31336 VIA COLINAS, WESTLAKE VILLAGE, CA 91361

SMOKE SIGNAL BROADCASTING

DISK SYSTEMS

SSB DISK SYSTEM REFERENCE MANUAL

PART 0: 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 6800 computer or its functional equivalent. The system should contain:
   a) 4K of RAM memory at $7000 to run DOS68 Version 3, or 6K of RAM at $6800 for DOS68 Version 4.
   b) Nothing that would interfere with the controller PROM at $8020 or PIA at $9FFC.
2. A source of 115 volts AC power.

MINIMUM CLOCK FREQUENCY

The minimum processor clock frequency for operation of the minifloppy systems is 0.85 MHz. For the 8" systems, the minimum clock frequency is 1.65 MHz. Most 6800 computers now in use operate with a clock frequency near 1.0 MHz. Thus, unless you are using one of the SSB Chieftain Series of 2 MHz processors, 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 all running nearly twice as fast.

If you are converting an existing 1 MHz system, a 1.8 MHz clock frequency is recommended. Most likely the MOS integrated circuits supplied with your 1 MHz system will operate satisfactorily at 1.8 MHz. If you experience problems, however, all the 6800 series components are available in a "B" series which is guaranteed to operate at 2 MHz. The 6800 may be replaced by a 68B00, a 6820 or 6821 by a 68B21 etc.

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-16A 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 a M-16A.

The SWTP MP-A CPU board may be converted to 1.8 MHz operation as follows:

(1) Cut the trace between pin 5 of IC-20 and pins 12 and 13 of IC-19.
(2) Connect a jumper from pin 3 of IC-20 to pins 12 and 13 of IC-19.

The SWTP MP-A2 board may be modified for 1.8 MHz operation by changing capacitor C-1 to approximately 30 pf. The exact value should be determined experimentally and a low temperature coefficient capacitor should be used to minimize frequency change caused by changes in operating temperature.

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".
5. 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 "BOOTING DOS68".

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 diferences. 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. Consequently, the interface

using MPI drives will not work with the SA400 drives due to the faster stepping speed. However, the B51 drives will work with an interface using the SA400 drives.

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 any 8" system will operate with both the SA800 and SA850 without modification.

WHICH DOS68 TO USE

The disks supplied with the disk system contain two versions of DOS68. Version 3 is the default version and uses the same disk file management system and monitor as earlier versions but is supplied with more transient commands. Version 4 is the new version of DOS68 which handles random files and requires more memory space. To use version 4 you should use the LINK command to link to DOS68.4X, where X is the latest version number listed in the directory.

This manual describes DOS68 version 4; however, with the exception of references to random access files, the operating instructions are nearly identical.

DOS68 version 4 is assembled to use the $6000 and $7000 memory blocks. A version using $A000 and $B000, and a version using $C000 and $D000 are available as options from Smoke Signal Broadcasting.

PROBLEMS IN BOOTING DOS68

If you cannot get DOS68 to boot in, you may have hardware problems. Some commonly encountered problems are:

1. The head loads and unloads without DOS68 printing its banner. This may be due to no memory at $7000. 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 dislodged 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 SA400 drive disk system systems.

3. DOS68 works but every character is printed on the terminal twice. See the section "Using DOS68 with monitors other than MIKBUG".

PART 1: OPERATING DOS68 VERSION 4

INTRODUCTION

DOS68 is a disk oriented monitor for use with Smoke Signal Broadcasting's disk systems.

Most DOS68 commands are implemented as transient programs. This means that the routine to process the command is not resident in memory, but rather is loaded into memory when it is needed to be executed. This facility allows two things: 1) the resident monitor is smaller because all the routines are not present in memory, and 2) the user is easily able to implement new monitor commands without modification to the monitor.

In addition, in the calling of transient monitor commands, the users of multi-disk systems have the ability to specify that the transient command is to be found on a specific disk drive. This feature is particularly useful when debugging new monitors with transient commands with names which conflict with the old system's names. The disk number is specified by preceeding the command name with the disk 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.

DOS68 is supplied with the following commands:

	COMMAND NAME	FUNCTION
	LIST	List the disk file directory
	SAVE	Save memory into a file
*	GET	Load a binary object file into memory
	GETH	Load a hex formatted object file
*	RUN	load a file into memory and begin execution
	DELETE	Remove a file from a disk
	RENAME	Change the name of a file
	APPEND	Merges two files together to form one file
	PRINT	Print the contents of a file
	COPY	Allows files to be copied from disk to disk
	SDC	Single disk drive copy
	LINK	Set up information to boot the monitor
	INSTAL	Convert object file to a command file
	REMOVE	Convert a command file to an object file
	FIND	Type load address information
	VIEW	Type contents of an editor text file
*	CLOSE	Close all open files
*	EXIT	Exit to other resident monitor
	ASYS	Assign system disk
	FORMAT	Format a blank disk

(* indicates a memory resident command)

In the command descriptions to follow, the following conventions are used:

The disk drive unit numbers are Ø, 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 files 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 unit Ø
FNAME	A file called "FNAME" assumed to be on unit Ø
Ø:FILE.ONE	Note that Ø:FILE.ONE is different file from 1:FILE.ONE

COMMAND DESCRIPTIONS

DOS68 indicates its readiness to accept a new command by typing an ampersand, "&", on the terminal. At this time a new command line may be entered.

Commands given to DOS68 are entered on a line input basis. This means that the entire line is typed in before DOS68 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 DOS68 starts any processing. The previously entered character can be deleted by using CONTROL O or SHIFT O (back arrow). DOS68 will echo all deleted characters enclosing them in backslashes, "\". CONTROL U or CONTROL X can used to delete the entire line; DOS68 will echo ^U OR ^X and do a carriage return line feed and be ready for a new command (a new prompt charcter is not issued).

NOTE: DOS68 assumes that the character input routine for the terminal does not automatically echo the character. Thus, while the user is typing a command, the input routine within DOS68 supresses the usual automatic echo used by the MIKBUG (TM Motorola) ROM. Using DOS68 with ROM monitors other than MIKBUG may result in having characters double echoed to the terminal. For this problem see the section "USING DOS68 WITH ROM MONITORS OTHER THAN MIKBUG".

RESIDENT COMMAND DESCRIPTIONS

This section describes those DOS68 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 "DOS68 MEMORY MAP" table. The address of the ROM monitor entry point can be changed by changing this table entry.

CLOSE — &CLOSE

The close command attempts to close any currently open disk files. (NOTE: DOS68 automatically performs the CLOSE function upon entry to DOS68 and after execution of each monitor command.)

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 disk Ø
&GET,1:XYZ        Load the file "XYZ" into memory from disk 1
&GET,PROG.REL,1ØØØ    Load "PROG.REL" $1ØØØ locations above
                      the default load address
```

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,1ØØØ   Load with a $1ØØØ offset and
                       run "PROG.REL"
```

TRANSIENT COMMAND DESCRIPTIONS

This section describes the transient commands supplied with DOS68. DOS68 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.

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.

Examples:

```
    &LIST                 List directory of disk 0
    &LIST,0               List directory of disk 0
    &LIST,2               List directory of disk 2
```

The format of the directory listing is as follows:

```
    &LIST,2
    FILE NAME    SB    EB    NB    FS
    ABC    .1    8042  8043  0002  0200
    FILE   .TWO  8044  8049  0006  0200

    AVAILABLE SECTOR COUNT: 026C
```

The file names are under the heading "FILE NAME" followed by information about the file. "SB" is the starting disk address of the file, "EB" is the ending disk address for the file, "NB" is the number of blocks being used by the file, and "FS" is the file status. The first two hex digits are the file type (e.g. 02 for a sequential file and 04 for a random file). The second two hex digits are the file status and are 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).

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 available to be used is given in hexadecimal on the last line of the listing.

The LIST command may also be used to list the directory of the monitor transient commands resident on disk. The command file names are listed by following the unit number by a '$' parameter.

For example:

```
    &LIST,0,$
    FILE NAME    SB    EB    NB    FS
    SAVE   .$    8042  8044  0003  0200
    DELETE.$     8045  8046  0002  0200
    RENAME.$     8047  8048  0002  0200
```

The above files with the extension of '$' are examples of some transient commands available to the monitor.

SAVE &SAVE,<FILE>,<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. <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. The SAVE command may also be used to save transient monitor commands by using the '$' parameter.

For example:

```
&SAVE,1:MEMORY.LOW,0,1000     $0 thru $1000 is saved into a file
                              called "MEMORY.LOW" on disk 1
&SAVE,2:CMD,7080,7180,7080,$      A transient command file is
                                  saved on disk 2.
```

NOTE: Command files must not have an extension on the file name when saved.

GETH &GETH,<FILE SPEC>[,<OFFSET>]

The GETH command is used for loading the object files created by using "CORES" (the coresident assembler/editor package). See the description of the "GET" command.

DELETE &DELETE,<FILE SPEC>[,$]

The DELETE command removes the specified file from the directory of the unit specified. The sectors used by the file now become available for reuse. No disk repacking is never required when files are deleted. The delete command may also be used to delete monitor transient command files from the disk by following the name of the command by a '$' parameter.

Examples:

```
&DELETE,AB.123        This deletes file "AB.123" from disk 0
&DELETE,2:FILE.ONE    This deletes a file called "FILE.ONE" from
                      the directory of disk unit 2

&DELETE,1:RENAME,$    This deletes from disk 1 the transient command
                      file for the rename command.
```

RENAME &RENAME,<THIS>,<THAT>[,$]

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

For example:

```
&RENAME,FILE.ONE,FILE.1     On disk 0 the file "FILE.ONE" is
                            renamed to "FILE.1"
```

&RENAME,2:THIS,THAT	On disk 2, the file "THIS" is renamed to "THAT"
&RENAME,LIST,LI,$	On disk Ø, renames the command file "LIST.$" to "LI.$"

APPEND &APPEND,<THIS FILE>,<THAT FILE>

The APPEND command allows two files to be merged 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.

FOR EXAMPLE:
&APPEND,FILE.ONE,2:FILE.TWO

This causes the file "FILE.ONE" on disk two to be appended to file "FILE.TWO" on disk two.

PRINT &PRINT,<FILESPEC>[,<LINE COUNT>]

The PRINT command prints the contents of the specified file on the terminal. If the optional line count is not specified, the entire file is listed without pausing. If the number of lines is supplied, the file is listed until that number of line feed characters occurs at which time the listing pauses until a key on the terminal is struck. If the key is a carriage return character the listing is terminated, otherwise, the listing continues until another set of lines has been printed. The PRINT command will list any non-command file, including binary files which will appear as a string of odd characters and control characters.

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.

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?????.HEX	means all files with an extension of "HEX" 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:*.HEX 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 "HEX" 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 feature which is dependent upon the fact that COPY assumes that an ACIA is being used for the terminal I/O. This feature is a Control-C abort. After each file is copied, COPY checks to see if a character has been entered from the terminal. If a character has been entered, the character is checked for being control-C. If it is control-C, an abort message is printed, and the copy is terminated.

Users still forced to live with MIKBUG's PIA for serial I/O will have to patch out this feature by setting memory locations $107 and $108 to $00,$00 rather than the address of the ACIA. COPY is saved on disk with a starting address of $100, and ending address which is the contents of memory locations $109 and $10A, and a transfer address of $104.

<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 Ø 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 1 through 7 for 4 through 28K.

"/P" Partial. The partial option forces COPY to copy as much possible of a file which has a read error preventing 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 specifying switches:

&COPY Ø: 1:/L=7
Use all 28K for buffer to copy all of disk Ø to disk 1

&COPY,Ø:,1:/L=7/P/NV/C
Use all 28K for buffer to copy all of disk Ø 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 buffer to copy all command files from disk Ø to 1 with confirmation

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

SDC — &SDC,<FILE SPEC>[,$]

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 copy files which will fit into memory.

The SDC command assumes that the user has 16K of RAM origined at Ø, and that the disk drive in the system is unit zero.

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 disk in drive zero to be set up such that the file "MON" will be run when this disk is booted in. &LINK,2:NEWMON similiarly 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.

INSTAL &INSTAL,<OBJ FILE>[,<CMD NAME>]

INSTAL is a command which renames an executable file (that is, a file with a transfer address) to a "$" file.

General command format:

&INSTAL,<EXECUTABLE FILE SPEC>[,<COMMAND NAME>]

The name of the command remains the same as the 6 character file name of the original file unless the optional 6 character command name is specified.

The advantage of this command is that a command file may now be SAVEd in separate files (non-$ files), appended together to form the final file, and then INSTALled as a "$" file.

Examples:

```
1)  &SAVE 1:PART.1,100,200       save first memory range
    &SAVE 1:PART.2,500,600,100   save second memory range
    &APPEND 1:PART.2,1:PART.1    append the parts together
    &INSTAL 1:PART.1,MYCMD       install the file as a command
                                 called "MYCMD"

    Note: the above is equivalent to:
          &SAVE MYCMD,100,600,100,$
          except that less disk space is used by the
          "INSTAL" method since the range 201 through 4FF
          is not saved on the disk when not actually needed.

2)  &INSTAL,2:PROG.BIN           on disk 2, rename PROG.BIN
                                 to PROG.$
```

NOTE: INSTAL and REMOVE perform inverse functions of each other.

REMOVE &REMOVE,<COMMAND>[,<OBJ FILE>]

REMOVE is a command which renames a transient command file (a "$" file) to a non-$ file. The effect of a REMOVE command is to change the "$" extension to a blank extension unless the optional <NEW FILE NAME> is specified.

General command format:

```
&REMOVE,<TRANSIENT FILE NAME>[,<NEW FILE NAME>]
```

Examples:

```
&REMOVE LIST            is the equivalent to the illegal
                        command  &RENAME LIST.$,LIST

&REMOVE,ASMB,OLDASM.BIN
                        is the equivalent to the illegal
                        command  &RENAME,ASMB.$,OLDASM.BIN
```

NOTE: REMOVE and INSTALL perform the inverse function of each other.

FIND — &FIND,<FILE NAME>[,$]

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

General command format:

```
&FIND,<OBJECT FILE SPEC>[,$]
```

Where <OBJECT FILE SPEC> specifies an object file in the binary record format used by the "SAVE" command and the SA-1 assembler (NOT the format used by CORES). If the object file is a command file, indicate so by following the command name by a ",$".

Example: Find out where FIND loads.

```
&FIND,FIND,$
 LOADS FROM Ø1ØØ THRU Ø267
TRANSFER ADD = Ø1Ø6
```

WARNING: FIND.$ is slightly too large to load in the transient command area and so was origined to $1ØØ.

VIEW — &VIEW,<FILE NAME>[,<LINE COUNT>]

VIEW is a command for typing the contents of an editor 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 responce to a pause, a carriage return will terminate the listing; all other characters will cause the listing to continue.

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 disk in drive zero to be set up such that the file "MON" will be run when this disk is booted in. &LINK,2:NEWMON similiarly 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.

INSTAL &INSTAL,<OBJ FILE>[,<CMD NAME>]

INSTAL is a command which renames an executable file (that is, a file with a transfer address) to a "$" file.

General command format:

&INSTAL,<EXECUTABLE FILE SPEC>[,<COMMAND NAME>]

The name of the command remains the same as the 6 character file name of the original file unless the optional 6 character command name is specified.

The advantage of this command is that a command file may now be SAVEd in separate files (non-$ files), appended together to form the final file, and then INSTALled as a "$" file.

Examples:

```
1)  &SAVE 1:PART.1,100,200       save first memory range
    &SAVE 1:PART.2,500,600,100   save second memory range
    &APPEND 1:PART.2,1:PART.1    append the parts together
    &INSTAL 1:PART.1,MYCMD       install the file as a command
                                 called "MYCMD"

    Note: the above is equivalent to:
          &SAVE MYCMD,100,600,100,$
          except that less disk space is used by the
          "INSTAL" method since the range 201 through 4FF
          is not saved on the disk when not actually needed.

2)  &INSTAL,2:PROG.BIN           on disk 2, rename PROG.BIN
                                 to PROG.$
```

NOTE: INSTAL and REMOVE perform inverse functions of each other.

REMOVE &REMOVE,<COMMAND>[,<OBJ FILE>]

REMOVE is a command which renames a transient command file (a "$" file) to a non-$ file. The effect of a REMOVE command is to change the "$" extension to a blank extension unless the optional <NEW FILE NAME> is specified.

General command format:

```
&REMOVE,<TRANSIENT FILE NAME>[,<NEW FILE NAME>]
```

Examples:

```
&REMOVE LIST            is the equivalent to the illegal
                        command  &RENAME LIST.$,LIST

&REMOVE,ASMB,OLDASM.BIN
                        is the equivalent to the illegal
                        command  &RENAME,ASMB.$,OLDASM.BIN
```

NOTE: REMOVE and INSTALL perform the inverse function of each other.

FIND — &FIND,<FILE NAME>[,$]

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

General command format:

```
&FIND,<OBJECT FILE SPEC>[,$]
```

Where <OBJECT FILE SPEC> specifies an object file in the binary record format used by the "SAVE" command and the SA-1 assembler (NOT the format used by CORES). If the object file is a command file, indicate so by following the command name by a ",$".

Example: Find out where FIND loads.

```
&FIND,FIND,$
 LOADS FROM Ø1ØØ THRU Ø267
TRANSFER ADD = Ø1Ø6
```

WARNING: FIND.$ is slightly too large to load in the transient command area and so was origined to $1ØØ.

VIEW — &VIEW,<FILE NAME>[,<LINE COUNT>]

VIEW is a command for typing the contents of an editor 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 responce to a pause, a carriage return will terminate the listing; all other characters will cause the listing to continue.

ASYS &ASYS,[<UNIT NUMBER>]

The ASYS command allows users of DOS68 Version 4 to assign the system drive (the drive on which DOS68 expects to find system overlay files).

To determine which drive the system is currently assigned to, type:

```
&ASYS
```

The ASYS command will respond with:

```
SYSTEM ASSIGNED TO DRIVE n:
```

where "n" corresponds to the disk drive number. To assign the system to another unit, type:

```
&ASYS,n
```

where, again, "n" corresponds to the drive number to which the system is to be assigned. ASYS will respond with the same message shown to indicate the new drive assignment.

DOS68 COMMAND ERROR MESSAGES

DOS68 prints only its prompt character unless an error condition occurs. The following are error messages which can be generated by DOS68:

ILL CMD
: DOS68 does not understand the format of the command entered. Try again.

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

ILL UNIT #
: An invalid unit number has been entered. The only valid unit numbers are Ø, 1, 2 and 3.

ILL VALUE
: 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 file to be RUN. The file has been loaded but DOS68 does not know where to begin execution.

RUN DENIED
: The requested transient command file could not be found. Therefore, the RUN is denied since no file was loaded.

CS ERR: XXXX

A checksum error has occured during the loading of the 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). The file should be deleted and replaced with a backup copy.

CLOSE ERR: XXXX

DOS68 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, DFM68 cannot close the file. This is usually caused by a program corrupting the contents of the FCB. The only cure for this error is too cold start DOS68 (it may be advisable to reboot DOS68 since part of DOS68 may also have been corrupted by the offending program).

BOOTING DOS68

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 monitor).

The ROM supplied on the interface board contains a booting routine capable of reading in sector zero of track zero on disk drive zero and transferring to the routine read in. The content of this sector is initialized by the disk formatting program to contain a program capable of loading the monitor.

Thus to load DOS68, the user must use MIKBUG (TM Motorola) or some other method of transferring to location $8020. This will cause DOS68 to be brought in off the disk and executed. Booting in this manner is referred to as "COLD STARTING" because all monitor and disk file manager temporaries are initialized. It is possible to "WARM START" the monitor, that is, to reload the monitor without initializing everything by instead transferring to the ROM at location $8023.

NOTE 1: For new disks to know what file to boot in when using the ROM resident booting routine, the monitor must have been "LINKED" (see the "LINK" command). "LINKING" the disk is a process whereby the booting routine is told what file to load and run when the disk is booted. Also, note that the COPY command does NOT transfer linking information.

NOTE 2: The ROM supplied with the controller board utilizes memory locations $7000 through $707F and the boot program additionally uses locations $7F80 through $7FFF when boot loading. This is true even if the program being booted does not reside in the $7000 range.

WRITE PROTECT

The BFD-68 is provided with a write protect feature. Each 5" diskette has a small rectangular cutout on one edge. Covering the cutout with a piece of tape will protect the diskette from an accidental write. An attempt to execute any command that would write to the diskette will return an error message (error EDWP for version 4 DOS and error EDW for version 3 DOS).

LOADS FROM 0100 TO 0112
0118 TO 0A92

DISK FORMATTING TRANSFER ADD 0A00 &FORMAT[,<FORMAT PARAMETERS>]

FORMAT is a command for the initialization of blank diskettes for use with DOS68. FORMAT must be used to prepare every disk to be used by DOS68. FORMAT is invoked by either of the following two monitor commands:

```
&FORMAT,<Format Parameters>
                              or
&FORMAT(cr)
FMT:
```

The first command will cause FORMAT to perform the functions specified by the parameters - after which control will return to DOS68. The second command format will cause FORMAT to prompt the operator for new commands. Entering a Carriage Return immediately after any prompt will return control to DOS68. (This second command format saves having to reload FORMAT if it is to be used to format more than one disk).

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 Ø. If a unit number is supplied, it must be one of the numbers Ø, 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

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 F and Q options may be turned off by preceeding the command symbol by a minus sign. (-F means do not use Fast Mode).

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.

Upon completion, the formatter types the message:

FORMATTING COMPLETE. XXXX YYYY ZZZZ

Where:

XXXX	Is the first sector available for use,
YYYY	Is the last sector available for use, and
ZZZZ	Is the count, in hex, of available sectors

Only a few seconds are required to actually format the disk. The remainder of the time is spent checking each sector of the disk to verify that each has no dropouts.

Sectors which fail the verification procedure are eliminated from the disk file structure at this time to prevent their subsequent usage.

FORMAT asks two questions of the operator to prevent accidentally formatting a wrong disk:

ARE YOU SURE YOU WANT TO FORMAT ON DRIVE n?.

Where "n" should correspond to the unit number the operator desires to format on. Respond with "Y" and a carriage return if this is the desired drive number. Any other entry will abort the formatter. If "Y" is entered, FORMAT will then ask:

HONEST?

Enter "H" followed by a carriage return if this, honestly, is the intended 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 DOS68 operating system on every disk used by DOS68. it is only necessary to have the system present on the disk from which the system is to be boot loaded.

STEP 1: FORMAT A DISK

The first step in building a new disk is to format a new disk. DOS68 is supplied with a command called "FORMAT". FORMAT is a program which initializes blank soft sectored diskettes (see the section on formatting new disks).

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 using DOS68 Version 4 will have to use Version 3 to copy "DFM68Ø.??1" over to the new disk before being able to use version 4 to copy other sequential files to the new disk. Version 3 must be used because "DFM68Ø.??1" contains version 4's sequential file write close program which version 4 cannot use until the file has been closed on the new disk. Similarly, to copy random access files, DFM68Ø.??3 must first be copied to the new disk. (?? refers to the DFM revision number).

MULTIPLE DRIVE SYSTEMS

On multiple drive systems the transient monitor command "COPY" is used to copy files from disk to disk.

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

```
&1:COPY 1: Ø:
```

NOTE: PREPARING SYSTEM DISKS

If the disk being prepared is to be used to boot from a cold or warm start, the disk must contain a copy of DOS68 and the disk must be "LINKED" by executing the following monitor command:

```
&LINK,DOS68.4X    (Assuming the new disk is in drive zero)
                  ("X" is the latest revision number)
```

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

DOS68 VERSION 4 MEMORY MAP

$7000 - $707F Used by ROM to load system boot from disk later reused by monitor for stack area and line input buffer

$7080 - $727F Transient Command Area (TCA) (It is recommended that future transient commands be located $6080 to allow for future monitor expansion.)

$7280 - $777F Monitor program area

$7780 - $7FFF DFM program area
$6800 - $6FFF more DFM and future monitor space (may expand downward if insufficient room).

$7F80 - $7FFF I/O buffer for ROM boot routine

$0000 - $5FFF Is assumed to be user area when not used by the following commands.

$0100 - $3FFF Is used by the "COPY" command

$0100 - $02FF Is used by "SAVE.BLD" program

$0100 - $03FF Is used by "FIND" command

ROM ENTRY POINTS

$8020 - Monitor cold start boot entry
$8023 - Monitor warm start boot entry

$A070 - $A07F ROM temporary table
$A070 - $A079 Stack for ROM when booting.

MONITOR ENTRY POINTS

$7280 - Cold start entry

$7283 - Warm start entry

$7286 - Jump to keyboard output (OUTEEE)

$7289 - Jump to keyboard input (INEEE)

$728C - Jump to user's monitor

$728F - Resident command table extension

DOS68 AT HIGHER MEMORY ADDRESSES

A disk with DOS68 Version 4 located at either $C000 through $DFFF or $A000 through $BFFF is available from Smoke Signal Broadcasting. This will allow the user to have the entire lower 32K available for user programs, except during boot operations. The boot routine will use $7000 through $707F and $7F80 through $7FFF while it loads the monitor into the higher addressing region. After loading the monitor, the entire $7000 block is available to the user.

No change to the CPU card is needed to locate memory in the higher addresses; however, the SWTPC 4K or 8K memory cards will not locate in that area without modification. The user, desiring to use DOS68 at the higher addresses is responsible for modifying his memory card to properly operate at that location. If the memory is to be located at $A000 the on board RAM will need to be disabled.

The Smoke Signal Broadcasting M-16A 16K memory board may be switch selected to occupy $A000 through $DFFF and, thus, can provide the memory required to operate DOS68 at the higher addresses. When the M-16A is used at that location, a simple modification to the SWTPC CPU card is required. Modification instructions are given in the M-16A manual. No modifications are required to the SWTPC A2 card; however, the on board RAM at $A000 must be switched off.

When using the higher addressed versions of DOS68, all programs which access the monitor or DFM in the $6000 through $7FFF area should be changed to access corresponding locations in the higher addresses.

HIGH DOS USERS

For those of you converting from the standard DOS to the high address versions of DOS, two files have been supplied to make your job easier. The two files are patches to the editor (SE-1) and assembler (SA-1) to work with the high DOS.

On the $A000 - $BFFF disk:

```
REMOVE,EDIT
APPEND,EAP.PAT,EDIT
INSTAL,EDIT
REMOVE,ASMB
APPEND,AAP.PAT,ASMB
INSTAL,ASMB
```

will patch the editor and assembler. (This example assumes all files are on drive 0). On the $C000 - $DFFF disk use the same proceedure except substitute:

```
ECP.PAT in place of EAP.PAT and
ACP.PAT in place of AAP.PAT.
```

PART 2: DOS68 SYSTEM PROGRAMMER'S GUIDE

INTRODUCTION

The Smoke Signal Broadcasting disk operating system, DOS68, consists of two distinct programs:

(1) The monitor portion (TMON), and
(2) The Disk File Management portion (DFM68).

DFM68 is strictly a disk file handling system and can be used independently of the monitor portion of DOS68 when the user wishes to manipulate disk files.

The monitor portion of DOS68 handles all other functions of DOS68 not handled by DFM68.

The following sections provide the necessary interfacing information for the user to be able to make use of the functions available through DOS68.

MONITOR SYSTEM

This section describes the user interface to the DOS68 monitor.

MONITOR JUMP TABLE

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
$7280	ZCOLDS	Monitor cold start
$7283	ZWARMS	Monitor warm start
$7286	OUTEEE	Character output routine
$7289	INEEE	Character input routine
$728C	ZMON	JMP to ROM monitor
$7291	ZFLSPC	Get a file specification
$7294	ZGCHAR	Get current character from the line buffer
$7297	ZGNCHR	Get the next character from the line buffer
$729A	ZANCHK	Check for alphanumeric
$729D	ZDIE	Print command string, error message, and exit
$72A0	ZGETHN	Get a hex value from the line buffer
$72A3	ZADDX	Add the B register to the index
$72A6	ZOUTST	Print a string
$72A9	ZTYPDE	Type the disk error message
$72AC	ZOUTHX	Print a byte in hex
$72AF	ZOUTHA	Print an address in hex
$72B2	(Unused)	
$72B5	ZLINEI	Input edited line from the terminal

ZCOLDS - Monitor cold start

This entry to DOS68 resets the processor's stack and all internal status of both the monitor and DFM. The banner:

DOS68 VX.YR

is printed on the terminal where VX.YR represents the version number. DOS68 then proceeds to do a warm start.

ZWARMS - Monitor warm start

This entry to DOS68 resets the processor's stack, sets $A048 to the address of 'ZWARMS' for subsequent restarting, closes any files that may have been left open, and then prompts the operator for a new command by typing an ampersand, '&'.

OUTEEE - Character output to the control terminal

This JMP is used for all output to the user's terminal. This JMP is set to use "OUTEEE" at $E1D1 within the resident ROM. It is assumed that the output routine preserves the B register and the X register but not necessarily the A register (the data to be output).

INEEE - Character input from the control terminal

This JMP is used for all input from the user's control terminal. This JMP is set to use "INEEE", $E1AC, within the resident ROM. It is assumed that the B and X registers are preserved and that the character input is returned in the A register.

DOS68 assumes that INEEE does not automatically echo input back to the terminal. See description of 'ZLINEI'.

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 $E0E3 to cause entry into the resident ROM.

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+Ø through FCB+$25 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 and file name and the A register will return with the delimiting character of the file name.

No 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.

When control is given to a transient program, the buffer pointer is pointing to the delimiter of the command name.

NOTE: When using the line input routine, this routine must not be called until either ZGNCHR, ZFLPSC, or ZGETHN has been used because 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.

ZANCHK - Alphanumeric check

The character in the A register is checked for being Ø-9 or A-Z. If the character is not one of these characters, the carry bit will be set on return. The A, B, and X registers are not affected.

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 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-alphanumeric 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 value in the B register is added into the value in the X register and the result is returned in the X register. The A register is not affected, and the B register will contain the low order sum.

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 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 "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; the B and X registers are unchanged.

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.

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
CONTROL U AND CONTROL X: Restart line input
CONTROL O AND SHIFT O: 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 the carriage return ending the input.

NOTE: The automatic echo feature of the MIKBUG serial I/O PIA is inhibited during the line input routine and re-enabled upon exit. This is done upon entry to ZLINEI by:

```
LDA A       #$3C
STA A       $8007
```

Auto-echo is re-enabled at the ZLINEI exit approximately 35 bytes later:

```
LDA A       #$34
STA A       $8007
```

If a ROM monitor other than MIKBUG is to be used, refer to the section "USING MONITORS OTHER THAN MIKBUG".

USER COMMAND TABLE

When a command line is processed by DOS68, 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, "RUN DENIED" is output to the terminal.

At memory locations $728F and $7290 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     $728F       Tell DOS68 about this table
        FDB     START
```

In the above example, DOS68 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 "SAVE.BLD" 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, 'SAVE.BLD' is a resident version of the transient SAVE command.

SAVE.BLD loads at memory location $0400 by the GET command and is approximately 512 bytes in length. When loaded, the monitor will recognize the command "SAVE" as a resident command which functions identically as the transient save command. The new transient program can be loaded into the TCA and saved using the resident save command. To release the resident save command, it is necessary to zero the two byte long resident command table extension pointer located at locations $728F and $7290 (or to return it to the address of the beginning of the user command table described in the previous section).

USING DOS68 WITH ROM MONITORS OTHER THAN MIKBUG

DOS68 can be used with ROM monitors other than MIKBUG with little difficulty. There are only four pieces of information DOS68 needs to know about the resident monitor:

These are:
1) Where to enter the ROM monitor on the "EXIT" command,
2) Where the character output routine is located,
3) Where the character input routine is located, and
4) If necessary, how to prevent the character input routine from automatically echoing input characters back to the output.

The first three pieces of information are changed by changing the monitor jump table addresses described previously. Suppressing the input echo requires a little more work.

DOS68 is configured to run with MIKBUG and as such automatic echo is suppressed by storing $3C into location $8007 and is restored by storing $34 into location $8007. This code is found in ZLINEI by following the jump table entry to the ZLINEI routine and looking for the code sequence 86 3C B7 80 07 and changing this as needed to turn off the echo. The echo is restored by the instruction sequence 86 34 B7 80 07.

If using SMARTBUG or SWTBUG, DOS68 will double echo characters entered from the terminal. The following proceedure will correct this situation:

1) Boot DOS68 into memory by jumping to $8020.

2) Type the following command:

   APPEND,SBF,DOS68.XX where XX is the version number of DOS68 to be changed.

3) Hit RESET on the computer.

4) Again, boot DOS68 by jumping to $8020.

Input characters should now echo correctly. The above proceedure is correct for SMARTBUG users. If using SWTBUG, substitute "SWTF" for "SBF" in step 2.

NOTES ON MODIFYING DOS68 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 DOS68 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 DOS68.

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, DOS68 will then transfer to the starting address.

DFM68 SYSTEM

This section is directed toward how to use DFM68 and how to interpret the disk structure used.

INTRODUCTION

DFM68 is a disk file management program written for Motorola 6800 based microcomputers using Smoke Signal Broadcasting's disk controller.

DFM68 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) Allocating and de-allocating 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 non-resident function handlers.
   Infrequently used function processors are kept in three system files on the system disk. These three 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). The 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 Ø 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 32Ø ($14Ø) 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.

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

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

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

NAME	LOCATION	USAGE
XFC	FCB+Ø	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+2Ø	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)
	+3Ø	" " " " 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+Ø 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 Ø 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
Ø	Track # of next directory block (Ø 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 Ø 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 $FF
5 - 14	Don't cares ($FF)
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
2Ø	Low order count of available sectors

INTERFACING TO DFM

There are three entry points into DFM68; they are:

1) The DFM68 initialization entry point
2) The DFM68 closing entry point
3) The DFM68 I/O service request entry point

These three entry points correspond to three "JMP" instructions located in the first nine bytes of DFM68 (see the memory map for specific addresses).

INITIALIZATION ENTRY POINT (ODFM)

DFM68 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 DFM68. Initializing DFM also logs drive Ø as as the system drive (see the QLOGD function description).

DFM CLOSING ENTRY POINT (CDFM)

When no further use is to be made of DFM68, 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 the 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.

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 are passed to DFM in the FCB. The address of the FCB is loaded in the X register when DFM is called. All registers 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 appendices.

QFREE: REPORT FREE SPACE

DFM68 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.

QSO4W: OPEN A SEQUENTIAL FILE FOR WRITE

DFM68 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 are will be used in future versions of DOS68).

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 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 openned 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 wiil 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 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 of error status codes).

QSRC: CLOSE A SEQUENTIAL READ FILE

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

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 te 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 (ØØ). 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 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).

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+Ø and FCB+XFN+1. If the ordinal is Ø 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. This function is not available in DOS revisions below 4.Ø.

QLOGD: LOGGING A SYSTEM DRIVE

DFM must be told on which disk to expect to find its three overlay files if the drive is to be other than drive Ø. On calling ODFM, DFM logs drive Ø 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. This function is not available in DOS revisions below 4.Ø.

QLOGE: EXAMINE SYSTEM DISK LOG

The user can determine which drive is the logged in as the system drive by the QLOGE function. The QLODE function will return the system drive number in in FCB+XUN. This function is not available in DOS revisions below 4.Ø.

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+3Ø (XCT) and FCB+31 (XCS) respectively; the track number must be a value between Ø and 34 and the sector number a value between Ø and 17.

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 counter part of the single sector read command (see above descripton).

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.

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 user must supply the following information:

1) The unit number on which to create the file is place 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 appendix B for values of 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 files contents to the value contained in FCB+XRID. If FCB+XRIF is zero then the contents of the file are indeterminant.

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 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 in 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 cuurent 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

DFM68 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, XRBHW is set to $FF, $FF, $FF to indicate that nothing has been written to the file. As with all other FIB contents, XRBHW is available in the FCB while the file is open.

QERF: RANDOM FILE EXPANSION

DFM68 Version 4 contains a convenient means for expanding a random access file once it has been created. The function code is named QERF and its value is $1C. 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 last data sector of the original file remain initialized as determined during the file creation, not as during expansion.

From this information, DFM68 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 DOS68 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

1) 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 Ø, 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+Ø (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+1Ø        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     $7780       Open DFM entry point
CDFM    EQU     $7783       CLOSE DFM entry
DFM     EQU     $7786       DFM service request entry

        LDX     #FCB        Load the address of the FCB
        LDA A   #QSO4R      Load the value of the "open for read"
                            function code
                            (see the appendices for the function
                            code values)
        STA A   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 A   #QSREAD     Change to function code to read from
                            the file
        STA A   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 A   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 A   #QSRC       Set the function code to "Read Close"
        STA A   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 perparing 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.

```
DMF     EQU     $7786       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 A   #FTRB       set up the file type (byte or record)
        STA A   XFT,X
        LDA A   #1          set flag to force file contents
        STA A   XRIF,X      to be initialized
        CLR     XRID,X      initialize all bytes to 0
        LDA A   #QCRF       set up command for DFM
        STA A   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 CONTAINDS THE FILE
* NAME AND UNIT NUMBER
        LDX     #FCB
        LDA A   #QORF       set up command for DFM
        STA A   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
```

```
        LDA A   #4000/256   mid order byte address
        LDA B   #4000       low order byte address
        STA A   XRBA+1,X
        STA B   XRBA+2,X
        LDA A   #QPRF       set up command to position the file
        STA A   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 A   #QWRF       perpare write function code
        STA A   XFC,X
        LDA A   #'A
        JSR     DFM         write a byte
        BNE     PER
        LDA A   #'B
        JSR     DFM
        BNE     PER
        LDA A   #'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.

PART 3: DISK SYSTEM HARDWARE

PREFACE

This section describes the characteristics of the SSB floppy disk interface. This interface allows users of the SWTPC 6800 microcomputer system to easily interface up to four SA400/SA450 5" disk drives or four SA800/SA850 8" disk drives.

BOOT AND I/O ROUTINES

To facilitate disk I/O a ROM has been provided on the disk interface board. 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.

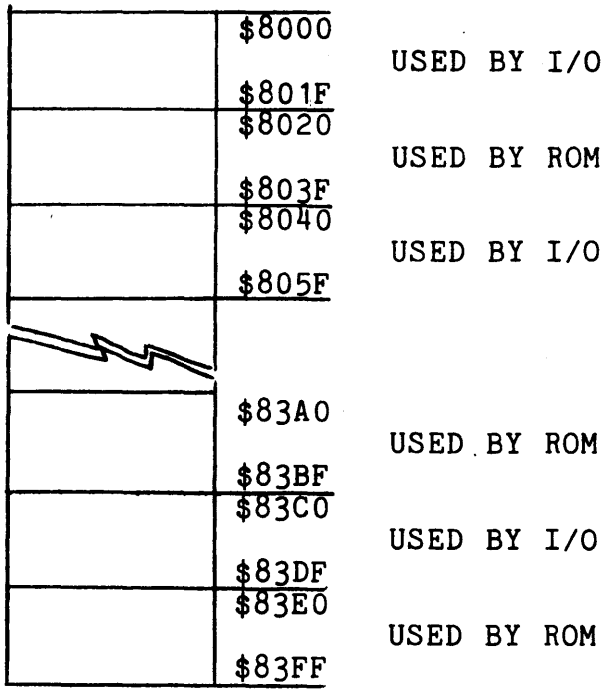
FD1771B-01 CONTROL

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.

ROM

The ROM located at U5 provides 512x8 bits of information for the user. The ROM addressing is decoded to use all unused addresses in the I/O page between $8020 and $83FF. The 9324 decoder U7 provides high order address decoding. Use of unused areas of the I/O page is achieved by requiring address bit 5 to be true to enable the ROM. Address bits 0-4 of the ROM are driven by address lines 0-4. Address bits 5-8 are driven by address lines 6-9. The figure below illustrates the interleaving of I/O and ROM addresses.

ROM MEMORY MAP



The above figure shows I/O memory illustrated as pages of 32 words. When bit 5 is a Ø, that 32 word page is used as I/O locations. When bit 5 is a 1 that page is decoded by the ROM as one of its addresses and the particular byte of information required is placed on the system data buss.

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 AØ and A1 at U18 pins 36 and 35.

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-Ø1 IC are tied to the B-PORT of the PIA at U18(1Ø-17). Data to and from the FD1771B-Ø1 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-Ø1 as is required with the control registers of the PIA. Data exchange between the PIA and FD1771B-Ø1 are controlled by the following control lines:

WRITE ENABLE

CB2 of the PIA is used to strobe information from the 6821 into the FD1771B-Ø1. CRB bits 3-5 should be programmed to '1Ø1'. 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 Ø respectively.

REGISTER SELECT

A-PORT bits Ø and 1 drive the register select lines of the floppy disk controller chip. PAØ drives address line AØ 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-Ø1. Mode two is used when reading data from the disk. In mode one PA6 (U18-8) is programmed to 'Ø'. PA6 low forces U25-6 to the low state thereby enabling information from 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 =11Ø). 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 SA4ØØ minifloppy is approximately 75 milliseconds. The 96Ø2 oneshot (U9) provides the delay signal required for proper operation of the FD1771B-Ø1. 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 perriod of approximately 30 seconds. After the last head load, U16-2 goes high which reverse biases D1, allowing U8 to time ouut. 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 gatte (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 progrrammed 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.

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 reversed 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 BFD-68 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.

TYPE

1 - BINARY SEQUENTIAL
DEFAULT TO THIS

2 - COMPRESSED SEQ.
ARE SPECIFIED BY FOLLOWING FILE NAME B "/C"
WHEN CREATING THE FILE (I.E.)

```
CREATE #2, "COMPRS.ME/C"
```

OR

```
INPUT "ENTER NAME FOR COMPRESS FILE", F$
CREATE #2, F$ CAT "/C"
```

TO OPEN (AUTO) WILL EXPAND BLANKS

```
OPEN #3, "COMPRS.ME"
```

3 BYTE ADDRESSABLE RANDOM (I.E.)

```
CREATE #1, "BYTE.RND/R=10000"
```

WILL CREATE RANDOM ACCESS FILE 10,000 INDIVIDUALLY ADDRESSABLE BYTES W/ RESTORE STATEMENT (I.E.)

```
INPUT "ENTERED ADDRESS TO POSTION TO:" ADDR
RESTORE #1, ADDR
```

THEN

```
DIM FILECONTENTS$(20)
READ #1, FILECONTENTS$
```

{ WILL READ 20 BYTES FROM FILE STARTING @ SPECIFIED BY THE CONTENTS OF ADDR. FILE WILL BE LEFT POSITIONED TO NEXT BYTE AFTER THE READ

TO CHANGE 20 BYTE

```
FILECONTENTS$ = "NEW DATA TO BE WRITTEN"
RESTORE #1, ADDR
WRITE #, FILECONTENTS$
```

4 RECORD ADDRESSABLE RANDOM (I.E.)

```
CREATE #1, "REC.RND/R=1000,60"
```

WILL CREATE FILE 1000 BYTES LONG OF 60 BYTE LONG RECORDS
POSITIONING (I.E.) TO THE 5TH 60 BYTE RECORD WOULD BE

```
RESTORE #1, 5
```

PART 4: 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 be 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) 462-5652. 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
    6304 Yucca Street
    Hollywood, CA 90028

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 DOS-68 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.

SYSTEM ACCESSORIES

The following accessories for the BFD-68 are available from Smoke Signal Broadcasting. Normally these items are kept in stock and are available for immediate delivery.

Additional 5" single-sided disk drive	$ 355.00
Additional 5" double-sided disk drive	$ 450.00
Additional 8" single-sided disk drive	$ 580.00
Additional 8" double-sided disk drive	$ 755.00
5" single-sided blank diskettes (Box of 10)	$ 50.00
5" double-sided blank diskettes (Box of 10)	$ 70.00
8" single-sided blank diskettes (Box of 10)	$ 60.00
8" double-sided blank diskettes (Box of 10)	$ 90.00
Cooling fan	$ 18.00
DOS-68 program source listings (transient command and monitor listings. DFM listing not available)	$ 30.00
BASIC compiler This BASIC is a comprehensive business oriented basic for serious BASIC users	$ 325.00
BASIC compiler manual	$ 10.00
SE-1 Text Editing system (diskette)	$ 29.00
SA-1 Mnemonic Assembler (diskette)	$ 29.00
SE-1/ SA-1 Editor/Assembler combination (diskette)	$ 53.00
TP-1 Text Processor (diskette)	$ 39.95
TD-1 Trace-Disassembler (cassette) (add $5.95 for diskette)	$ 19.95
SG-1 Source Generator (cassette) (add $5.95 for diskette) SG-1 is a disassembler which reconstructs source files which can be directly assembled or edited.	$ 24.95

Prices subject to change without notice.
Delivery of double sided disk drives subject to delay.

USER GROUP INFORMATION

Smoke Signal Broadcasting operates a 6800 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 6800 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.

APPENDIX A

DFM68 FUNCTION CODES

CODE NUMBER	CODE NAME	MEANING
Ø	QFREE	*Report amount of free space on a disc
1	QSO4W	Open a file for write (create a file)
2	QSWRIT	*Write data to a file
3	QSWC	Close a file open for write
4	QSO4R	*Open a file for reading
5	QSREAD	*Read from a file
6	QSRC	*CLose a read file
7	QDEL	Delete a file
8	QREN	Rename a file
9	QAPP	Append two files
1Ø	QDIRI	*Open a disc directory
11	QDIRT	*Retrieve a file name from the directory
12	(reserved)	
13	QRAFC	*Read Active FCB chain
14	(reserved)	
15	(reserved)	
16	QLOGD	*Login a new system disc
17	QLOGE	*Examine logged in drive number
18	QSSR	*Single sector read
19	QSSW	*Single sector write
1C	QERF	Expand random file
2Ø	QCRF	Create random file
21	QORF	Open random file
22	QPRF	*Position random file
23	QRRF	*Read random file
24	QWRF	*Write random file
25	QCLSRF	Close random file
26	(reserved)	
27	(reserved)	
28	(reserved)	

* means the function processor is memory resident.

The non-resident functions are kept in three files as follows:

FILE	FUNCTIONS
DFM68O.??1	QSO4W
	QSWC
DFM68O.??2	QDEL
	QREN
	QAPP
DFM68O.??3	QCRF
	QORF
	QCLSRF
	QERF

where ?? represents the revision number of DFM

GENERAL

USE RANDOM FILES

CREATE #EXP, "STRINGEXP" OPEN #EXP, FILENAME/R
OR
FILE NAME/R=???

RECORD ADDRESSABLE

FILENAME$ CAT "/R=" CAT NUM$(RECORDS) CAT "," CAT NUM$
(BYTES

(RECORDS CONTAINS # OF RECORDS DESIRED)
BYTES CONTAINS # OF BYTES IN EACH RECORD)

RANDOM FILES ARE ZERO-BASE (FILE W/ N BYTES = N+1 BYTES LONG)

APPENDIX B

FILE TYPE CODES

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 both FCBs and FIBs.

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

FILE STATUS CODES

The following table enumerates the file status codes. These are the only valid file status codes which can appear in the lower four bits of the file status (XFS) in both FCBs and FIBs:

NAME	VALUE	TYPE
FANA	Ø	Not active
FASR	1	Sequential read status
FASW	2	Sequential write status
FARA	4	Random access status

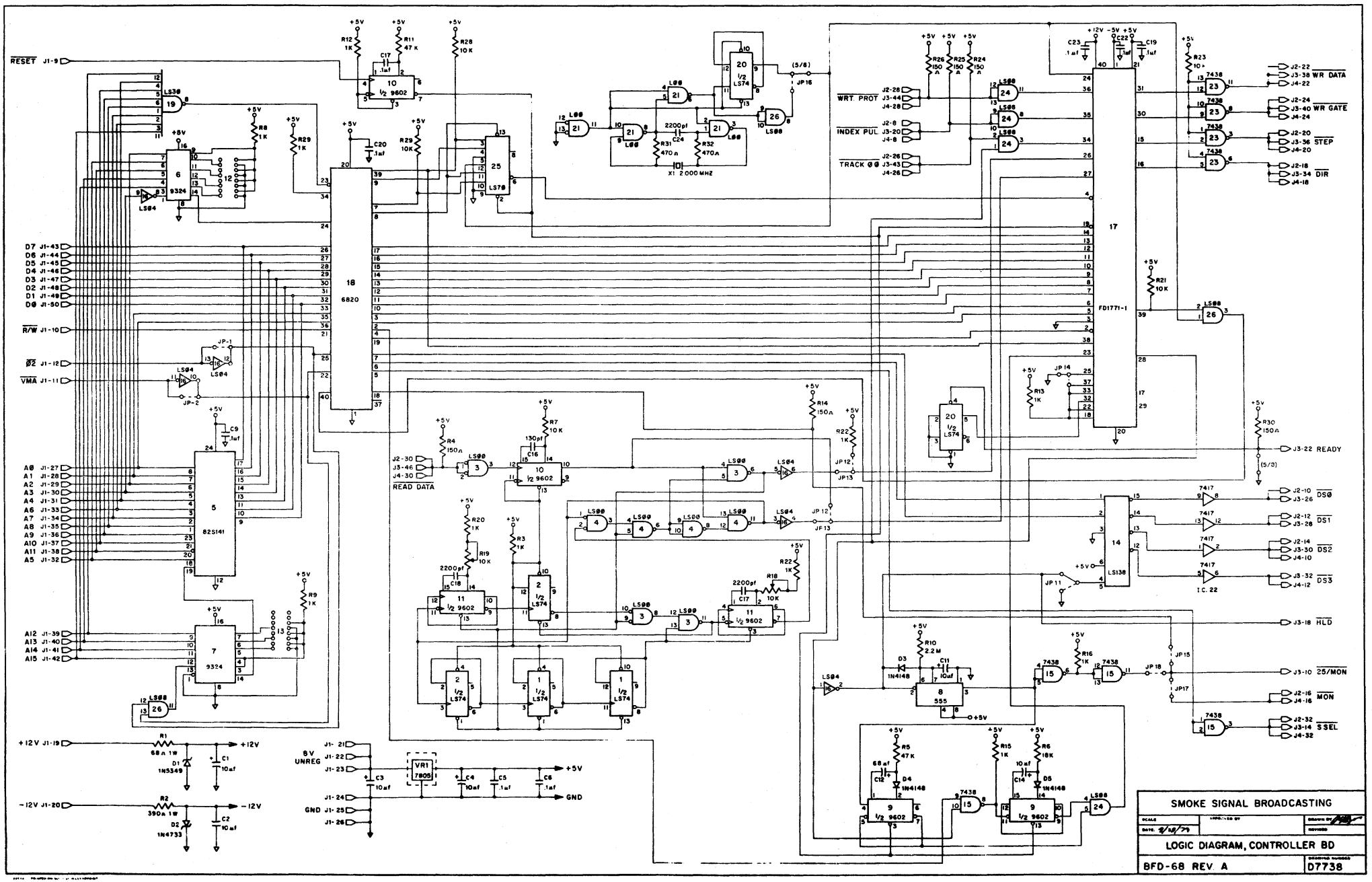
APPENDIX C

DFM ERROR CODES

ERROR NUMBER	ERROR NAME	MEANING
$Ø1	EIFC	Invalid DFM function code
$Ø2	EFE	File exists
$Ø3	EFIB	Master file directory error
$Ø4	EFB	File is in use
$Ø5	ENSF	No such file exists
$Ø6	EEOF	End of file
$Ø7	EDF	DISC full
$Ø8	EIF	Invalid file control block (FCB) address
$Ø9	EIFN	Illegal file name
$ØA	EFS	File status error
$ØB	EITS	Invalid T# or S# on QSSR or QSSW
$ØC	EIUN	Invalid unit number (only Ø, 1, and 2 allowed)
$ØD		(unused, better left alone)
$ØE	EDR	Disc read error
$ØF	EDW	Disc write error
$1Ø	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=Ø or is too large
$15	EDWP	Disc is write protected
$2Ø	ENSD	Non-system disc in logged drive
$21	ESFF	System file format error (should not occur)
$22	ECSS	Checksum error on system file

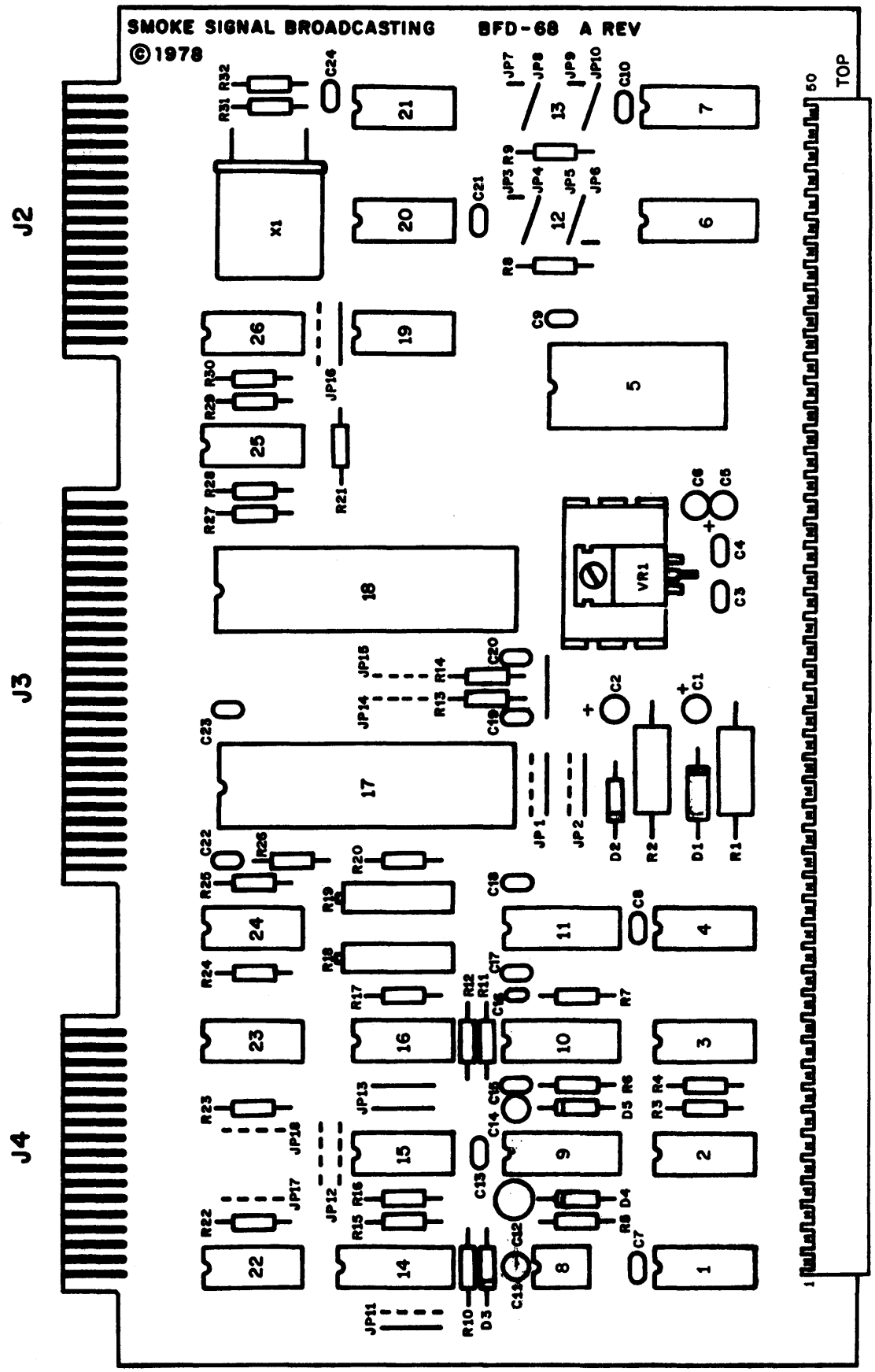
MONITOR CLOSE COMMAND ERROR CODES

ERROR NUMBER	MEANING
Ø1XX	Illegal file activity, i.e. the file is not open for sequential read, sequential write, or random acc XX = file status
Ø2XX	(UNUSED)
Ø3XX	File closing error XX = Closing error code (see DFM error codes)
Ø4XX	Read error on free chain descriptor XX = error status from the 1771
Ø5XX	Write error on the free chain descriptor XX = error status from the 1771


RESET J1-9
D7 J1-43
D6 J1-44
D5 J1-45
D4 J1-46
D3 J1-47
D2 J1-48
D1 J1-49
D0 J1-50
R/W J1-10
Ø2 J1-12
VMA J1-11
A0 J1-27
A1 J1-28
A2 J1-29
A3 J1-30
A4 J1-31
A6 J1-33
A7 J1-34
A8 J1-35
A9 J1-36
A10 J1-37
A11 J1-38
A5 J1-32
A12 J1-39
A13 J1-40
A14 J1-41
A15 J1-42
+12V J1-19
-12V J1-20
8V UNREG J1-21 J1-22 J1-23
GND J1-25 J1-26
J2-30 J3-46 J4-30 READ DATA
WRT PROT J2-26 J3-44 J4-28
INDEX PUL J2-8 J3-20 J4-8
TRACK Ø J2-26 J3-43 J4-26
X1 2000 MHZ
J2-22 J3-38 WR DATA J4-22
J2-24 J3-40 WR GATE J4-24
J2-20 J3-36 STEP J4-20
J2-18 J3-34 DIR J4-18
J3-22 READY
J2-10 DSØ J3-26
J2-12 DS1 J3-28
J2-14 J3-30 DS2 J4-10
J3-32 DS3 J4-12
I.C. 22
J3-18 HLD
J3-10 25/MON
J2-16 MON J4-16
J2-32 SSEL J4-32
SMOKE SIGNAL BROADCASTING
LOGIC DIAGRAM, CONTROLLER BD
BFD-68 REV A
D7738

APPENDIX E

CONTROLLER PARTS LOCATION



APPENIX F

CORES PATCHES

Load the SWTPC co-resident editor and assembler from cassette tape. Our version contained a 1B7C in locations 00FE and 00FF. type "GET, CORES.PAT". Now type "SAVE,CORES,FE,1D7A,100. to run, type "RUN, CORES". CORES will operate in the same fashion as before except that when "SAVE" or "LOAD" is typed followed by a carriage return, the system will prompt with "D OR OTHER?". if you wish to save or load from the disc, type "D". Any other character will cause the unit to save or load from cassette. If you type "D", the system will ask for a filename and then save or load to that file.

When running the assembler, the program will ask for an output filename and will create a file on disc in hex format. The source program must contain an "OPT O" statement or an output file will not be created. When you want to load that file into memory, type "GETH,FILENAME". Then exit to MIKBUG and go to the beginning of the program in the normal MIKBUG manner. remember, the file created by the assembler is in hex format and takes twice the normal amount of disc space. If the assembled program is a finished program that you expect to keep and use often, you may wish to load it into memory with the GETH command and then save it in binary form using the SAVE command.