read a file the user must fill in three entries in the FCB. These three entries are: a) the unit number on which to find the file, b) the name of the file, and c) the operation to perform on the file.

The function code to open the file for read is put in the first byte of the FCB. The unit number on which the file is to be found is put in the third byte of the table (the unit number may be 0, 1, 2 or 3). The nine bytes following the unit number are used to designate the file name. The first six bytes are treated as the file name while the last three are treated as the file extension.

The first portion of the FCB then looks like:

```
        FCB+0 (XFC)   Operation code
        FCB+1 (XES)   Error code returned to the user
        FCB+2 (XUN)   Unit number
        FCB+3 (XFN)   File name (1st character)
        FCB+4
          .
          .
          .
        FCB+8         File extension (1st character)
        FCB+9
        FCB+10        File extension (last character)
```

To actually open the file, load the X register with the address of the first byte of FCB and call DFM. Upon return, a "BNE" will branch if an error occured. If an error occurs at any time the non-zero code will be returned in the 2nd byte of the FCB. The simplest manner in which to handle errors is to request DFM to close all open files. This is done by calling "CDFM", the DFM closing entry point. See the description of this entry point to DFM for further details. Thus the following section of code represents how one would open a file for read:

```
ODFM    EQU     $D780       Open DFM entry point
CDFM    EQU     $D783       CLOSE DFM entry
DFM     EQU     $D786       DFM service request entry

        LDX     #FCB        Load the address of the FCB
        LDA     #QSO4R      Load the value of the "open for read"
                            function code
                            (see the appendices for the function
                            code values)
        STA     XFC,X       Store the function code in the FCB
        JSR     DFM         Ask DFM to open the file
        BEQ     FILOPN      Branch if no error ocured
        JSR     ZTYPDE      Ask the monitor to
                            type "DISK ERROR: XX"
        JSR     CDFM        Ask DFM to close all files
        JMP     ZWARMS      Restart the monitor

FILOPN  LDA     #QSREAD     Change the function code to read from
                            the file
        STA     XFC,X

* WE ARE NOW READY TO READ DATA FROM THE FILE
```

2) To read a byte from the file, load the X register with the address of the FCB and call DFM. DFM will return the next byte from the file in the A register. If an error occurs a "BNE" will branch if an error occured (end-of-file is treated as an error condition also). The following section of code may be used to read from a file:

```
        LDX     #FCB        Point to the FCB
        JSR     DFM         Ask DFM to read a byte
        BEQ     READOK      BRA if no error occured
* IF DESIRED, "LDA  XES,X" HERE WILL PICK UP THE ERROR CODE
* SO THAT THE PROGRAM MAY DETERMINE WHAT TO DO WITH THIS
* SPECIFIC ERROR.
        JSR     ZTYPDE      Ask the monitor to type
                            "DISK ERROR: XX"
        JSR     CDFM        Ask DFM to close any open files
        JMP     ZWARMS      Restart the monitor

READOK  EQU     *           At this point a byte has been
                            read from the file and is
                            in the A register.
```

3) After all the data has been read from the file, the file must be closed. Closing the file releases the FCB from its role as table of pointers to the file. Closing the file is accomplished as follows:

```
          LDX     #FCB        Point to the FCB
          LDA     #QSRC       Set the function code to "Read Close"
          STA     XFC,X       Save the function code
          JSR     DFM         Ask DFM to close the file
          BEQ     CLSOK       Branch if the file was sucessfully
                              closed
          JSR     ZTYPDE      Tell the monitor to type the disk
                              error code
          JSR     CDFM        Ask DFM to close any open files
          JMP     ZWARMS      Restart the monitor

CLSOK     EQU     *           At this point the file is closed
```

HOW TO WRITE (CREATE) A FILE:

Writing a file follows the identical sequence of operations as reading from a file with the following exceptions:

1) When opening the file, the function code is "QSO4W" to open the file for write instead of "QSO4R".

2) "QSWRIT" is used in place of "QSREAD" when preparing the file to be written.

3) When data is to be written to the file, the data must be in the A register when DFM is called.

4) When closing the file, "QSWC" is used instead of "QSRC".

(see the appendices for the function code values)

HOW TO USE A RANDOM ACCESS FILE:

1) Creating a random access file.

The follow segment of code illustrates the creation of a byte mode random access file.

```
DFM      EQU      $D786          DFM service request entry
XRFS     EQU      20             Offest into FCB to file size
XRIF     EQU      32             Offset into FCB to initialization flag
XRID     EQU      33             Offset to initializtion constant

* WE'LL ASSUME THAT THE FCB ALREADY CONTAINS THE
* FILE NAME AND UNIT NUMBER
         LDX      #5000          set low order 16 bits of file size
         STX      FCB+XRFS+1
         CLR      FCB+XRFS       clear the 8 high order bits of file size
         LDX      #FCB
         LDA      #FTRB          set up the file type (byte or record)
         STA      XFT,X
         LDA      #1             set flag to force file contents
         STA      XRIF,X         to be initialized
         CLR      XRID,X         initialize all bytes to 0
         LDA      #QCRF          set up command for DFM
         STA      XFC,X
         JSR      DFM            ask DFM to create the file
         BNE      ERROR          branch if an error occured

* THE FILE HAS BEEN CREATED

ERROR    JSR      ZTYPDE         type an error message
         JSR      CDFM           close any files which might be open
         JMP      ZWARMS         exit to the monitor
```

2) Opening a random access file

The following code segment will open a random access file either to be read or written:

```
* AGAIN WE'LL ASSUME THE FCB CONTAINS THE FILE
* NAME AND UNIT NUMBER
         LDX      #FCB
         LDA      #QORF          set up command for DFM
         STA      XFC,X
         JSR      DFM            ask DFM to open the file
         BEQ      OK             the file is ready to be accessed
* MAY WISH TO PERFORM SOME SORT OF ERROR RECOVERY AS ABOVE
* AT THIS POINT
```

3) Reading and Writing random access files

The following code segment will position the file to byte 4000 and write "ABC" to the file.

```
XRBA    EQU     29          Offset to the byte address

        LDX     #FCB        point to FCB used to open the file
        CLR     XRBA,X      high order address = 0
        LDD     #4000       mid & low order byte address
        STD     XRBA+1,X
        LDA     #QPRF       set up command to position the file
        STA     XFC,X
        JSR     DFM         position the file
        BEQ     POK         branch if positioned OK
PER     JSR     ZTYPDE      give an error message
        JSR     DFM         and quit
        JMP     ZWARMS

POK     LDA     #QWRF       perpare write function code
        STA     XFC,X
        LDA     #'A
        JSR     DFM         write a byte
        BNE     PER
        LDA     #'B
        JSR     DFM
        BNE     PER
        LDA     #'C
        JSR     DFM
        BNE     PER
* THE THREE BYTES HAVE BEEN WRITTEN
```

To read from the file, QWRF is replaced with QRRF and DFM will return the byte read in the A register.

[redacted]

PART 5: DISK SYSTEM HARDWARE

PREFACE

This section describes the characteristics of the SSB floppy disk controller board. This interface allows users of the Chieftain or other SS-50 bus 6809 micro- computer system to easily interface up to four 5" disk drives or four 8" disk drives. Either single or double sided disk drives may be used.

BOOT AND I/O ROUTINES

To facilitate disk I/O the monitor ROM includes disk firmware routines. The ROM contains all necessary I/O routines to read, write, seek, step and restore the disk drives. In addition, a disk boot routine has been included in the ROM.

DISK CONTROL

Disk control is achieved by the use of the Western Digital FD1771B-01 floppy disk controller IC. This IC controls read/write format, head step, seeking, and checks the write protect status of the disk. The read/write format can be programmed to many formats including IBM 3740 format. CRC generation and checking are also performed within the FD1771B-01 IC. Additional information on programming the floppy disk controller chip may be found in the Western Digital FD1771B-01 product guide.

FD1771B-01 CONTROL

All communications between the host system and the FD1771B-01 IC are through a 6821 PIA. Control line functions of the FD1771B-01 are handled by the A-PORT and other programmable control input/output pins of the PIA. Data to and from the floppy disk chip is transmitted through the B-PORT of the PIA.

DISK INTERFACE DESCRIPTION

Address decoding for the 6821 PIA is provided for by NAND gate U19 and the 9324 decoder, U6. The 74LS30 8-input NAND gate U19 generates the non-programmable address decoding of the address bits A5-A12 at U19-8 to the CS2- input at U18 pin 23. Register selection within the PIA is controlled by address lines A0 and A1 at U18 pins 36 and 35. The PIA is decoded to occupy address space $F77C through $F77F.

The PIA data inputs (U18,26-33) are tied directly to the the negative true system data buss. Care must be exercised in programming the PIA as the PIA expects to see positive true data at the system port. All control functions sent to the PIA should first be complemented by the controlling program. All control information received from the PIA should be interpreted as negative true data by the receiving program. The DAL- lines of the FD1771B-01 IC are tied to the B-PORT of the PIA at U18(10-17). Data to and from the FD1771B-01 on the DAL lines is defined as negative true data. As data is not inverted through the PIA, it is not necessary to invert data from or to the FD1771B-01 as is required with the control registers of the PIA. Data exchanges between the PIA and FD1771B-01 are controlled by the following control lines:

WRITE ENABLE

CB2 of the PIA is used to strobe information from the 6821 into the FD1771B-01. CRB bits 3-5 should be programmed to '101'. In this mode CB2 is cleared on the positive transition of the first 'E' pulse following a write 'B' data register operation and set high on the positive transition of the next 'E' pulse. During write operations the read flip-flop, U25, must be disabled by programming A-PORT bits 6 and 7 to 1 and 0 respectively.

REGISTER SELECT

A-PORT bits 0 and 1 drive the register select lines of the floppy disk controller chip. PA0 drives address line A0 and PA1 drives address line A1.

READ ENABLE

Reading information from the FDC is controlled by its read enable input at U17-4. This input is driven by the read enable flip-flop U25-6. Flip-flop U25 is used in two modes. Mode one is used when reading control registers in the FD1771B-01. Mode two is used when reading data from the disk. In mode one PA6 (U18-8) is programmed to '0'. PA6 low forces U25-6 to the low state thereby enabling information from the selected register onto the DAL lines (U17,7-14). This information can then be read in on the B-PORT of the PIA. In mode two PA6 and PA7 (U18-8,9) are programmed to '1's. When DRQ (U17-38) comes true and PA7 is true flip-flop U25 will be set on the positive transition on the 1 Mhz clock (U25-12). DRQ is also tied to the CA2 input of the PIA (U18-39). This programmable pin is programmed as an input triggered by a positive transition (CRA bits 3-5 =110). On the positive transition of DRQ, IRQA at U18-38 will be set low (IRQA reflects the status of the bit set by the positive transition at input CA2(U18-39)). The IRQA output is used to drive the K input for read enable flip-flop U25. As long as IRQA remains active low U25 will not be allowed to reset. U25 setting causes a byte of data from the floppy disk to be placed on the DAL lines (U17-7,14). When the processor detects CRA bit 6 set it can then read the byte of data on the DAL lines though the B-port data register. After the processor has read the data from the B-PORT the flag in the A-PORT control register is cleared by reading the A-PORT data register. Reading the A-PORT data register also deactivates IRQA. This allows U25 to reset thereby preparing the floppy disk chip for the next byte of data from the disk.

HEAD LOAD TIMING

Head load time of the SA400 minifloppy is approximately 75 milliseconds. The 9602 oneshot (U9) provides the delay signal required for proper operation of the FD1771B-01. The time delay generated prevents the floppy disk controller from reading or writing before the head has had time to settle.

DISK INTERFACE SIGNALS

DISK SELECT

During operation one of four disks may be selected at any one time. Disk select is controlled by PIA A-Port bits 4 and 5. These lines are connected to a 74LS138 (U14-1,2) which decodes the signals to select one of four drives. The disk select lines are buffered by the 7417 buffer located at U22. U22 provides the required drive capability needed to drive the disk interface buss.

SIDE SELECT

PIA Port A Bit 3 is buffered by U15 and provides the side-select output for use in double sided disk systems.

MOTOR ON

The motors on 5" drives are turned on as soon as the disk is selected and will stay on as long as the disk system is accessed. U8 is wired as a retriggerable one-shot and has a period of approximately 30 seconds. After the last head load, U16-2 goes high which reverse biases D1, allowing U8 to time out. U1-5 is an inverting buffer used to drive the MON- line of the SA400 drive. 8" drives use AC motors designed for continuous duty and operate at all times for fastest disk access.

One-shot U9 is used to provide a motor start delay and a head load delay. The motor start delay is disabled by U8-3 if the motor is already running.

WRITE DATA

The Write Data signal at U17-31 is buffered and inverted by U23-12. Write data on the disk interface buss is negative true data (WR DATA-).

WRITE GATE

The Write Gate signal at U17-30 is buffered and inverted by U23-9. Write gate (WR GATE-) along with WR DATA- controls writing of data to the selected disk.

STEP

The step pulse at U17-15 is buffered to the disk interface buss by U23-2 (STEP-). The step output provides the step instruction to the disk at a controlled rate. The step rate is programmed by the user. For the SA400 the step rate should be programmed to 40 msec per step. For additional information on step rates see the Western Digital FD1771B-01 product guide available from Western Digital, and the description of the SSTEPR command.

DIRECTION

The direction output of the FD1771B-01 (U17-16) is buffered by U23-5 (DIR-). For step-in (towards the disk hub) the direction line will be high. For step-out the direction line will be low (this level will be inverted on the buss).

TRACK 00

The TRACK 00 status of the selected disk is buffered by U24-2 and is ANDed to HEAD LOAD from the WD1771-1 (U17-28). This signal "fakes" the WD1771-1 into thinking it is on track 00. This allows the system to respond faster after a reset or on power up.

WRITE PROTECT

Write protect (WRT PROT-) is buffered by U24. Resistor R26 provides the required pullup. The write protect line reflects the status of the currently active disk. If the write protect hole in the disk is covered the write protect line will be low. The buffered write protect line U24-11 drives U17-36. Before doing any write operations the write protect line is sampled. If the line is low the write operations will be aborted by the controller chip.

READ DATA

READ DATA- is buffered by NAND gate U3-(1,2). It is then sent through a one-shot to shape the signal. From here there are two options: Option 1) is to use the external data separator, and option 2) is to use the data separator internal to the FD1771-1. The FDC board is configured to operate with the external data separator. The internal separator may be selected by cutting both traces labeled J-12 and inserting jumpers at both locations labeled J-13. The difference between the two separators is one of resolution; The external separator is better at rejecting jitter from an SA400, and thus it is recommended that the external data separator be used.

The external data separator consists of IC'S U1, U2, U3, U4, U11, and U16. The separator works by generating "windows" through which data and clock pulses are gated from read DATA- to the

FD1771-1. The synchronization of the separator to the incoming data is done by retriggerable one-shot U11, associated gates, and flipflops. The one-shot timing is controlled by potentiometers R18 and R19 which are set at the factory and should not be adjusted in the field.

INDEX PULSE

The index pulse is buffered by AND gate U24-9,10 (INDEX PULSE-). Pullup is provided by resistor R25. The buffered index pulse signal at U24-8 drives U17-35. This signal provides sychronization information for the floppy disk chip.

INSTALLING ADDITIONAL DRIVES

Additional minifloppy disk drives may be installed in the field. To install a second or third drive, proceed as follows:

1) Locate the drive select jumpers located in a dip socket on the top corner of the board on the disk drive.

2) The jumpers are cut as shown in the table below for SA400 drives:

DRIVE	CUT	REMOVE
0	MX, DS2, DS3	
1	MX, DS1, DS3	RESISTOR PACK 760-3-R150 OHM
2*	MX, DS1, DS2	RESISTOR PACK 760-3-R150 OHM

The jumpers are cut as shown in the table below for B51 drives:

DRIVE	CUT	REMOVE
0	DS2, DS3, MUX, HM	
1	DS1, DS3, MUX, HM	RESISTOR PACK 760-3-R150 OHM
2*	DS1, DS2, MUX, HM	RESISTOR PACK 760-3-R150 OHM

* A fourth drive (accessed as unit #3) may be installed on J4 by jumpering it the same as unit #1 on J2. Or, drives 0 and 1 may be connected to J2 and 2 and 3 connected to J4. In that case, the drives on J4 should be jumpered the same as the drives on J2.

NOTE: The resistor pack is removed from all drives except the drive which is on the end of the ribbon cable connecting to the controller (normally this is drive zero).

To install additional SA800 drives, the jumper on the back of the SA800 printed circuit board near the cable edge connector should be moved as shown below:

DRIVE	MOVE JUMPER TO
0	DS1
1	DS2
2	DS3
3	DS4

DISKETTE REQUIREMENTS

The Smoke Signal Broadcasting disk systems use standard size media with one index hole. For maximum flexibility in adapting our system to special user requirements, we use a soft-sectored disk format. Thus, diskettes designed for the specialized requirements of hard- sectored systems such as the Northstar which use multiple index holes will not work with the our disk systems. If you inadvertently try to format a multiple index hole diskette, the formatting program will report a very large number of "bad sectors".

ADJUSTMENTS FOR 5" OR 8" DRIVES

The SSB disk controller board can be used with either 5" or 8" drives, but not both. The proper PC jumpers are installed at the factory and the data separator timing adjusted for the type of drive shipped with the system. It is recommended that modifications required to make the controller operate with a different size disk drive than supplied with original system be made at the factory.

PART 6: GENERAL INFORMATION

LIMITED WARRANTEE

Smoke Signal Broadcasting guarantees (to the original purchaser) its disk system hardware for a period of 90 days from date of purchase. Smoke Signal Broadcasting will, at its option, repair or replace any disk system with a hardware defect returned to it postpaid within 90 days from date of purchase, provided that, in its opinion, the defect was not caused by improper handling or improper connection to the host computer or a malfunction of the host computer. Shipping charges for return of the repaired unit to the purchaser shall be paid by Smoke Signal Broadcasting. The liability of Smoke Signal Broadcasting is specifically limited to repair or replacement of the hardware and shall not extend to consequential or incidental damages suffered by the user; nor shall Smoke Signal Broadcasting be liable for any representation as to the suitability of the its disk systems to any particular user application unless such representation is in writing and signed by an officer of Smoke Signal Broadcasting.

In the event of a problem during the warrantee period:
1) We suggest that you call first and explain your problem. Technical personnel are available from 9 AM to 5 PM local California time at (213) 889-9340. Many times problems can be solved quickly over the phone, thus, saving you time.
2) If it is necessary to return your unit for repair, send it to:

   Smoke Signal Broadcasting
   31336 Via Colinas
   Westlake Village, CA 91361

3) Be sure that the unit is packed adequately and that a brief explanation of the problem is enclosed with the unit.
4) Be sure to include your return address and a phone number where you can be reached during business hours.

Some states do not permit the limitation or exclusion of incidental or consequential damages. In those states this limited warrantee is not valid and the system is sold AS IS. See our repair policy.

While in the interest of good customer relations, Smoke Signal Broadcasting will attempt to correct any software errors brought to its attention, the software is provided AS IS without warrantee.

This warrantee is in lieu of all other warrantees expressed or implied.

REPAIR POLICY

In most cases, repairs will be made within 7 days of receipt. No charge will normally be made for repairs to units returned to us within 90 days of purchase even in states where the limited warrantee does not apply. This should be construed only as a statement of policy and not as a guarantee or legal obligation to make such repairs.

After 90 days from date of purchase, repairs will be made according to a flat rate repair schedule unless the unit has been subject to physical damage or connected to improper voltages. The current charge for repairs to a disk system is $95.00. You pay the shipping charges to us, we pay the return charges. If outside the United States, these provisions do not apply and you should contact us for instructions. Generally, you will be referred to a repair facility in your country since customs clearance charges run about $100. This is in addition to shipping charges.

SOFTWARE LICENSE

The purchaser of a disk system purchases, in addition to the hardware, a license for the limited use of the DOS09 software supplied with the system. This license allows the purchaser to use the software on any disk system manufactured by Smoke Signal Broadcasting and to make copies of the software for use on any disk system manufactured by Smoke Signal Broadcasting. Use of the software on any other disk system or the copying of the software for any other use is a violation of this license unless specific written approval for other uses has been obtained from an officer of Smoke Signal Broadcasting.

USER GROUP INFORMATION

Smoke Signal Broadcasting operates a 6809 program users group. Purchase of a SSB disk system and return of the warrantee registration form entitles the user to a one year membership to the users group. The purpose of the group is to provide a low-cost program exchange service to group members. We do not intend the users group to become a profit center for Smoke Signal Broadcasting, however, we will attempt to recover the direct expenses of program duplication, advertising of the user's group and of employees assigned to user group projects.

To help us meet the goal of a low cost program exchange service, we would appreciate the contribution of all types of programs for 6809 based systems - not necessarily disk based systems.

We are particularly interested in additional transient commands for our disk systems. If everyone will share with us the programs they have created to make the operation of their disk system more convenient to them, it will quickly enhance the value of all our systems.

It is hoped that shorter programs, 500 bytes or so, will be contributed without charge. For longer programs, where the contributor needs to recover some of his development costs, a royalty will be paid. Large general purpose programs (BASIC, FORTRAN, editors, etc) will be extensively advertised to insure wide distribution, low cost to the user, and reasonable compensation to the program author.

We believe that the SSB disk systems are by far the best disk systems available to the microcomputer user today. Your support of the users group will enable us to provide evolutionary changes to the system that will keep it the leader in microcomputer disk systems.

D795-[illegible] DRIVE "SIZE" TABLE

APPENDIX A

DOS09 VERSION 1 MEMORY MAP

$C000 - $C87F	DFM program area
$C880 - $C9BF	DOS FCB area
$C9C0 - $C9FF	Machine stack area
$CA00 - $CCBF	DOS (TMON) program area
$CCC0 - $CCFF	Printer driver area
$CD00 - $CD7F	DOS (TMON) line input buffer
$CD80 - $D27F	Transient command area (TCA)
$D280 - $D77F	DOS (TMON) program area
$D780 - $DF7F	DFM program area
$DF80 - $????	Stack area for ROM monitor
$3E00 - $3FFF	EXEC program area. Used only when EXEC is active.
$5E00 - $5FFF	Optional EXEC program area
$7E00 - $7FFF	Optional EXEC program area
$BE00 - $BFFF	Optional EXEC program area
$0000 - $3FFF	Required by system programs FORMAT, SAVET, COPY, and BACKUP. Otherwise is available to user.

ENTRY POINTS

$D280	Cold start entry into DOS
$D283	Warm start entry into DOS
[$F81E]	Cold start boot entry (indirect)

APPENDIX B

MONITOR ENTRY POINTS

ENTRY ADDR	ENTRY NAME	FUNCTION:
$D280	ZCOLDS	Monitor cold start
$D283	ZWARMS	Monitor warm start
$D286	ZOUTEE	Character output routine
$D289	ZINCH	Character input routine
$D28C	ZMON	JMP to ROM monitor
$D291	ZFLSPC	Get a file specification
$D294	ZGCHAR	Get current character from the line buffer
$D297	ZGNCHR	Get the next character from the line buffer
$D29A	ZANCHK	Check for alphanumeric
$D29D	ZDIE	Print command string, error message, and exit
$D2A0	ZGETHN	Get a hex value from the line buffer
$D2A3	ZADDX	Add the B register to the index
$D2A6	ZOUTST	Print a string
$D2A9	ZTYPDE	Type the disk error message
$D2AC	ZOUTHX	Print a byte in hex
$D2AF	ZOUTHA	Print an address in hex
$D2B5	ZLINEI	Input edited line from the terminal
$D2B8	ZLP	FORTRAN Line printer output vector
$D2BB	ZPEEK	Peek ahead at next char. in line buffer.
$D2BE	ZOUTCH	User alterable output vector.
$D2C1	ZPUTCH	Directed output vector.
$D2C4	ZGETCH	Input directed vector.
$D2C7	ZSTAT	Terminal input status
$D2CA	ZRESTR	Restore I/O vectors
$D2CD	DCMDLN	Call DO processor.
$D2D0	ZEXCMD	Execute command
$D2D3	ZLOAD	LOAD - File loader
$D2D9	ZNAMEJ	Decode name and jump.
$D2DC	ZCRLF	Print carriage return and line feed
$D2DF	ZSTEXT	Enter user default file extension.

APPENDIX C

DOS09 PARAMETER TABLE

```
YMONV  EQU PARTBL+$00   DISK MONITOR VERSION
YMEMAX EQU PARTBL+$02   USER MEMORY LIMIT
YLINAD EQU PARTBL+$04   LINE BUFFER ADDRESS
YLINPT EQU PARTBL+$06   LINE BUFFER RESET ADDR.
YBSCHR EQU PARTBL+$08   BACKSPACE CHARACTER
YDLINE EQU PARTBL+$09   DELETE LINE CHARACTER
YLPAUS EQU PARTBL+$0A   LINE PAUSE CHAR
YLCONT EQU PARTBL+$0B   LINE PAUSE RESUME
YABORT EQU PARTBL+$0C   ABORT CHARACTER
YABRTV EQU PARTBL+$0D   ABORT VECTOR ADDRESS
ZHCINT EQU PARTBL+$0F   HARD COPY INITIALIZE
ZHCOUT EQU PARTBL+$12   HARD COPY CHAR OUTPUT
YCPORT EQU PARTBL+$17   CONTROL I/O PORT 0=DISABLED
YPPORT EQU PARTBL+$19   PRINTER I/O PORT 0=DISABLED
YDEPTH EQU PARTBL+$1B   LINES/PAGE
YWIDTH EQU PARTBL+$1C   CHARACTERS/LINE
YNULLS EQU PARTBL+$1D   CR/LF NULLS
YHCFLG EQU PARTBL+$1E   HARD-COPY ENABLE
YEJECT EQU PARTBL+$1F   BLANK LINES END OF PAGE
YPPPAS EQU PARTBL+$20   POST PAGE-PAUSE,0=ON.0=OFF
YSYSDR EQU PARTBL+$21   SYSTEM DRIVE NUMBER
YWRKDR EQU PARTBL+$22   WORK DRIVE NUMBER
YCLINE EQU PARTBL+$23   CURRENT LINE NUMBER
YCCOL  EQU PARTBL+$24   CURRENT CHAR POSITION/COLUMN
YUCSWT EQU PARTBL+$25   UPPER CASE SHIFT LOCK 0=>NOT UC
YOSWT  EQU PARTBL+$26   OUTPUT CONTROL SWITCH    ZERO = ENABLE
YDCMDA EQU PARTBL+$27   DO COMMAND PROCESSOR ACTIVE
YERSWT EQU PARTBL+$28   SYSTEM ERROR SWITCH
YCFLG  EQU PARTBL+$29   COMMAND LOAD FLAG
YLOADE EQU PARTBL+$2A   LOAD ERROR FLAG
YTAFLG EQU PARTBL+$2B   VALID TRANSFER ADDRESS FLAG
YTADDR EQU PARTBL+$2C   TRANSFER ADDRESS
YOFSET EQU PARTBL+$2E   OFFSET FOR FILE LOAD
YMSTAT EQU PARTBL+$30   MONITOR STATUS BYTE
YDATE  EQU PARTBL+$49   SYSTEM DATE STRING
YTIME  EQU PARTBL+$59   SYSTEM TIME STRING
EXTTBL EQU PARTBL+$62   FILE EXTENSION TABLE
```

APPENDIX D

DOS09 COMMAND ERROR MESSAGES

DOS09 prints only its prompt character unless an error condition occurs. The following are error messages which can be generated by DOS09:

"CMD NOT FOUND" DOS09 could not find the command as being memory resident or disk resident.

"?" DOS09 does not understand the syntax of the command entered. Try again.

"ILL FILE NAME" A file name was entered incorrectly. Try typing the line again.

"NOT HEX NUM" An invalid digit was encountered in a hexadecimal number. Check the value and try again.

"NO TA" No transfer address was found on the transient command or the file to be RUN. The file was loaded but DOS09 does not know where to begin execution.

"CS ERR: XXXX" A checksum error has occured during the reading of a binary file. XXXX is the address of the object record being loaded. The file has been written on (most likely by someone trying to patch the file), or it is not a binary file. The file should be deleted and replaced with a backup copy, or the correct binary file should be specified.

"CLOSE ERR: XXXX" DOS09 has attempted to close a file left open by some program but the information in the File Control Block (FCB) needed to determine how to close the file is not valid, thus DFM09 cannot close the file. This is usually caused by a program corrupting the contents of the FCB. The only cure for this error is to re-boot DOS09. This is because part of DOS09 may also have been corrupted by the offending program.

"DISK ERR: XX" A Disk File Manager (DFM) error has occurred. XX is the DFM error code indicating the nature of the error. Refer to the ERROR CODES in the DFM Programming Tables to interpret the error code.

APPENDIX E

DFM PROGRAMMING TABLES

DFM ENTRY POINTS

FUNCTION	NAME	ADDRESS
DOS BASE ADDRESS		$C000
OPEN DFM	ODFM	$D780
DFM I/O	DFM	$D786
CLOSE DFM	CDFM	$D783

DFM FUNCTION CODES

CODE NUMBER	CODE NAME	DEFINITION
0	QFREE	* Report free space on disk
1	QSO4W	* Open a sequential file for write
2	QSWRIT	* Write data to sequential file
3	QSWC	* Close a sequential file for write
4	QSO4R	* Open a sequential file for read
5	QSREAD	* Read data from sequential file
6	QSRC	* Close a sequential file for read
7	QDEL	Delete a file
8	QREN	Rename a file
9	QAPP	Append two files
10 A	QDIRI	* Open a disk directory
11 B	QDIRT	* Retrieve a file name from directory
12 C	QTYPE	* Retrieve disk type byte
13 D	QRAFC	* Read active FCB chain
14 E	(reserved)	
15 F	(reserved)	
16 10	QLOGD	* Log in a system disk
17 11	QLOGE	* Examine logged drive number
18 12	QSSR	* Single sector read
19 13	QSSW	* Single sector write
20 14	QCRF	Create random access file
21 15	QORF	Open random access file
22 16	QPRF	* Position random access file
23 17	QRRF	* Read from random access file
24 18	QWRF	* Write to random access file
25 19	QCLSRF	Close random access file
26 20	(reserved)	
27 21	(reserved)	
28 22	QERF	Expand random access file

* means the function processor is memory resident

The disk resident functions are located in 2 system overlay files. These files must exist on the system disk for the disk resident DFM functions to be executed.

DFM OVERLAY FILE	FUNCTIONS:
DFM090.??2	QDEL, QREN, QAPP
DFM090.??3	QCRF, QORF, QCLSRF, QERF

Where ?? represents the revision number of DFM

DFM ERROR CODES

ERROR NO.	ERROR NAME	EXPLANATION
$01	EIFC	Invalid DFM function code
$02	EFE	File already exists
$03	EFIB	Master file directory error
$04	EFB	File in use
$05	ENSF	File not found
$06	EEOF	End of file
$07	EDF	Disk full
$08	EIFN	Invalid FCB address
$09	EIFN	Illegal file name
$0A	EFS	File status error
$0B	EITS	Invalid track or sector number o
$0C	EIUN	Illegal unit no.
$0D	----	Unused, better left alone
$0E	EDR	Disk read error
$0F	EDW	Disk write error
$10	EIFT	Illegal file type
$11	ENER	Not enough room to create file
$12	EWP	File is write protected
$13	EDP	File is delete protected
$14	EFSE	Random file size error (size=0 or too big)
$15	EDWP	Disk is write protected
$20	ENSD	Non-system disk in logged drive
$21	ESFF	System file format error (should not occur)
$22	ECSS	Checksum error on system file

DOS09 CLOSE ERROR CODES

01XX	Illegal file activity. The file is not open for sequential read,write, or random accessing.
02XX	Unused.
03XX	File closing error - XX = DFM error code.
04XX	Read error on free chain descriptor. XX = 1771 Controller error status.
05XX	Write error on free chain descriptor. XX = 1771 Controller error status.

The following table enumerates the file type codes. These are the only valid file type codes which can appear in the lower four bits of the file type (XFT) in FCB's and FIB's.

FILE TYPE CODES

NAME	VALUE	TYPE
FTCS	1	Sequential compresses ASCII data.
FTSQ	2	Binary sequential.
----	3	(reserved)
FTRB	4	Byte mode random access.
FTRR	5	Record mode random access.
----	6-7	(reserved)
----	8-15	(unused)

The following table enumerates the file status codes. These are the only valid status codes which can appear in the lower four bits of the file status (XFS) in both FCB's and FIB's.

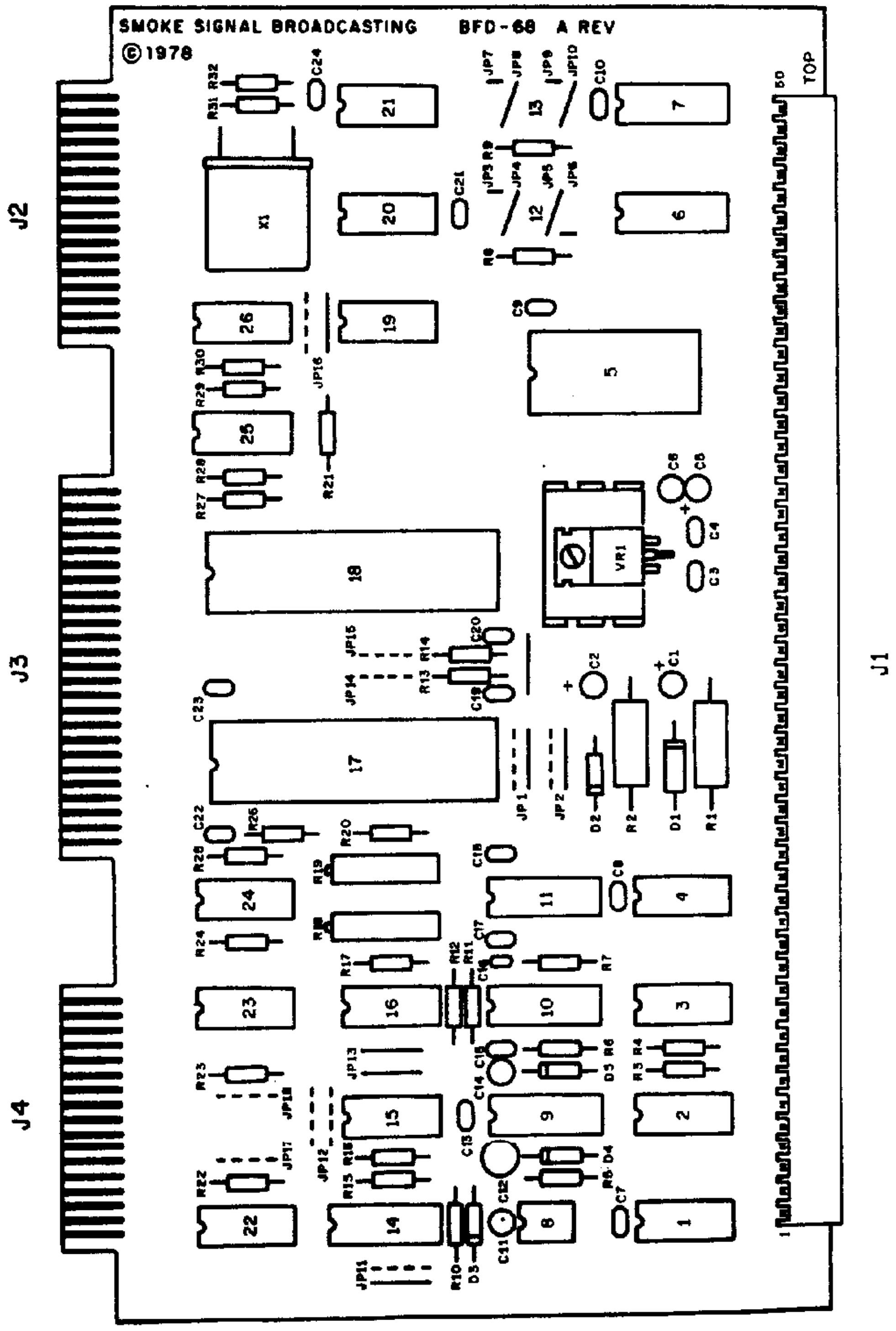
FILE STATUS CODES

NAME	VALUE	TYPE
FANA	0	Not active
FASR	1	Sequential read status.
FASW	2	Sequential write status.
FARA	3	Random access status.

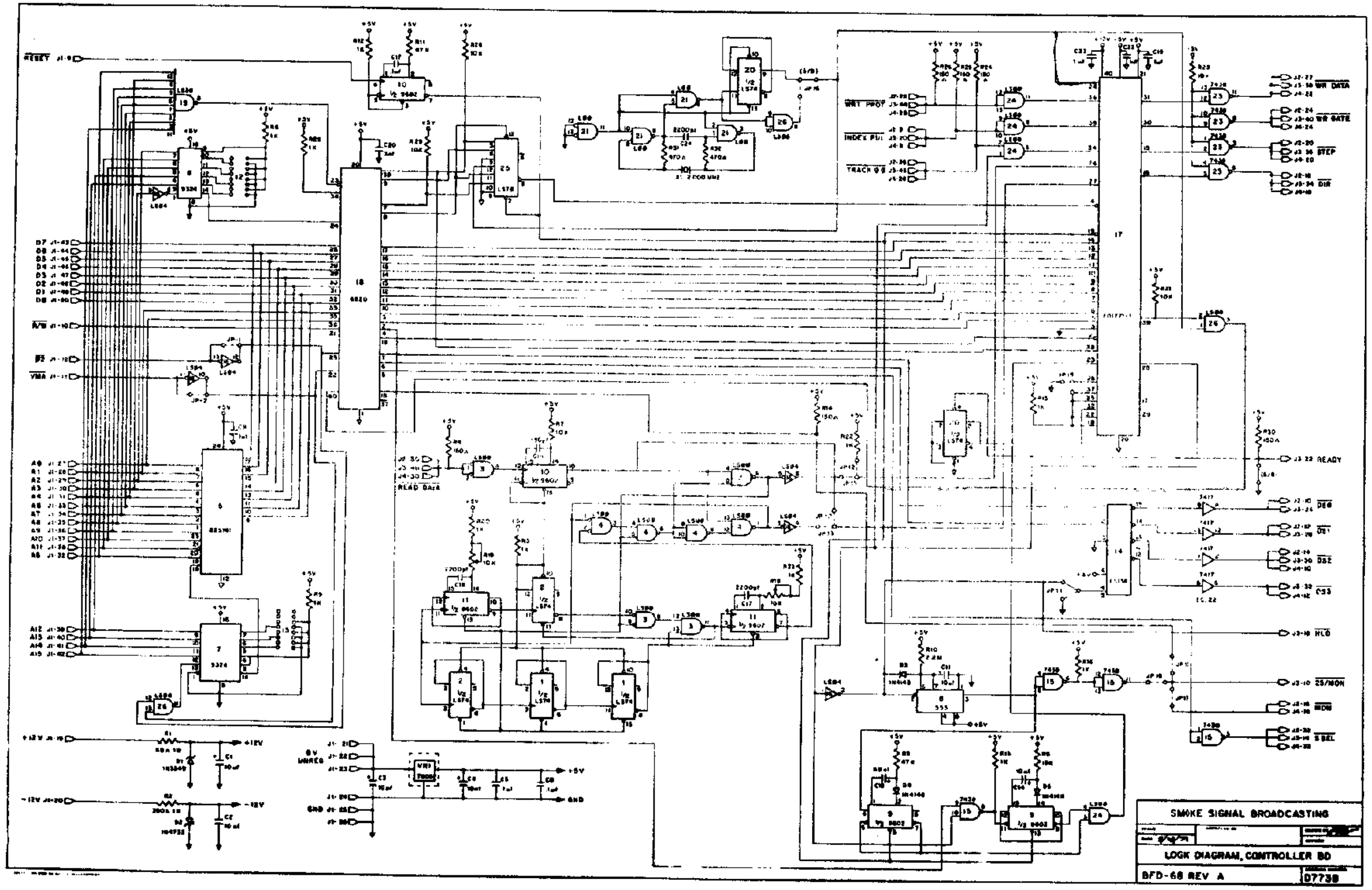
EXTENSION TYPE CODES

TYPE	INITIAL DEFINITION	USE
0	BIN	Default extension for Binary object files created by the Assembler.
1	TXT	Default extension for Editor files and Assembler input files.
2	SRC	A source file.
3	BAS	A BASIC program.
4	CTL	Default extension for control or procedure files that are processed by EXEC, and procedure files that are created by BUILD.
5	BAK	Reserved extension to be used only by the Editor for generating backup files.
6	DAT	A data file.
7	FOR	A FORTRAN program.
8	TMP	Reserved extension to be used only by the Editor for naming temporary work files.
9	null	Spare - user definable.

CONTROLLER PARTS LOCATION



1528



SMOKE SIGNAL BROADCASTING

9/4/79

LOGK DIAGRAM, CONTROLLER BD

BFD-68 REV A

D7738