MSC 8303

MONOLITHIC SYSTEMS OPERATING SYSTEM
USER'S MANUAL
(MSOS)

14 Inverness Drive East
Englewood, Colorado, 80112
(303)-770-7400

CHAPTER 1

GENERAL

Introduction

MSOS (Monolithic Systems Operating System) is a file management and program development system for use with the MSC 8001 Single Board Computer family. MSOS provides the user with a powerful tool for loading and saving program files and performing simple I/O on a variety of devices. The built in debug capabilities allow the user to get application programs up and running with minimal effort.

MSOS occupies less than 6K bytes of EPROM on the MSC 8001 board. ROM residence eliminates the time consuming bootstrap and restart procedures common to the early stages of program development. The operating system requires approximately 256 (decimal) bytes of user RAM memory for temporary storage, plus a 156 byte RAM buffer for each I/O channel. All other user memory is undisturbed by the operating system. The diagram on the following page outlines the memory map utilized in the MSC 8001 computer.

Commands are typed on the keyboard or read from disk storage, and direct the system to load or save programs from mass storage, examine registers or memory locations for debug purposes, or perform other monitor operations. The system contains a complete set of commands for maintaining the user file library.

MSOS commands are uniform from processor to processor in the Monolithic Systems Single Board Computer family. Users may transfer their MSOS experience from processor to processor with ease.

User programs may access all I/O devices known to the system via character oriented routines. It is also possible to pass system commands from the users program for immediate execution by the monitor. The user may link in additional system commands and I/O devices to the system at run time. These additional commands and device drivers may be RAM or ROM resident.

Startup Procedure - Starting the monitor

1. The firmware ROM(s) containing MSOS must first be inserted into the processor board. A serial I/O terminal such as a Teletype or CRT must be connected to the board as described in the appropriate section of the hardware user's manual. This I/O terminal will be referred to as the system console. Some boards require an MSOS compatible addressing prom.

0

MSOS ROM

17FF

1800

MSIL ROM (OPTIONAL)

1FFF

2000

SYSTEM RAM

27FF　USER STACK

2800

ON BOARD USER RAM

3FFF

4000

ADDITIONAL USER RAM

7FFF　24K BYTES MINIMUM RAM TO

8000　RUN ALL SUPPORTED SOFTWARE

ADDITIONAL USER RAM

OR USER ROM

FFFF

Figure 1. MSC 8001 Memory Allocation Map

2. When the board is first powered up, and after a CPU RESET, MSOS will determine the baud rate at which the terminal is operating. The operator must type two or three "carriage returns", slowly, to let MSOS find the correct baud rate from any of the following: 9600, 4800, 2400, 1200, 600, 300, 150, and 110 baud.

   When MSOS finds the correct rate, it will print MSOS REV n.n, the Date, the size of memory, and then prompt the operator for commands with a question mark (?). The prompt ? is a signal to the user that the monitor is in idle (keyboard mode) and a user command is required. "n.n" is the revision level for this version of the monitor.

   After the monitor prints its version number it will print what it thinks is the current date. If this is wrong, it should be corrected using the DATE command.

3. MSOS will re-establish the baud rate whenever an ESCAPE character (marked as ALTMODE on some terminals) is typed. ESCAPE signals MSOS to perform a "cold start". The ESC key also closes any open channels.

CHAPTER 2

OPERATION

Command Format

The following pages describe the resident monitor commands. Additional commands may be linked in at any time (see Chapter 4, Adding User Commands).

A command is comprised of a KEYWORD, followed by one or more PARAMETERS with separator characters between them, followed by a terminator character.

The KEYWORD is recognized by the first two characters, allowing abbreviations. Misspelling of the command from the third character on is not examined.

The number and kind of the PARAMETERS in each command is described in this section with the command. Command parameters are described in an abbreviated notation to simplify the use of this manual as a reference guide during system operation. An item enclosed in square brackets represents an item that is to be filled in by some actual character string supplied by the user. For example:

```
[fn]    Indicates that a file name is required
[addr]  Indicates that a memory address is required.
[chan]  Indicates a channel number is required.
```

An item enclosed in angle brackets < > means that that item may be optionally provided by the user. For example:

```
<[fn]>  Indicates that a file name is optional.
```

Command parameters must be either a file name or numeric field. File names and numeric fields are described in the following sections.

Some examples of valid commands follow.

```
ASSIGN Ø2 DW UPROG
AS    Ø2   DW   UPROG    (equivalent to above)
HEXLOAD AØFF
MODIFY  AØFF Ø 1 2 3
```

The elements of a command may be separated with any of the following characters:

SPACE	( )	EXCLAMATION MARK	(!)	LEFT PAREN	(()
COMMA	(,)	QUOTATION MARK	(")	RIGHT PAREN	())
HYPHEN	(-)	POUND SIGN	(#)	PERIOD	(.)
PLUS	(+)	DOLLAR SIGN	($)	APOSTROPHE	(')
SLASH	(/)	PERCENT	(%)	ASTERISK	(*)

It is recommended for clarity that the user utilize separator characters from the left most set above (sp , - + /). In some system commands, the plus (+) and hyphen (-) have special meaning.

These separator characters can be surrounded by any number of blanks. The following examples are equivalent.

```
LOAD JERRY
LOAD,JERRY
LOAD,    JERRY
LOAD  ,  JERRY
LOAD/JERRY
```

But the following line is not a legal entry as it contains 2 separators between elements:

```
LOAD  *  ,  JERRY
```

A system command must be terminated by a carriage return, or an ampersand (&). The latter is used to enter several system commands on a single line. These are then executed in sequence. There is a command line limit of 80 characters to a line.

When a carriage return terminator is sensed, the command line is executed by the system. If the system does not recognize the command, the monitor will print the following:

> ERROR
> The line in error up through the illegal character.
> ?

While entering a command string the following characters may be used to correct errors before the line is terminated:

Ctrl H, Backspace	Deletes the previous character.
Ctrl A and DELETE	Echoes a \, deletes previous character.
Ctrl X	Delete the entire line, but do not exit the mode (such as RE) you are in.
Ctrl C	Cancel command, exit to keyboard mode, issue a prompt. (warm start)
Ctrl [, ESC	Cancel command, close all files, perform a cold start.

Files

Programs or data stored on the disk are organized into discrete areas of storage called FILES. A FILE is a group of blocks linked together which is referenced by a file name. When a file is created, an entry is created in the file directory containing the name of the file, and the first and last block linked to the file. Each directory entry utilizes an entire block, therefore there is an overhead of 124 bytes for each file on the diskette.

There is no limit to the number of files which can be stored on a disk, other than available disk space. MSOS maintains a special file (FREE. USR), which is a linked list containing all the free space currently not in use on the disk. When a file is created, blocks from the beginning of FREE.USR are removed from the FREE.USR list, and assigned to the file. When a file is deleted, the blocks from the file are attached to the beginning of FREE.USR.

File Name Formats

A file name consists of 3 parts: the UNIT NUMBER, the NAME, and the EXTENSION.

The NAME consists of from 1 to 8 characters. The characters may be any ASCII character that is not a separator. It is suggested that the characters used be restricted to numbers and letters to maintain compatability with future versions of the monitor. The name is the only part of the filename that is necessary.

The EXTENSION is a 1 to 3-character abbreviation for the type of file being referenced. Except for the length, the EXTENSION follows the same construction as the name. The EXTENSION follows the name and is separated from it by a period. The monitor doesn't require any particular extension for any type of file. However, it is strongly recommended that the user choose a convention for filename extensions and stick to it.

The monitor will assign an extension to any filename that doesn't have one. If the filename is used in a LOAD, SAVE, or ZAP command, the default extension is ".SAV". In all other cases the default is ".USR". Note that a system program running under the monitor (i.e., an Assembler) may have its own defaults.

The UNIT NUMBER, if present, precedes the name. A file may reside on any disk unit. If a file resides on the primary disk unit, the UNIT need not be specified. When a file is referenced in a command and the file does not reside on the primary disk, the unit number upon which it does reside must be explicitly specified by use of the colon (:) followed by the unit number. For example, the following commands are identical except that in the second command the requested file is located on the second disk.

```
?LOAD PROG       PROG resides on primary disk unit
?LOAD :1,PROG    PROG resides on second disk unit
```

Unit Ø is assumed to be the primary disk, and therefore is interpreted to be the same as if no unit number were supplied.

Examples of file names

:1, BASIC.SAV	BASIC resides on unit one, and was created with a SAVE command.
TEMP	Minimum format, default unit is Ø, default extension is .USR.

Don't Care Character

A special convention has been adopted to improve the flexibility of file references. This is the question mark (?) or so called "don't care" character. The question mark will match any character in that particular character postion. This is true for any operation which references a file name. It is particularly useful for the ZAP command discussed in the next chapter. For example

```
LOAD JERRY.???
```

will load the first file found with the name "JERRY", regardless of the extension.

Numeric Fields

This section contains the proper format for entering numbers as parameters in system commands.

The number system used by the system is hexadecimal. A number is separated from neighboring items by separator characters. Under some circumstances certain separator characters may have special meaning. However, unless specifically mentioned, any separator may be used.

ADDRESSES -- Most numeric entries will be addresses. An address is an unsigned hexadecimal integer in the range ØØØØ to FFFF. When a numeric entry can be either a BYTE (see below) or an ADDRESS, the field will be interpreted as an address if it contains three or more digits. Addresses are represented internally as 16 bit binary quantities.

BYTES -- Some numeric fields may be BYTES. A byte is an unsigned hex integer restricted to the range ØØ through FF. A BYTE interpretation will be assumed if the number is composed of less than three digits. The internal representation for a byte is an 8-bit binary quantity.

BLOCKS -- Some numeric fields allow entry of a pair of addresses which represent the first and the last memory location in a block of contiguous memory. There are two forms of entering a BLOCK field--absolute and offset:

The ABSOLUTE form of a BLOCK field is the starting address followed by the ending address for the block, e.g.,

```
12ØØ,14ØØ
```

The OFFSET form of a BLOCK field is the starting address followed by a plus sign (+) followed by a 16-bit hexadecimal number which will be added to the starting address to derive the ending address, e.g.,

1200+3FF

There is an indirect capability for addresses. If the user prefixes an address with an at sign (@), the contents of the address and the following byte will become the effective address. E.g., @200 will refer to the address contained in the pointer at locations 200 and 201.

Note that the @ may be repeated any number of times. This consturction is useful whenever the address of the address needed is what is known.

Examples of valid numeric fields:

123	valid ADDRESS, invalid BYTE
12	valid ADDRESS, valid BYTE, defaults to BYTE if either is permitted.
012	valid ADDRESS, invalid BYTE
@10	INDIRECT address
@@1200	doubly-INDIRECT ADDRESS
100,200	BLOCK field, beginning and ending addresses.
100+1FF	BLOCK field, beginning address plus offset.
@100+1FF	BLOCK field, first address is INDIRECT
100+@1FF	BLOCK field, length of block is INDIRECT

Special Character Functions

Certain control characters perform special functions during output to Channel 1 from MSOS. They are described as follows:

Ctrl C	Cancel command, same as in edit form.
Ctrl S	Stop output and wait for Ctrl C or Ctrl Q.
Ctrl Q	Continue from Ctrl S.

CHAPTER 3

SYSTEM COMMANDS

ASSIGN Assign command

Formats: ASSIGN [chan] [dev] <[fn]>   Examples: ASSIGN 1 LP
AS 1,DW PROG.TST

The ASSIGN command assigns a device or file to one of the I/O channels. When the logical device is a mass storage device (disk), the optional file name must be supplied. [chan] may be 1, 2, 3, or 4. [dev] is a two letter designation of which device is to be assigned to the indicated channel. The present ROM system has the following device codes:

TT	Terminal, full duplex
LP	Line printer
DR	Disk, mode = Read
DW	Disk, mode = Write
DA	Disk, mode = Append
PI	Parallel Input port
PO	Parallel Output port

The device characteristics are described more fully in Chapter 4.

All user I/O operations will require an ASSIGNment before a data transfer can take place. If an I/O operation is attempted on an unassigned channed, MSOS will type--

```
ASSIGN [chan]  ?
```

where [chan] is the channel on which the I/O is being attempted. MSOS then pauses for the operator to enter a device name, and, where applicable, a file name. Once the assignment is accepted, the I/O operation will proceed without further intervention.

There is a special channel, channel Ø, which is the same as the other channels except for the fact that it is preassigned to the system command buffer. Reading from channel Ø reads the last command string typed. Writing characters to channel Ø is equivalent to typing them on the system console for execution.

BLOCK Block Move Command

Format: BLOCK [block] [dest]   Examples: BLOCK 2000-2400 5000
BL 2000 2400 5000
BLOCK 2000+400 5000

The BLOCK command moves a block of contiguous memory to another location. [Block] specifies the boundaries of the memory region which is to be moved, and [dest] is the new starting address for the block. The new starting address may be contained within the old block.

Example--To move a block of memory which is at 2000 (hex) and which extends to location 2400 (hex), to a new region which is origined at location 5000 (hex), enter:

```
        BLOCK 2000-2400 5000
(or)    BL 2000 2400 5000
(or)    BLOCK 2000+400 5000
```

The first example shows the source block specified absolutely, giving the actual addresses for the limits of the block. The 2nd example is analagous, but the command name is abbreviated, and a space is used as a separator instead of a hyphen. The 3rd example is the same as examples 1 and 2, but offset addressing is used to specify [block].

CONTINUE Continue command

Format: CONTINUE   Examples: CO
CONT

The CONTINUE command resumes execution following a software trap function which transfers control to the monitor. The program is returned to execution with the machine state identical to when the program trap occurred. See the REGISTERS command regarding altering the processor status before returning to execution.

DATE Date Set Command

Format: DATE <[month]/[day]/[year]>   Examples: DATE 5/6/78
DA 5/6/78
DA

This command sets the system date, which is recorded in the directory entry for each file when it is created. DATE should be executed immediately after a cold start. If a date is not supplied with the DATE command, the system prints the current date on the console.

DUMP Dump Command

Format: DUMP [block]   Examples: DUMP Ø,FF
DUMP 53-5AA

DUMP prints out the contents of memory to channel 1 in hexadecimal format, 16 bytes per line, with each line preceded by the address of the first byte on the line. Channel 1 would typically be assigned to the console or line printer. For example :

AS 1,LP & DU Ø,FF

would be used to dump the contents of Ø through FF to the line printer.

ECHO Echo command

Format: ECHO [count]&[command string]   Examples: ECHO 2 & ZAP PROG, 2

ECHO repeats all commands to the right of it [count] times. The ampersand is required to set off the ECHO command from the command which follows it and will therefore be repeated. The ECHO may not be requested to repeat another ECHO command, vix:

ECHO 1Ø & ECHO 2 & CONTINUE

is illegal. An ECHO with a count of zero will ECHO the following command repeatedly. A typical ECHO type in would be:

ECHO & ZAP PROG,2

which will excute the ZAP command repeatedly. The command will execute until the ZAP PROG,2 produces an error when all the requested editions of PROG are deleted.

EXAMINE Examine Command

Format: EXAMINE [block] [data]...[data]Examples: EX 8ØØØ-9ØØØ Ø1AB
EX 3AH-3BH CD

The indicated block of memory is scanned for the [data], which may be either byte or address. For each instance of the data, the address where it is found will be printed on the system console.

FILES Files Command

Format: FILES <[unit]>   Examples: FILES :1
FI

Files is the command which lists the directory of files assigned to a disk unit. The optional [unit] must be the unit number of a disk drive with a disk inserted. The default value of [unit] is Ø, the primary disk unit.

The first line of printout is a header line describing the information to follow. The header line is

FILENAME FIRST LAST SIZE DATE

where

FILENAME	is the NAME of the file, including the extension
FIRST	the block number of the first block in the file
LAST	the block number of the last block in the file
SIZE	the number of blocks in the file
DATE	date the file was created

The first file listed in the directory is always a special system file "FREE.USR" FREE.USR is a linked list containing all the free space on the diskette. The size of FREE.USR is the amount of space available on the disk. The date of FREE.USR will always be the last date when data was written on the disk. The remaining files listed on the directory are user and system program files.

The logical block number (as listed in the directory) corresponds to the physical block number in the following way: the first byte of the block number is the track number, the second byte is the sector number.

File Editions

When a user SAVEs a file with the same file name as a file already on the diskette, a new EDITION of the file is created (i.e. it is linked to the front of the directory). A file is never deleted automatically by the system.

The only edition of a file which can be loaded is the most recent edition which is the first file of that name listed in the directory. Older files can be accessed for loading only by ZAPping or renaming all newer editions. Refer to the ZAP command for further discussion of file editions.

GOTO GOTO command

Format: GOTO <[addr]> Examples: GOTO 3AH
GO 1000

The user stack pointer will be set to the default value (2800H). The monitor actually executes a subroutine jump to the specified address, therefore the last value on the stack will be a pointer to the return loop, and a return will transfer control back to the monitor.

If no address is given, the last address used by a GO or set by LOAD or REG, is used.

HEXLOAD Hexload Command

Format: HEXLOAD <[addr]> Examples: HEXLOAD 1000
HE 100

Hexload loads a standard hex format absolute binary tape or file into memory. If the optional address is specified in the command, the file will be loaded at that address.

The tape will be loaded from channel 1. If channel 1 is ASSIGNed to the console device (TT), an XON code will be sent to start the paper tape reader at the start of the load, and an XOFF code will be sent to stop the reader when the code is complete. The tape or file will be assumed to be in the standard hex format for the processor in use. Appendix D describes the standard hex format for the Z80 and 6800 processors.

IO Read, Send 8 bit data quantity command (Z80 only)

Format: IO [addr] <[byte]> Examples: IO FF (read)
IO FD, 4F (send)

This command is used in the Z80 version to send or read a character from the IO port [addr]. If the optional [byte] is not supplied, the character will be read from the port and displayed on the console. If the optional [byte] is supplied (in hex format), the character will be written to the IO port specified. Because of processor differences, this feature is not needed in the 6800 based versions.

LOAD Load Command

Format. LOAD [fn] Examples: LOAD PROG
LO :1,PROG1.TST

This command loads a program from the disk into memory. The program must have been written on the disk by a system SAVE command. LOAD sets the address used in the next GO command to the starting address of the program. Therefore the following sequence would be used to load and execute a program:

```
?LOAD [fn]
?GO
```

JAM Jam command

Format: JAM [block] [data] . . . [data]   Examples: JAM 100-300,0
JA 1000+8,1,2,3,4,5,6,7,8

JAM will fill a block of memory with the given data. JAM 8000-8100 00 will clear all locations from 8000 to 8100. The list of data can be any length that doesn't overflow the line.

If while jamming data into the block, the end of the block is reached before the end of the data list is reached, the JAM will continue, overflowing the block, until all of the data in the list have been stored.

If the end of the data list is reached before the end of block is encountered, the entire data list will be repeated, as needed to fill the block.

MODIFY Modify command

Format: MODIFY [addr] [data] . . . [data]   Examples: MO 3A 0D
MODIFY 100,0,1,2,3

The MODIFY command is used to change the contents of memory locations. Starting at the [addr] given, the [data] is stored into memory. If [data] is a byte, it is stored into the next memory location and the location pointer is incremented by one. In the Z80 version if [data] is an address, the least significant half is stored first, then the most significant half is stored and the location counter is incremented by two. In the 6800 version, this order is reversed.

The command line is terminated with a carriage return. However, the carriage return does not exit MODIFY mode, so that the system will print the address of the next loaction to be stored and will accept further data to be stored. If the user responds with another carriage return or a system command instead of data, MSOS will exit MODIFY command mode and execute the command if supplied.

NAME Name Command

Format: NAME [fn] [new name] <[date]> Examples: NAME PROG, PROG.TST
NA :1,PROG PROG1 4/23/78

This command allows the user to change the name and/or extension of a disk file. A unit number may not be supplied on the [new name]. If the file was not dated, the date will be set to the system date. If the file was dated, the date will be unchanged unless the optional [date] was supplied, in which case the date will be set to the [date] specified.

NEWDISK Newdisk Command

Format: NEWDISK <[unit]> Examples: NEW
NE

This command is used to format and initialize a new diskette. Because this command destroys any data that may have been recorded, NEVER use this command except when a new, unformatted disk is to be initialized. The exception is when a formatted disk has been "crashed" and must be reinitialized.

After the NEWDISK command has been typed, the system will print:

NEW DISK INSERTED?

The user must respond "Y" to start formatting. Any other response will abort the command.

This command will require several minutes to execute since the entire disk surface will be written and verified.

PUNCH Punch Command

Format: PUNCH [block] <[addr]> Examples: PU A000 +1FF, A000
PUNCH 100-200

The PUNCH command formats the indicated block of memory into a standard absolute hex format binary file, and transmits it on channel #1. If channel #1 is ASSIGNed to equipment TT, an XON code will precede the file, and an XOF code will terminate the file. The file begins and ends with two feet of null characters.

In the Z80 version only, if the optional [addr] is given, this will be used to set the starting address for the file.

A typical command would be:

PU A000+1FF A000

which will punch out locations A000 through A1FF as a hex load file with a start block of A000.

If the PUNCH is to a disk file, the file can be loaded again for use via the system HEXLOAD command.

REGISTERS Registers command

Format: REGISTERS

Examples: REGISTERS
RE

The REGISTERS command allows the user to examine and change the contents of the named CPU registers. The actual registers are of course different from processor to processor but the method of operation is the same. The monitor prints out a header line listing the CPU register mnemonics. Beneath this line, the current contents of the registers are displayed. The user changes the contents of a register by positioning the carriage or cursor below the appropriate value and typing the new value. The cursor may be positioned by repeatedly pressing the space bar, or by pressing tab (control I). The tab character will automatically adjust the cursor to the next register column position. A register value is left unchanged by spacing or tabbing past it or typing a carriage return before reaching the register position.

A carriage return with no new values typed leaves all register contents unchanged. If the user types a system command in response to the registers command, the system will exit register mode without changing any values and execute the new command.

Register commmand format for the Z80 microprocessor

In the Z80 based CPU, the REGISTER command might produce the following print out:

```
PC    PSW   BC    DE    HL    SP    IX    IY    I
790B  1E28  DE01  95DA  DC00  E9A3  0000  0A3F  00
```

where

PC	is the processor program counter (16 bits)
PSW	is the processor status word (16 bits)
BC	is the B and C register pair (16 bits)
DE	is the D and E register pair (16 bits)
HL	is the H and L register pair (16 bits)
SP	is the stack pointer (16 bits)
IX	is the X index register (16 bits)
IY	is the Y index register (16 bits)
I	is the interrupt vector (8 bits)

The following printout would be produced if the user wished to set the Accumulator and register pair DE to 0.

```
PC    PSW   BC    DE    HL    SP    IX    IY    I
790B  1E28  DE01  95DA  DC00  E9A3  0000  0A3F  00
      0028        0000
```

Note that the AC and processor status word, and the register pairs, are treated as a single 16 bit data quantity. Therefore in the example above, the user had to supply 0028 to change the AC to 0 and leave the status word alone.

In the Z80, the SWAP command would be used to display the alternate register set.

Register command format for the 68ØØ processor

In the 68ØØ based CPU, the REGISTER command might produce the following print out:

```
PS  B   A   XREG  PC    SP
ØØ  3A  ØØ  3FFB  Ø1ØØ  Ø2AF
```

where

PS	is the processor status word (8 bits)
B	is accumulator B (8 bits)
A	is accumulator A (8 bits)
XREG	is the X register (16 bits)
PC	is the processor program counter (16 bits)
SP	is the stack pointer (16 bits)

RUN Run Command

Format: RUN [fn] <[param] . . . [param]> Examples: RUN PROG.TST
RU BASIC 1.2,Ø

This command loads and executes a program from the disk. It is equivalent to:

```
?LOAD [fn]
?GO
```

An attempt to RUN a file with no starting address will load the file but will not execute the program. An attempt to RUN a non-program file will result in an error message. [param] are optional parameters supplied by the user to be passed to the application program being executed. Chapter 4 describes this feature in more detail.

SAVE Save command

Format: SAVE [fn] [block]< . . . [block]><[entry address]>
Examples: SAVE PROG, Ø-1ØØ
SAVE PROG.TST, Ø-1ØØ

This command copies blocks of memory [block] to a disk file [fn], and optionally assigns an [entry address] which is the address of the initial instruction in the program. The default extension for [fn] is ".SAV"

Example: To save a program on disk from memory locations A424 through BØ15, with the initial instruction defined to be at A43Ø:

```
        SAVE JERRY A424-BØ15 A43Ø
(or)    SA   :Ø,JERRY.SAV,A434,BØ15,A43Ø
```

which are equivalent forms.

The SAVE command writes out the file in a compressed binary format to save space on the disk and to improve corresponding load time. A file that has been SAVEd can be loaded into memory again only by using the system LOAD command.

SWAP SWAP command (Z8Ø only)

Format: SWAP Examples: SWAP
SW

In the Z8Ø CPU, the SWAP command is used to SWAP in the alternate set of registers. The new registers are displayed as in REGISTERS and may be changed.

VERIFY Verify command

Format: VERIFY [fn] Examples: VERIFY ASM.SYS
VE EDIT.TST

The VERIFY command reads a file and checks that all sectors can be read without error. At the end of the verify, the number of soft errors encountered is printed.

The maximum number of soft errors allowed is 255. If "FF ERRORS" is printed, the system did not complete reading the file. If any lesser number was printed, the system can read the file successfully.

It is very unusual to get anything but zero when running this command.

If a disk has persistent errors, there are three possibilities:

1) the media is faulty. (Disks do wear out.)

2) the drive is faulty. A mis-alligned drive should particularly be suspected if the disk can be read on another drive.

3) the interface is faulty. This is more likely to be the cause of hard errors rather than soft errors. If one drive of a two drive system works, it is unlikely to be the interface.

ZAP Zap command

Format: ZAP [fn] [edition] Examples: ZAP PROG.TST,1
ZAP JERRY 2

The ZAP command is used to delete files from disk.

Whenever a new file is created, it is linked to the front of the directory. Thus an old file of the same name will not be deleted, but it cannot be accessed normally. The most recent edition of the file is the only one which can be accessed by file name. The edition number does not appear on the file directory, but is implicit in directory list position.

ZAP deletes a particular [edition] of a disk file. The edition number for all older editions of the file are adjusted by subtracting one.

[Edition] cannot be Ø, and cannot reference an edition number which doesn't exist.

A convenient method of deleting all editions from a given edition number upwards is to ECHO a ZAP command for the selected edition, e.g.,

```
ECHO & ZAP JERRY 2
```

will delete all but the most recent edition of JERRY since each execution of the command reduces all the existing edition numbers by one. To delete all editions of a file,

```
ECHO & ZAP JERRY 1
```

will suffice.

The "don't care" character is particularly useful in conjuction with the ZAP command. For example if a directory contained files

```
TEST1.JOE
TEST2.USR
PROG3
```

then the command

```
ECHO 2 & ZAP TEST?.???
```

would delete both TEST1.JOE and TEST2.USR. The "don't care" character may also be used to delete ambiguous file names such as those which can be erroneously created by an untested user program.

CHAPTER 4

USER ACCESS TO THE MONITOR

PERFORMING USER I/O

MSOS performs all I/O through routines which are accessible to user programs. This section describes the use of these routines so that a user may perform character oriented I/O to a device or disk file.

All I/O through MSOS is performed via CHANNELS. A channel serves as a data path for information from the monitor to the device and vica versa. There are five such channels in the present version of the monitor.

Each channel in the monitor is full duplex. This means that a bidirectional device such as a teletype can be assigned a single channel rather than one for each direction as in some systems.

All I/O is performed a single character at a time through one of ten monitor routines. The routines are as follows:

CHØR, CHØW, CH1R, CH1W, CH2R, CH2W, CH3R, CH3W, CH4R, CH4W.

A call to CHnR will read a character from the device assigned to channel n and place it in the A accumulator. The carry flag is set if the character read is the last one in the file (EOF conditon), otherwise it is cleared. No other registers are affected. The channel is automatically closed after the last character read. Further attempts to read on this channel will prompt a request from the system for a channel ASSIGNment.

A call to CHnW will write the character in the A accumulator to the device assigned to channel n. No registers, including the Accumulator, are affected except processor status. File oriented devices will actually buffer the data before writing onto disk, but this feature is transparent to the user.

Refer to Appendix B for the locations of these routines.

Channel ASSIGNment

Before character I/O can occur, the channel must be ASSIGNed to a device. There are three ways the user can ASSIGN a channel:

1. By using the ASSIGN command from the monitor while in keyboard mode prior to calling the user program.

2. By passing an ASSIGN command to the monitor while under user program control. (See Chapter 4.),

3. By calling CHnR or CHnW without pre-ASSIGNing, and then responding to the monitor ASSIGNment request.

CHØR, CHØW are special in the channel Ø is preassigned to the system command buffer. Reading via channel Ø will fetch the characters from the last command line. This feature is often used to pass RUN parameters. Writing to channel Ø is equivalent to typing them on the system console for execution.

The following describes the device codes and device driver characteristics for the devices available for user ASSIGNment:

TT Terminal. Full duplex, auto baud rate (11Ø-96ØØ), hard or soft copy, 72 or more character line with a minimum asynchronous serial communications. Must have "control" character generation available.

LP Line Printer. Output (write) character only. Otherwise same as TT.

PI Parallel Input. Accesses one of the 8255's on the 8ØØ1 board. Each read operation transfers a single 8 bit data quantity across the port.

PO Parallel Output. Accesses one of the 8255's on the 8ØØ1 board. Each write operation transfers a single 8 bit data quantity across the port.

Disk

Disk files can be accessed in three different ways. Each access type is treated as a separate device.

DR	Disk Read. Only read character is defined.
DW	Disk Write. Only write character is defined.
DA	Disk Append. Append is a write operation. Each write character call appends the character to the current contents of the disk file.

When a user is finished writing or appending to a disk file, the file must be "closed" by reassigning the channel used to the null (NU) or other device. This operation writes out any remaining information in the file buffer and updates the file directory.

Note that an attempt to write to a read device is ignored. This allows prompts that a user program may produce to be successfully handled. An attempt to read from a write only device will cause an ASSIGN request just as if the channel was closed.

ADDING USER DEVICES

User devices can be added at any time by adding a new entry in the device driver table. The format for an entry in the table is:

```
POINTER:        DB      "X","Y" ;Device name, two chars. this
                                 ;device is "XY"
                DW      OPEN    ;Pointer to open device routine
                DW      OUT     ;Pointer to character output routine
                DW      INPUT   ;Pointer to character input routine
                DW      CLOSE   ;Pointer to device close routine
                DW      NEXT    ;Pointer to next device entry in the
                                 table
```

Any routine that is not needed by a user device should have a "Ø" entered for its address. The monitor checks for a "Ø" address before calling the routine. The monitor saves all registers except the A accumulator so the routines may use any desired registers.

The location referenced by symbol DDD contains a pointer to the first device driver table entry. The user makes a new device known to the system by setting NEXT to the current contents of DDD, and then setting contents of DDD to the address POINTER. Therefore, the new device is linked in as the first device in the chain.

A cold start resets DDD to its original value, so the user has to link in the new device after every power-up or ESC. Refer to Appendix B, Useful Routines, for a listing of the absolute location of symbol DDD in this version of the monitor.

The user's OPEN routine can call GNC (see Appendix B for address of GNC routine) to fetch any characters after the device name in the command string. This allows file names or options to be passed.

ADDING USER COMMANDS

The programmer can link a new command into the command table at run time by providing a command sub table in the format:

```
        POINTER:        DB      "N","C" ;Command mnemonic, two characters
                        DW      NEXT    ;Pointer to next command
                                :       ;Code to be executed upon command
```

The system pointer to the first command is kept in location referenced by symbol CDT in private RAM. The user command is added by picking up the contents of CDT and saving it in the second pair of bytes in the new command table (NEXT). The contents of CDT should then be set to the address POINTER of the new command.

Whenever a cold start happens, the monitor restores CDT to its original value. Therefore, the user has to link in the new command after every power up or ESC.

AUTOLINK

If the MSIL rom is not installed, there is a procedure that allows new commands or IO devices to be automatically linked in at initialization. If location 1800H is 0A5H then the monitor will call 1801H after it has completed initialization but before it asks for a command. A user supplied routine can--among other things--set CDT and DDD to new values.

APPENDIX A SYSTEM ERRORS

When MSOS encounters an error, it prints a diagnostic message on the console and returns to keyboard mode. There are three distinct types of errors which may occur:

> Syntax errors - A syntax error occurs when the monitor encounters an illegal command syntax. No action on the user command is taken by the monitor.
>
> Fatal errors - A fatal error occurs when the monitor encounters some problem during command execution. The monitor makes every attempt to recover from the error by attempting to close all files, etc., but full recovery from the error is uncertain.
>
> Miscellaneous errors - A miscellaneous error is an error which does not fit in either of the above categories. The system can recover from a miscellaneous error. Trying to write on a write protected disk is one example of a miscellaneous error.

Syntax Errors

All syntax errors produce the error message "ERROR", followed by the command line which produced the error up through the character which caused the error. For example, if the user types

```
ASSIGN 2 TX
```

where TX is an illegal device mnemonic, the system would respond with

```
ERROR
ASSIGN 2 T
?
```

Fatal Errors

There are many possible fatal errors. In each case the system prints a diagnostic error message describing the error. The error messages and a discussion of possible causes follows.

Message	Meaning
BAD CHANNEL NUMBER	The user attempted to assign a device to an illegal channel number.
CAN'T DELETE FREE CHAIN	User attempted to delete the free chain file "FREE.USR".
CHECK CHARACTER ERROR,	The CRC character used to verify the accuracy of the sector on a disk was incorrect. This occurred while reading (or write verifying) a data block or directory block.
DISK FULL	The user is out of free space on the disk. This could occur during a write block operation, during an open file operation, or during a close file operation.
DISK POSITION LOST	Hardware error. Cannot find a sector. Sector counter logic is not working.
DISK SEEK ERROR	Hardware error. Cannot find a track. Servo moter head actuator is not working.
MEM	Both LOAD and HEXLOAD verify the data that has been loaded into memory. This error occurs when an error is encounter while verifying. Attempting to load into ROM or nonexistent memory locations will produce this error.
NO SUCH UNIT	Unit number supplied is illegal.
SECTOR NUMBER TOO BIG	Attempt to read or write on an illegal sector number. Usually a result of incorrect user call to disk write routi
SECTOR SIZE ERROR	Illegal sector byte count. Either for a data block or directory block. Usually a hard disk error while reading or incorrect call to disk write routine.

Message	Meaning
SERIAL RECEIVER ERROR (Z8Ø only)	An error has occured in the 8251 serial I/O chip. Typing too many characters during a disk access will cause this error.
TRACK NUMBER TOO BIG	Illegal track number. Usually caused by an incorrect call to disk read or write routine.

Miscellaneous Errors

Message	Meaning
FILE NOT FOUND -unit number and filename-	Requested filename could not be located in device directory.
LINE OVERFLOW	More than eighty characters typed on a command line or sent as a command line from a user program.
WRITE PROTECT	Attempt to write on a write protected disk. User should replace the write protect and then type a character. Operation will then proceed as normal.
CLOSE DOOR - HIT KEY	The user has left the door to the disk drive open. Close the door and hit any key.

APPENDIX B

USEFUL ROUTINES

This Appendix outlines the structure of the pointers to certain useful routines which may be called by the user. It is intended to serve as an outline only, for further details, consult the listing.

Channel I/O Table

Locations 3 and 4 in MSOS ROM point to the channel I/O Table. The channel I/O table contains the address of the CHnR and CHnW routines for each channel as described in Chapter 4. The structure of the table is as follows

CHØW:	address of	write	character	routine	for	channel	Ø (2 bytes)
CHØR:	"	read	"	"	"	"	Ø
CH1W:	"	write	"	"	"	"	1
CH1R:	"	read	"	"	"	"	1
CH2W:	"	write	"	"	"	"	2
CH2R:	"	read	"	"	"	"	2
CH3W:	"	write	"	"	"	"	3
CH3R:	"	read	"	"	"	"	3
CH4W:	"	write	"	"	"	"	4
CH4R:	"	read	"	"	"	"	4

Interesting Pointers Table

Locations 5 and 6 in MSOS ROM point to a table containing the addresses of other locations useful to the programmer. The contents of the table are as follows:

```
INPTAB: IOTP          ;address of the channel control table (see listing)
        CHAN          ;address of the universal channel call routine
        BREAK         ;address of routine which returns control to
                      ;monitor if control C key is struck
        CDT           ;address of command dispatch table
        DDD           ;address of device driver table
        TCHN          ;Ø if channel Ø closed, otherwise points to device
                      ;descriptor table for chan Ø.  The remainder of the
                      ;the channels follow in 2-byte increments.
        BLLA          ;Points to table of addresses for the
                      ;parameter blocks.
        DAT           ;Address of system date.
```

NOTE:

The monitor stores the size of user Ram in the first locations of private memory. (i.e. 2ØØØH in the Z8Ø)

Useful Routines Jump Table

Locations BH is the beginning of the Useful Routines Jump Table. Each entry in this table is a jump to a particular routine. The jump allows the programmer to transfer control via subroutine jump directly to the appropriate location in the table, thus avoiding the extra level of indirection. The structure of the Useful Routines Jump Table is as follows.

```
JMP     OUTB    ;output a character
JMP     GIC     ;input a character,strip parity,set carry if
                ;control character,and echo chr if not.
JMP     CRLF    ;issue a carriage return and line feed to channel 1
JMP     SPACE   ;issue a space character to channel 1
JMP     MSG     ;print a message pointed to by H and L registers
                ;(Z8Ø) or XREG (68ØØ) to channel 1
JMP     MSGI    ;print an in line message to channel 1.
                ;message must end in Ø byte
JMP     D8      ;Read a byte from the command line, return in
                ;H and L (Z8Ø) or XREG (68ØØ).
JMP     D16     ;same as above but reads a word from command line
JMP     BKPT    ;address of where to go in monitor if software
                ;trap occurs.
JMP     MONSR   ;warm start, resets stack is all
JMP     INI     ;initialize serial I/O
JMP     FAT3    ;closes all channels and jumps to MONSR.
JMP     TSTAT   ;sets carry if a key has been struck, clears it
                ;otherwise
Next four locations are reserved for future expansion.
JMP     VECS+21 ;NMI vector transver address.
```

APPENDIX C

SOFTWARE BREAKPOINTS AND USER INTERRUPTS

MSOS allows the programmer to generate software breakpoints to the monitor for debugging operations. Software breakpoints and user interrupts are vectored through private MSOS RAM. This Appendix describes the operation of these two features.

Software Breakpoints

In the Z80 processor, the programmer makes use of the RST0 - RST7 instructions to generate a software breakpoint. Each of these instructions with the exception of RST0, transfers control through the interrupt vector table to location BKPT which is the monitor breakpoint routine. Refer to Appendix B and the listing for the locations and format of this table. The monitor breakpoint routine takes the following action:

1) Save current machine state

2) Display the registers as if the REGISTER command was activated

At the point the user might alter register status, or just type CONTINUE to return control to the user program.

RST0 is special in that it transfers control to the monitor cold start location.

The easiest way to use the software breakpoint feature is for the programmer to assemble RST1 - RST7 instructions into critical locations in the user program. When these breakpoints are no longer needed, they can be replaced with a NOP instruction.

In the 6800 based processor, the SWI instruction is used to transfer control to the BKPT routine.

User Interrupts

In the Z80 processor, if the user requires hardware interrupt, he must set the RST1 - RST7 location to the address of the interrupt handler routine. On the execution of a cold start, these locations will be automatically reset to the MSOS BKPT value, therefore the user must set up the interrupt vector table after every cold start, possibly by using the Autolink feature.

An interesting debug technique is to set up a logic analyzer to generate an interrupt under a given set of software/hardware conditions. Upon detection of this condition, the logic analyzer can generate an interrupt which will transfer control to the monitor breakpoint routine.

APPENDIX D

HEXLOAD BINARY FORMAT

This Appendix describes the absolute binary object code format accepted by the HEXLOAD command and generated by the PUNCH command. The hex binary formats are the same as used industry wide for transmittal of Z80 and 6800 object tapes. Therefore, the HEXLOAD command may be used to load tapes generated at another location, and PUNCH should be used to generate tapes for transmittal to other locations.

Hexadecimal binary object code format is an ASCII representation of program memory, expressed as a series of hexadecimal digits. These are blocked into records, each of which contains the record length, type, memory load address, and checksum, in addition to the data. The descriptions below apply to paper tape on a frame by frame basis, or disk storage format, which is the same but eliminates the null characters between blocks.

6800 Format

For the 6800, there are three types of blocks which are separated by null characters if the output file is the paper tape punch. Each block starts with the character "S" followed by the characters for the rest of the block. The two block types are:

1. Data block
2. End block

Data Block Format

The format for a data block is:

S1ccaaaaddd...ss

where

S1	indicates that this is a data block
cc	is the count of the number of bytes in the block. This includes the checksum byte, the two address bytes, and all the data bytes.
aaaa	is the two byte address at which the data in this block is to load. Successive data bytes are stored in consecutive addresses. (High order digit first.)
dd	are the data bytes. There is a maximum of 16 data bytes in each data block. (High order digit first.)
ss	is the checksum byte. The checksum is the negative of the sum of all 8 bit bytes in the record, beginning with record length, and ending with the last data byte. Therefore, the sum of all bytes in the record (including the checksum) should be zero. Only the least 8 bits of the checksum are used.

End Block Format

The format for the end block is:

S9

Z8Ø Format

For the Z8Ø, there are two types of blocks which are separated by null characters if the output file is the paper tape punch. Each block starts with the character colon (:) followed by the characters for the rest of the block. The two block types are

1. Data Block
2. End Block

Z8Ø Data Block Format

The format for a data block is:

```
:ccaaaaØØdddd...ss
```

where

:	is the first character in every block
cc	is the count of the number of data bytes in the block. Note that as opposed to the 68ØØ, only data bytes are included here.
aaaa	is the two byte address at which the data in this block is to load. Successive data bytes are stored in consecutive addresses. (High order digits first.)
ØØ	block type code. All blocks are type Ø.
dd	are the data bytes. (High order digit first.)
ss	is the checksum byte. The checksum is the negative of the sum of all 8 bit bytes in the record, beginning with record length, and ending with the last data byte. Therefore the sum of all bytes in the record (including the checksum) should be zero. Only the least 8 bits of the checksum are used.

Z8Ø End Block Format

The End Block is really a special case of data block with a data count of zero. The format for the end block is

```
:ØØaaaaØØss
```

where

aaaa	is the starting address of the file if specified, Ø otherwise.
ss	is the checksum.

APPENDIX E

SAVEFILE FORMAT FOR MSOS

The save file format is a very compact way of storing memory blocks on disk. The format can be used with other media, but there is no error checking other than that supplied by the media.

A record starts with ØFFH. Any data prior to the ØFFH is ignored.

Next there is a 2 byte address (HI LO)

then a 2 byte data count (HI LO)

then the data--possibly an entire memory content.

If the count is Ø, the address is taken as the start address.

?