MicroDoZ™

The Microcomputer Disk Operating System for Z80™ Microprocessors

Developed for
users of the North Star™ Horizon™
and other Z80-based computers

by

Micro Mike's, Inc.
3015 Plains Blvd.
Amarillo, Texas 79102 USA

telephone 806/372-3633

manual release 1

October 17, 1980

making technology uncomplicated...for **People**

COPYRIGHT NOTICE

DISCLAIMER

Micro Mike's, Inc. makes no representation or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Micro Mike's, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Micro Mike's, Inc. to notify any persons of such revision or changes.

TRADEMARK NOTICES

This documentation package will mention several names and products that have been granted trademarks. It is the intent of Micro Mike's, Inc. to acknowledge and respect these trademarks of these companies so that all the rights and privileges of these companies are preserved.

baZic™ and **MicroDoZ**™ are registered trademarks of Micro Mike's, Inc.

North Star™ and Horizon™ are registered trademarks of North Star Computers, Inc.

Z80™ is a registered trademark of Zilog, Inc.

TABLE OF CONTENTS

INTRODUCTION

MicroDoZ was written and implemented by the staff of Micro Mike's, Incorporated as a replacement for North Star DOS. Micro-Doz is completely compatible with DOS but has many enhancements that make it more efficient to use and easier to interface to machine language programs.

For OEMs, MicroDoZ can easily be changed to include drivers for different computer systems and disk drives allowing all computers to have the benefit of programs that run under North Star DOS. MicroDoZ has been written with timesharing features so that users can easily advance from single user to multiple user configurations.

MicroDoZ has been written to work without regard to disk size. A two byte disk address lets MicroDoZ address over 33 million bytes directly and in conjunction with JOEDOS (Micro Mike's, Incorporated Hard Disk Operating System), millions of additional bytes through segmentation.

baZic, Micro Mike's, Incorporated Z8Ø BASIC interpreter is available to run under MicroDoZ as well as many application programs.

In this documentation, a block refers to 256 bytes while a sector can be many values but is generally 512 bytes.

1.1 Mnemonics

DEVICE# The number of an input or output device. Must be in the range of Ø to 7.

DRIVE# The number of a disk drive. Must be in the range of 1 to 7.

HEXADDR A 16 bit Hexadecimal address. Must be in the range of ØØØØH to FFFFH.

BLOCKS Blocks are divisions of disk space that are equal to 256 bytes.

DENSITY The density of a disk is either "D" for double or "S" for single.

FILENAME The name of a disk file. Must be eight characters or less. Can contain any letters, numbers, or special characters except a space or comma.

DISKADDR A decimal address of a sector on a disk.

COMMANDS

MicroDoZ contains all of the commands in the North Star DOS plus several additional commands which allow the user to set input or output default devices, define default drives, rename files, set and reset write only files, set and reset system files, and to assign up to 64 different attributes to a file.

Multiple MicroDoZ commands may be issued on the same line by separating them with a "\" or a ":". As an example, if the user wanted to list the directories of both drives in a two drive system, the command would appear as follows:

```
1>LI \LI 2
```

The preceding example also shows the prompt character for MicroDoz. The prompt is the default drive followed by the ">" character. The default drive can be changed by using the DD command, but any drive can be accessed by the standard syntax no matter which drive is the default drive.

2.1 LIst the directory

```
LI [#<DEVICE#>] [<DRIVE#>] [,<WILD CARD>]
```

The LI command is used to LIst the directory of a disk drive. If no drive is specified, the default drive will be LIsted. The default drive is the drive shown in the prompt of MicroDoZ.

The LI command results in the following information being displayed from left to right across the console device for each file listed in the directory of the affected disk:

File Name (maximum of eight characters)
Starting Disk Address (Decimal)
Length of the File in Blocks (Decimal)
Density (Single or Double)
Type of File (Ø to 127 Decimal)
GO Address (If Type 1 in Hex)
R/O if Read Only File
SYSTEM if System File
Attribute Field (Ø to 63 Decimal)

A sample directory would appear as follows:

```
1>LI

MICRODOZ      4    20 D   0                    AF  0
M2D00M       14    10 D   1 2D00               AF  0
TEST         19     2 D   2       R/O SYSTEM   AF  0
BAZIC        20    54 D   1 0100               AF  0
```

Output can be directed to any of the eight user defined print devices (0 through 7) if desired by using the optional device specification. The device number should be entered after the command, preceded by the number character ("#"), and before the drive number specification.

A "wild card" feature is included in MicroDoZ. If you want to list only those files that have a certain letter in the third position, place an asterisk ("*") in each position where any letter is to be listed. If you only enter three characters (i.e. **L) MicroDoZ assumes all character positions past the last letter to be wild card letters. All MicroDoZ commands using the directory can take advantage of this feature as long as disk calls are routed through the MicroDoZ command call (0005 Hex).

Examples of the LI command are as follows:

```
LI              (List drive one)
LI #2           (List drive one on device two)
LI 2            (List drive two)
LI #2 2         (List drive two on device two)
LI,**X          (List files with third letter= X)
LI#2 2,***J     (List files with fourth letter=J)
```

2.2 INitialize a disk

IN [<DRIVE#>] [<DENSITY>] [,<CHARACTER>]

The IN command is used to INitialize diskettes to be used in the system. If no disk drive number is entered the command works on the default drive. The DENSITY argument is passed as an "S"or "D" for single or double density. If the optional density argument is omitted the command initializes the disk double density. If the optional character is omitted the disk is initialized to the ASCII space character (20 Hex). The optional character can be used to initialize a disk to different characters such as E5.

This command writes the specified character to all blocks on the disk. This command will destroy all information on the disk and should be used with caution.

2.3 WRite disk or Read Disk

WR OR RD <DISKADDR> [,<DRIVE#>] <HEXADDR> <BLOCKS> [<DENSITY>]

These commands are used to WRite (WR) or Read Disk (RD) directly between the disk drive and internal RAM. The disk address is a decimal address within the range of the drive being accessed. If the optional drive number is not specified, the default drive is used. The hex address is the RAM address where the information is to be written to or from.

If the optional density argument is not specified then the read or write will be double density.

The use of the WR and RD commands should be done with caution since they are capable of destroying valuable information if the wrong arguments are passed. However, these commands can be very powerful when used for a disk to disk copy because the entire free RAM can be used as a buffer resulting in very fast disk copies.

2.4 Save File or Load File

SF OR LF <FILE NAME>[,<DRIVE#>] <HEX ADDRESS>

The Save File (SF) or Load File (LF) command is used to save or load a specified file directly between RAM and the disk drive. The file name must evaluate to a legal file name. If the optional drive argument is not specified, the default drive will be used. The hex address must be specified and is the RAM address where you want the file to be saved from or loaded to.

2.5 Jump Program

JP <HEX ADDRESS>

The Jump to a machine language Program (JP) command is used to directly execute a machine language program from MicroDoZ. Most programs executed in this manner will set up their own stack upon entry. However, MicroDoZ sets up a temporary stack directly beneath itself when this command is entered.

Also MicroDoZ pushes a ØØØØ Hex address into the stack before jumping to the program. This address is the return address to MicroDoZ. This allows a machine language program to return to MicroDoZ by simply executing a RETurn instruction when the program is finished.

2.6 GO program

GO <FILE NAME>[,<DRIVE#>]

The GO command is very similar to the JP command. The GO command causes a machine language program to be executed but there are certain considerations that must be met. Upon receiving the GO command MicroDoZ looks for the specified file name on the specified drive. If no drive number is specified, MicroDoZ will use the default drive.

Once the file is successfully found and it is a type 1 file, the file is loaded into RAM at the GO address (specified in the directory entry). MicroDoZ then sets up a stack directly beneath MicroDoZ as in the JP command. After PUSHing the RETurn address of MicroDoZ on the stack, MicroDoZ then jumps to the GO address of the file and begins executing the program just loaded from the disk.

MicroDoZ has provisions for executing an implied GO command. If any word is entered in response to the prompt and the first two letters are not MicroDoZ commands, then MicroDoZ assumes that a GO command is to be executed. MicroDoZ looks for a file by the same name on the specified drive and if the type is 1 the program will be loaded and executed.

An example of the implied GO command would appear as follows:

```
2>BASIC
```

This command will attempt to GO BASIC,2 and would be the same command as follows:

```
1>GO BASIC,2  or
2>GO BASIC
```

2.7 CReate a file

CR <FILE NAME> [,<DRIVE#>] <FILESIZE> [,<DISKADDR>] [<DENSITY>]

The CReate command (CR) is used to make an entry into the directory of a disk for a new file and then assign space on the disk to hold the file. The file name must be a legal file name. If the optional drive number is not specified, MicroDoZ will use the default drive number. The file size specification is in 256 byte blocks.

The optional disk address can be used to cause the file to be placed at the specified disk address on the disk or if this address is not specified, MicroDoZ will "put" the file immediately after the last file on the disk. The optional density is either a "D" or "S" to signify Double or Single density. If the optional density is not specified, the file will be created double density.

When a file is CReated, it assumes the attribute of the last attribute (AF) command issued. The default is an attribute of Ø and the attribute must be Ø to be North Star compatible. The attributes are stored by using the seventh (high order) bit of each character of the file name.

2.8 TYpe a file

TY <FILE NAME> [,<DRIVE#>]<FILE TYPE> [<HEXADDR>]

The type command (TY) is used to set or change the type number of the specified file. The file name must be a legal file on the specified drive. If the optional drive argument is not passed, the default drive is used. The file type can be any number from Ø to 64 but if the type is 1, the optional GO address must be specified. The GO address (HEXADDR) is the RAM address where the file is to be loaded and run.

2.9 DElete a file

DE <FILE NAME> [,<DRIVE#>]

The delete command (DE) is used to delete an entry from the directory of the specified disk. The file name must be a file on the disk. If the optional drive is not specified, the default drive is used. This command takes no action on the file itself but merely deletes the directory listing of the file.

The DElete command will not work on a file that is set as a Read Only or System file.

2.1Ø Output Device or Input Device

OD or ID <DEVICE#>

The print device default commands are used to set the Output (OD) or Input (ID) Device to the specified device number. The device number must be in the range of Ø to 7. If no print device is specified to the MicroDoZ print routines, the default device is used.

The default print device can be used in a situation where all printing should be output to the printer, or if a programmer needs the output device changed for an entire program but doesn't want to change the program itself.

2.11 CLear the screen

CL

The CLear command is used to clear the screen of the CRT. The screen is cleared automatically by MicroDoZ when it is initially booted. Any other time the programmer desires the screen clear the CL command should be executed.

2.12 Define Drive

DD <DRIVE#>

The Default Drive command (DD) is used to set the default drive number. The default drive number is the number that will appear in the MicroDoZ prompt. The range of acceptable drive numbers is from 1 to 7. As an example, if the default drive is currently 1, and the default command is given to change the default to drive 2 the sequence would appear as follows:

```
1>DD 2
2>
```

Drive numbers other than the default can still be accessed by all commands if the optional drive number argument is used.

2.13 REname a file

RE <NEW FILENAME> <OLD FILENAME> [,<DRIVE#>]

The REname command (RE) is used to change the name of a file that already exits on the specified drive to a new name. The new file name must be a legal file name. If the optional drive number argument is not given, the default drive is used. If the optional drive number argument is given, the file will be renamed on the same drive as specified.

2.14 Read Only or Write Enable

RO or WE <FILENAME> [,<DRIVE#>]

The Read Only (RO) and Write Enable (WE) commands are used to set a file to a read only status or to reset a read only file so that it may be written to. Any attempts to write to a file that is marked as read only will result in an error being returned from MicroDoZ to the program calling for the write.

Files set as Read Only can not be REnamed. To REname a Read Only file, it must first be Write Enabled. For the files to remain North Star compatible, all files on a disk must be Write Enabled (WE).

2.15 Set System or Non System

SS or NS <FILENAME> [,<DRIVE#>]

The Set System file (SS) or reset to Non System file (NS) commands are used to specify system files. A system file is one that cannot be read or written to, cannot be Load Filed (LF) or Save Filed (SF), cannot be TYped, and cannot be DEleted or RENamed.

The only operations that can be performed on a system file is to GO to the file. For the files to remain North Star compatible, all files on a disk must be a Non System file (NS).

2.16 Attribute Field

AF <ATTRIBUTE#>

If the Attribute is set to Ø, all files on the directory are listed when the user executes a LI command. If the attribute number is set to any other value (1 to 63), only those files with the proper attribute or the attribute of Ø will be shown when a directory listing (LI) is called for. This command is to be used in conjunction with timesharing and user lockouts.

The attributes of a file are set at the time the file is CReated or REnamed. To set the Attribute Field for a file, first issue the AF command to set the attribute you want. Now CReate the file or REname it and the file will have the attribute you have selected. For a file to be North Star compatible, the Attribute Field must be set to Ø.

The attributes of a file (including its write status and system status) are stored using the eighth bit of each letter in the file name. The first letter controls the read only status, the second letter controls the system status, and the remaining letters form the 64 attribute combinations that a file can have.

MicroDoZ allows multiple files with the same name but different Attributes. Any attempts to access files with the same name will result in MicroDoZ accessing the first file with the name. It is recommended that files not be used which have the same name but different attributes.

If the Attribute field is to be used for file lockout, a simple procedure is followed. When a user wants to access a file and lock that file so other users cannot access it, use the ATtribute command to set an Attribute which is not common to any other user (such as the user's bank number). The file in question should now be REnamed to the same (or different) name. This procedure will cause the attribute of the file to be changed to the selected attribute.

When the user is finished with the file the procedure is similar. First set the ATtribute to Ø (so all users have access to the file) and REname the file so that it's attribute is restored.

During the time the file attribute is changed, no other user will be able to access the file. The reason for this is all other users will not "see" the file in the directory unless they have the same attribute as the user who set the attribute. If the user number is used for the attribute, no other user will have the same attribute and the file will be locked so only one user can access the file at any one moment.

MACHINE LANGUAGE INTERFACE

MicroDoZ is interfaced to other machine language programs by a jump table located in low memory. At location ØØØØ Hex is a jump that points to the entry point of MicroDoZ. This is the "warm boot" location when a program wants to return to MicroDoz. The low address of MicroDoZ can be found by loading the two bytes starting at address ØØØ1 Hex (LHLD 1).

Following this jump is the I/O byte at location ØØØ3 Hex. The I/O byte is divided into two nybbles. The high order nybble contains the current default input device and the low order nybble contains the default output device.

Location ØØØ4 Hex is a byte which contains the default drive number.

At location ØØØ5 Hex is a jump to the MicroDoZ interface command location. By putting a command number in the C register and calling location 5, any MicroDoZ command may be executed by another program.

At ØØ8Ø Hex is a command buffer which is 127 bytes long. This buffer may be used to pass commands between MicroDoZ and any other program such as **baZic.**

The actual location of MicroDoZ will vary depending on the hardware configuration. MicroDoZ is located at the top (high) end of memory. This leaves all memory from Ø1ØØ Hex to the bottom of MicroDoZ for other programs to execute.

A summary of the ØØØØ Hex section is as follows:

```
        ORG ØØØØH
ØØØØH   JMP  MicroDoZ
ØØØ3H   DB   I/O BYTE
ØØØ4H   DB   DEFAULT DRIVE
ØØØ5H   JMP  MDOZCOMMAND
        ORG ØØ8ØH
DOSBUFFER DS 127  *MicroDoZ INPUT AND TRANSIENT COMMAND
                    BUFFER
```

In the following sections that detail the use of the MDOZCOMMAND routine, the number of the command is displayed in the section title for each command detailed.

All commands use a stack that is local to MicroDoZ. Upon RETURNing from the command the stack is restored to the stack before the command call.

Remember that the C register contains the command number when communicating with MicroDoZ. The routine should call 0005 Hex to execute the command.

3.1 I/O Commands (0-6)

The I/O commands 0-6 should have the A register loaded with the device number. These I/O commands differ from commands 7-13 in that commands 7-13 use the I/O byte to determine to which I/O device they are to communicate.

3.1.1 INPUT (0) and OUTPUT (1)

All input CALLs return the byte input in the A register. The byte has had the eighth bit stripped by an ANI 7FH instruction. All ouputs use the B register to pass the byte that is to be output. When the routine returns, the B register will be equal to the A register.

To call these routines, the C register should be equal to a "0" for input and a "1" for output. A sample use of commands 0 and 1 to get a character and display the character is shown in the following example:

C=0 INPUT
C=1 OUPUT

```
LDA INPUTDEV    *Load A with the input device number
MVI C,0         *Load C with the command number
CALL 5          *Call MicroDoZ to input a byte
MOV B,A         *Move byte input to C
LDA OUTPUTDEV   *Load A with the output device number
MVI C,1         *Load C with the command number
CALL 5          *Call MicroDoZ to output the byte
```

3.1.2 CONTROL C (2)

This command is executed to determine if a control C has been entered by the user. When this command is called, if the carry is not set (i.e. equals 0) then no input was detected. If an input was detected, the carry will be set (equals 1).

On the condition where a character has been input, if the Z flag is not set (equals 0), the character input was NOT a control C. If the carry is set and the Z flag is set (equals 1) then the character input WAS a control C.

An example of the use of the control C command is as follows:

C=2 CONTROL C

```
MVI C,2          *Load C with the command number
LDA INPUTDEV     *Load A with the input device number
CALL 5           *Call MicroDoZ to check for control C
RNC              *Return if control C not detected
JZ CNTRLC        *Control C detected, branch to routine
```

3.1.3 INSTATUS (3)

The INSTATUS command is used to determine the status of an input device (i.e. is the device ready to transmit another character or not). If the routine returns with the Z flag NOT set (Z=Ø), the device is NOT ready. If the Z flag is true (Z=1), the device IS ready for the next character. If the device is not a legally implemented device, the A register will contain ØFF Hex.

An example of the use of this command appears as follows:

C=3 INSTATUS

```
MVI C,3          *Load C with the command number
LDA INPUTDEV     *Load A with the device number
CALL 5           *Call MicroDoZ to check status
JZ ROUTINE       *Character is ready to be input
```

3.1.4 OUTSTATUS (4)

The OUTSTATUS routine is the same as the INSTATUS except the routine checks the status of the output device to determine if it is ready or not. If the routine returns with the Z flag NOT set (Z=Ø) then the device is NOT ready. If the Z flag is true (Z=1) then the device IS ready for the next character. If the device is not a legally implemented device the A register will contain ØFF Hex.

An example of the use of this command appears as follows:

C=4 OUTSTATUS

```
MVI C,4          *Load C with the command number
LDA OUTPUTDEV    *Load A with the output device number
CALL 5           *Call MicroDoZ to check status
JZ ROUTINE       *Device is ready for a character
```

3.1.5 CLEARSCREEN (5)

The CLEARSCREEN command is used to clear the screen on the selected device number. An example of the use of this command appears as follows:

C=5 CLEARSCREEN

```
MVI C,5              *Load C with command number
LDA OUTPUTDEVICE     *Load A with output device number
CALL 5               *Call MicroDoZ to clear the screen
```

3.1.6 GOTOXY (6)

This command is used to address the cursor to the X and Y coordinates of the device passed in the A register. The H register contains the line and the L register contains the position on the line of where the cursor is to be addressed. These routines are defined in the I/O section of MicroDoZ and are specific to each CRT. An example of the use of this command is as follows:

C=6 GOTOXY

```
MVI H,XPOSITION     *Load H with row
MVI L,YPOSITION     *Load L with column
MVI C,6             *Load C with the command number
LDA OUTPUTDEVICE    *Load A with the output device #
CALL 5              *Call MicroDoZ to position cursor
```

3.2 I/O Commands (7-13)

These commands are identical to the preceding commands except there is no need to pass a device number in the A register because these commands use the appropriate nybble at location ØØØ3 Hex to determine the I/O device number. A summary of the commands is listed with a sample input and output command call:

C=7 INPUT
C=8 OUTPUT

```
MVI C,7    *Load C with the command number
CALL 5     *Call MicroDoZ to do input
MOV B,A    *Move the byte input from A to B
MVI C,8    *Load C with the command number
CALL 5     *Call MicroDoZ to output the character
```

C=9 CONTROLC
C=1Ø INSTATUS
C=11 OUTSTATUS
C=12 CLEARSCREEN
C=13 GOTOXY

3.3 MicroDoZ Non Disk Routines (14-23)

The following routines provide the programmer access to several routines that are an integral part of MicroDoZ. Access is provided to these routines so the programmer will not have to duplicate efforts if the routines are needed by another program. All of the routines in this section use the default device (defined by the I/O byte at ØØØ3 Hex) for output. All routines that output numbers, print a trailing space after the number.

3.3.1 CRLF (14)

This routine outputs a carriage return and a line feed to the default output device.

C=14 CRLF (CARRIAGE RETURN LINEFEED)

3.3.2 PRINT MESSAGE (15)

This command causes a message to be output to the default output device. The H and L register pair should contain the address of the beginning of the message to be printed. The string should end with a ØØ Hex to inform the routine that the message end has been reached. A command summary and example follow:

C=15 PRINT MESSAGE POINTED TO BY HL ENDING WITH ØØH

```
MESSAGE ASC HELLO    *Define message "HELLO"
        DB Ø         *End message with Ø
LXI H,MESSAGE        *Load HL with message address
MVI C,15             *Load C with command number
CALL 5               *Call MicroDoZ to output message
JMP CONTINUE         *Continue with program
```

3.3.3 PRINT 16 BIT HEX NUMBER (16)

This command is used to print a Hex number that is represented by a Binary number in the HL register pair. The number is printed as a four digit Hex number. No error is returned from this routine. A summary and example follow:

C=16 PRINT 16 BIT HEX NUMBER IN BINARY HL

```
LHLD NUMBER     *Load HL with number
MVI C,16        *Load C with command number
CALL 5          *Call MicroDoZ to print number
```

3.3.4 PRINT 8 BIT HEX NUMBER (17)

This command prints an 8 bit Hex number that is contained in the HL register pair as a Binary number. This routine assumes the H register is set to Ø so that a two digit Hex number is printed. This routine returns no errors. Leading zeros are not printed. A summary and example follow:

C=17 PRINT 8 BIT HEX NUMBER IN BINARY HL

```
MVI H,Ø         *Clear register H
MVI L,NUMBER    *Load number into L
MVI C,17        *Load C with command number
CALL 5          *Call MicroDoZ to print number
```

3.3.5 PRINT 16 BIT DECIMAL NUMBER (18)

This command prints the Binary number contained in the HL register pair as a Decimal number. The command is summarized as follows:

C=18 PRINT 16 BIT DECIMAL NUMBER BINARY IN HL

3.3.6 PRINT 8 BIT DECIMAL NUMBER (19)

This command prints the Binary number contained in the HL pair as a Decimal number. The command is summarized as follows:

C=19 PRINT 8 BIT DECIMAL NUMBER BINARY IN HL

3.3.7 CONVERT DECIMAL STRING TO BINARY (2Ø)

This command is used to convert a string that is pointed to by the H and L register pair to a Binary number. The results will be placed in the HL register pair when the operation is complete. A summary and example are as follows:

C=2Ø CONVERT DECIMAL STRING POINTED TO BY HL TO BINARY IN HL

```
STRING DB ' ',' ',' '    *Leading spaces allowed
       DB '1','2','3'    *String of numbers
       LXI H,STRING      *Load HL with string address
       MVI C,2Ø          *Load C with command number
       CALL 5            *Call MicroDoZ to convert
       JC ERROR         *Saw a separator character
                         *before a valid digit or
                         *overflow error occured
```

3.3.8 CONVERT HEX STRING TO BINARY (21)

This command is the same as command 2Ø except a Hex string is operated on by the routine rather than a Decimal string. The command summary follows:

C=21 CONVERT HEX STRING POINTED TO BY HL TO BINARY IN HL

3.3.9 OUTPUT SPACES (22)

This command is used to ouput the desired number of spaces to the default output device. The A register contains the number of spaces to be output. The command summary and example follow:

C=22 OUTPUT A REGISTER NUMBER OF SPACES

```
MVI A,1Ø  *Load A with number of spaces to output
MVI C,22  *Load C with command number
CALL 5    *Call MicroDoZ to output spaces
```

3.3.1Ø EXECUTE MicroDoZ COMMAND (23)

This is a very powerful command because it lets any calling program execute all the MicroDoZ commands. The general procedure is to define a command or series of commands somewhere in RAM, set the HL register pair to "point" to the address of the command(s) and call MicroDoZ with command 23 in the C register. Each command must end in one of the following separator characters: "ØDH" (carriage return), ":", or "\".

If a disk error occurs during the use of this command the carry will be set to 1 and the A register will contain the number of the error condition. The error numbers are defined in Section 3.4 (MicroDoZ Disk Routines).

The command summary and an example using **baZic** is as follows:

```
C=23 DOS COMMAND EXECUTE DOS COMMAND POINTED TO BY HL
     COMMAND STRING MUST END WITH A CR,:, OR \

     STRING ASC GO BAZIC,2\CHAIN CONTROL,2   *Define string
            DB ØDH                 *Define carriage return
            LXI H,STRING           *Load HL with string address
            MVI C,23               *Load C with command number
            CALL 5                 *Call MicroDoZ to execute
            JC ERRORCK             *Error detected
            JMP CMDOK              *Command was executed OK
     ERRORCK ORA A                 *Set flags for condition of A
            JZ FILEERROR           *If Z flag set then file error
            CPI 8                  *Is it less than 8?
                                   *This could be a CPI 7 to
                                   *detect a write protect error
            JNC FILEERROR          *File error
            JMP HARDDISKERROR      *Hard disk error
```

3.4 MicroDoZ Disk Routines (24-29 and 33)

The following routines use the I/O byte at ØØØ3 Hex for their device numbers. All disk accessing can and should use these commands. If all I/O and disk accessing goes through the proper commands, the programmer can expect his/her programs to work without modification under timesharing conditions using JOESHARE or HDSHARE.

On RETURNing from the following disk routines, if the carry bit is equal to Ø, the command was executed properly and no problems were detected by the routines. If the carry bit is set to 1, the error message code will be contained in the A register and can be determined by the following table:

CARRY=1 ON ERROR CONDITION
AND A REGISTER EQUALS

Ø= FILE ERROR EXCEPT DLOOK AND SAVE
1= SYNC ERROR
2= CRC ERROR
3= VERIFY ERROR
4= INDEX ERROR
5= DENSITY MISMATCH ERROR
6= DISK HAS WRITE PROTECT TAB
7= WRITE PROTECT BIT SET IN DIRECTORY
8= FILE ERROR
9= NO ROOM IN DIRECTORY

3.4.1 SELECT DRIVE AND DENSITY (24)

This command is used to select the correct drive and density before every DCOM call. The routine is called with the appropriate value in the A register. This value should be defined such that:

BIT 7 indicates the DENSITY (Ø=SINGLE 1=DOUBLE)
BIT 6 is set to Ø
BITS Ø..2 equal the DRIVE NUMBER

BIT 6 is reserved for the disk side information if your system has double sided drives. This bit is reserved for this purpose and should be set to zero (Ø) by software calling this routine. MicroDoZ will itself determine if the file to be accessed is on the front or back side of the disk and set this bit accordingly.

This routine does no error checking and returns no error messages.

3.4.2 DCOM (25)

This routine is similar to the North Star DCOM routine. DCOM is used to read or write a file depending upon the command that is issued. The MicroDoZ DCOM is not exactly equivalent to DOS and cannot be called in exactly the same manner as North Star DCOM. A program (DOS2ØØØ) is provided to appear the same as North Star DOS so that programs will not have to be changed to run under MicroDoZ.

To use command 25, **first call command 24** to set the proper disk drive and density. The command is called with the following register conditions:

C=25 DCOM

HL= DISKADDRESS
DE= RAM ADDRESS
A= NUMBER OF SECTORS (hardware dependent)
B= COMMAND Ø=WRITE 1=READ

Upon RETURNing from the routine the carry bit will be Ø if no errors were detected and the HL register pair will point to the next RAM address. A sample call of this routine would appear as follows:

```
LDA DISK            *Load A with the drive number
MOV C,A             *Save in C
LDA DENSITYSIDE     *Load A with the density and
                    *side information
ORA C               *Combine with C
MVI C,24            *Load C with command number
CALL 5              *Call MicroDoZ to set drive
LHLD RAMADDRESS     *Load HL with RAM address
XCHG                *Save in DE
LHLD DISKADDRESS    *Load HL with disk address
LDA COMMAND         *Load A with command 1=read
                    *Ø=write
MOV B,A             *Save in B
MVI C,25            *Load C with command number
LDA SECTORS         *Load A with number of sectors
CALL 5              *Call MicroDoZ to execute
RNC                 *Command was executed OK
ORA A               *Set flags for condition of A
JZ FILEERROR        *Jump on file error
CPI 8               *Compare for less than 8
JNC FILEERROR       *No so file error
JMP HDERR           *Hard disk error
```

3.4.3 DLOOK (26)

This command is normally used to look up an existing entry in the directory of a disk. This routine can also be used to "look up" an empty directory space when creating a file, but the normal procedure would be to use the CREATE command (33) to create new files. DLOOK retains the capability to look up empty directory spaces in order to remain compatible with North Star DOS.

If DLOOK is being used to look up an existing entry in the directory and the correct file name is found, the carry will be cleared to Ø and HL points to the beginning of the directory entry when the routine returns. The eighth bit of the A register will contain the density of the directory (1=double, Ø=single).

If the directory entry requested is not found, DLOOK will return with the carry set to 1, and the first 7 bits of the A register will be cleared to zero (Ø). If an error occurred during this routine, the A register will contain the number of the error as defined in Section 3.4. In all cases, the eighth bit of the A register will contain the density of the directory (1=double, Ø=single).

If DLOOK is being called and the specified file name does not exist in the directory, the carry will be set to 1, the first seven bits (Ø-6) of the A register will be cleared to Ø, HL will point to a free directory space, DE will contain the free disk address, and BC will be equal to the maximum negative disksize (space remaining on the disk) in sectors plus 1. If the carry equals 1 and the first seven bits (Ø-6) of the A register are NOT equal to Ø then an error occurred.

If the file type is 3 to 64 or Ø, bytes 14, 15, and 16 are not used.

C=26 DLOOK

```
HL POINTS TO <FILENAME>,<DRIVE>CR
ON RETURN
CARRY=Ø HL POINTS TO 1ST BYTE OF DIRECTORY ENTRY
CARRY=1 A=Ø (BITS Ø-6)
     HL POINTS TO FREE DIRECTORY SPACE
     DE= FREE DISK ADDRESS
     BC= -(DISKSIZE IN BLOCKS+1)
CARRY=1 A<>Ø DISK ERROR A=9 THEN DIRECTORY FULL
FNAME     ASC BASIC,3     *Define file name and drive
          DB ØDH          *End with carriage return
DLOOK     LXI H,FNAME     *Load HL with address of name
          MVI C,26        *Load C with command number
          CALL 5          *Call MicroDoZ to execute
          PUSH PSW        *Save A and flags
          ANI 8ØH         *Strip density bit
          STA DIRECTORY   *Store directory density
          POP PSW         *Recover A and flags
          JNC FOUND       *Carry not set so file found
                          *HL points to 1st byte of
                          *directory buffer
          ANI 7FH         *Strip 7th bit
          ORA A           *Set flags for A
          JZ NOTFOUND  *Empty space is available
                          *in directory
          CPI 9           *Compare with 9
          JZ DIRECTORY    *Directory is full
          CPI 8           *Compare with 8
          JNC FILEERROR   *Jump if file error
          JMP HDERR       *Hard disk error
```

3.4.4 DWRITE (27)

This command writes an updated directory entry. This routine is the equivalent of the North Star DWRITE routine. A command summary and example of the command follows:

C=27 DWRITE WRITE DIRECTORY BUFFER TO DISK
(NOTE FILE ATTRIBUTES MUST BE SET BY USER)

```
MVI C,27        *Load C with command number
CALL 5          *Call MicroDoZ to execute
RNC             *Carry not set so no error
ORA A           *Set flags
JZ FILEERROR    *Jump on file error
CPI 8           *Compare with 8
JNC FILEERROR   *Jump on file error
JMP HDERR       *Hard disk error
```

3.4.5 SAVE (28)

This command was designed to work with only a type 2 file (a **baZic** program) and implemented so that **baZic** would not have to duplicate routines already in MicroDoZ. The command is called with the HL register pair pointing at the file name and drive number. The DE register pair contain the address in RAM of the program and B contains the number of 256 byte blocks to save.

Upon RETURN from the routine if the carry is clear (carry=Ø), the operation was successful. If the carry is set (carry=1), an error condition was encountered. If the A register is Ø, the program was too large to fit in the specified file and if the A register is NOT equal to Ø, a file or disk error occurred.

The command summary and an example follow:

C=28 SAVE (SAVE A **baZic** PROGRAM UPDATING PGMSIZE BYTE OF DIRECTORY.

HL POINTS TO <FILE NAME>,<DRIVE>CR
DE = RAM ADDRESS
B= NUMBER OF 256 BYTE BLOCKS TO SAVE

ON RETURN
CARRY=Ø SAVE OK (IGNORE A REGISTER)
CARRY=1 A=Ø PROGRAM TOO LARGE
CARRY=1 A<>Ø FILE OR DISK ERROR

```
SAVE    LHLD RAM        *Load HL with RAM address
        XCHG            *Save in DE
        LXI H,FNAME     *Load HL with address of name
        LDA BLOCKS      *Load A with number of blocks
        MOV B,A         *Save in B
        MVI C,28        *Load C with command number
        CALL 5          *Call MicroDoZ to execute
        RNC             *Carry not set so save OK
        ORA A           *Set flags
        JZ PROGRAM      *Program was too large
        CPI 8           *Compare with 8
        JNC FILEERROR   *Jump on file error
        JMP HDERR       *Hard disk error
```

3.4.6 LOAD (29)

This command LOADs a **baZic** program from the disk into internal memory. The command is passed with the HL pair pointing to the file name and drive number. The DE pair point to the RAM address of where the program is to be loaded in RAM. This command works on type 2 files only.

Upon RETURNing from the routine, if the carry is clear (carry=Ø), the load was executed without problems. If the carry is set (carry=1), a file or disk error occurred.

The command summary and example follow:

```
C=29 LOAD (LOAD A BASIC PROGRAM USING PGMSIZE BYTE FOR SPEED

     HL POINTS TO <FILE NAME>,<DRIVE>CR
     DE= RAM ADDRESS

ON RETURN

     CARRY=Ø LOAD OK HL=NEXT DMA ADDRESS
     CARRY=1 FILE OR DISK ERROR

     LOAD LHLD RAM        *Load HL with RAM address
          XCHG            *Save in DE
          LXI H,FNAME     *Load HL with address of file name
          MVI C,29        *Load C with command number
          CALL 5          *Call MicroDoZ to execute
          RNC             *Carry not set so load was OK
                          *HL=points to next RAM address
          ORA A           *Set flags
          JZ FILEERROR    *Jump on file error
          CPI 8           *Compare with 8
          JNC FILEERROR   *Jump on file error
          JMP HDERR       *Hard disk error
```

3.4.7 CREATE (33)

This command is used to create a file. The command is called with the HL register pair pointing to the file name and drive number, followed by an ØD Hex (carriage return). The file name and drive must be separated by a comma. The DE register pair should be the size of the file in blocks while the B register should be the file type. The file is always created the same density as the directory.

The command summary follows:

```
C=33 CREATE A FILE

     HL POINTS TO <FILE NAME>,<DRIVE>CR
     DE= SIZE OF FILE IN BLOCKS
     B= FILE TYPE
```

ON RETURN

CARRY=1 ON ERROR CONDITION
AND A REGISTER EQUALS

```
Ø= FILE ERROR EXCEPT DLOOK AND SAVE
1= SYNC ERROR
2= CRC ERROR
3= VERIFY ERROR
4= INDEX ERROR
5= DENSITY MISMATCH ERROR
6= DISK HAS WRITE PROTECT TAB
7= WRITE PROTECT BIT SET IN DIRECTORY
8= FILE ERROR
9= NO ROOM IN DIRECTORY
```

3.5 Miscellaneous Routines (3Ø-32)

This section describes the miscellaneous commands.

3.5.1 NEXT SEPARATOR BYTE (3Ø)

No arguments are passed to this command. Upon RETURNing from the routine, the HL register pair point to the next separator byte in the MicroDoZ I/O buffer. The separators "found" by this command are the carriage return (ØDH),"\", and ":". Commands 25, 26, 27, and 28 should call Command 3Ø before they are executed to insure that the commands are always pointing to a valid separator byte.

Command 3Ø (NEXT SEPARATOR BYTE) can be called at anytime to set the pointers so that an assembly language program can have accces to the Common Command Buffer. **baZic** uses a typical call of this command to determine what **baZic** program is to be loaded and executed when **baZic** is booted.

The command summary and example call of the command follow:

C=3Ø RETURN HL POINTS TO NEXT SEPARATOR BYTE IN DOS IO BUFFER

SEPARATORS ARE ØDH,\,:

```
MVI C,3Ø            *Load C with command number
CALL 5              *Call MicroDoZ to execute
SHLD DOSPOINTER     *Save the pointer
MOV A,M             *Load A with byte
CPI ØDH             *Is it a carriage return?
JZ                  *Yes so no command follows
```

3.5.2 MicroDoZ BOUNDARIES (31)

This command is passed no argument. Upon RETURNing from the call, the HL pair contain the starting address of MicroDoZ and the DE pair contain the ending address of MicroDoZ. If the A register is equal to zero, there is not a hard disk "on line". If the A register is not equal to zero, a hard disk drive is "on line." The command summary and example follow:

C=31 ON RETURN

HL= START ADDRESS OF MicroDoZ
DE= ENDING ADDRESS OF MicroDoZ

3.5.3 PRINTER RELEASE (32)

This command is reserved for use in timesharing systems. In a non-timesharing environment, a call to this command will execute a RETURN instruction only. This command is used to release devices (usually printers) that are locked out by one user of the system. As an example, **baZic** calls this routine upon executing a CHAIN, READY, OR APPEND to release the printer so other users can print. The command summary follows:

C=32 PRINTER RELEASE (RESERVED FOR TIMESHARE)

3.6 Invalid Commands (34+)

All commands from number 33 up are invalid commands and have no meaning at the present time although commands 34 to 37 have been defined but not fully implemented. If the MDOZCOMMAND location is called with an invalid command, a file error will be returned. The command summary is as follows:

C>37 RETURN FILE ERROR
INVAILD COMMAND

The reserved command numbers and there function are as follows:

C=34 Open a file.
C=35 Close a file.
C=36 Lock a file.
C=37 Unlock a file.

DIRECTORY FORMAT

The first eight bytes in the directory is the file name. The seventh bit (high bit) of each letter is used to determine the Read Only (R/O) status, the System status, and the attributes of the file. To remain North Star compatible, each file must not have a Read Only, System, or Attribute bit set.

Bytes 9 through 16 of the directory carry the disk address, the size of the file in sectors, the type of the file and other information that is type dependent. Bytes 9 and 10 contain the disk address of the file with byte 9 containing the low order address and byte 10 containing the high order disk address.

Bytes 11 and 12 contain the size of the file in sectors. Byte 11 contains the least significant byte while byte 12 contains the most significant byte of the file size.

Byte 13 holds the density, the side (if applicable), and the type of the file. Bit seven sets the density of the file (1=double, 0=single), bit six sets the side (0=one, 1=two), and bits five to zero contain the 64 file types available.

Bytes 14, 15, and 16 contain type dependent information. If the file is a type 1 file, bytes 14 and 15 contain the low order and high order GO address of the file. Byte 16 is unused.

If the file is a type 2 (**baZic** program), byte 14 contains the number of 256 byte blocks the program currently requires. This byte is used to speed the LOADing or SAVEing of the file so that if the file is actually much larger than the program, the entire file is not loaded or saved. Only the sectors containing the program are affected. Bytes 15 and 16 are not used if the file is a type 2 file.

The LI command under MicroDoZ results in the following information being given for each file listed in the directory of the affected disk:

File Name (maximum of eight characters)
Starting Disk Address (Decimal)
Length of the File in Blocks (Decimal)
Density (Single or Double)
Type of File (0 to 64 Decimal)
GO Address (If Type 1 in Hex)
R/O if Read Only File
SYSTEM if System File
Attribute Field (0 to 63 Decimal)

A sample LIsting of the directory under MicroDoZ would appear as follows:

```
1>LI

MICRODOZ      4    20 D   Ø                    AF  Ø
M2DØØM       14    1Ø D   1 2DØØ               AF  Ø
TEST         19     2 D   2       R/O SYSTEM  AF  Ø
BAZIC        2Ø    54 D   1 Ø1ØØ               AF  Ø
1>
```

MICRODOZ STRUCTURE

MicroDoZ is divided into four main sections. The first section is the command processor section. This section is the lowest (has the smallest address) of the sections and is always at the beginning of MicroDoZ. This section is responsible for interpreting input from the keyboard and if a command has been typed to execute the appropriate command. The command section is 3K in length.

Directly following the command processor is the I/O section. All user written code in this section must be relocatable. This section is divided into two parts, each part having 256 bytes. The I/O section therefor is 1/2 K (512 bytes) in length. The first half of this section is the configure block. This block contains user written code that defines the following parameters:

CRT LINES PER PAGE
VERIFY FLAG
CONFIGURATION BYTE (QUAD DRIVES)
DISPLAY SYSTEM FILES FLAG
CLEAR SCREEN CODES
CURSOR ADDRESSING CODES
CURSOR ADDRESSING OFFSETS
TURNKEY FLAG
TURNKEY PROGRAM(S)
BACKSPACE SEQUENCE

The second block of the I/O section defines the I/O routines for MicroDoZ to communicate with different hardware. The following routines must be defined by the user in this section if the user does not have a standard Horizon setup:

INPUT ROUTINES
OUTPUT ROUTINES
PANIC (CONTROL C) ROUTINES
INITIALIZATION ROUTINES
INPUT PORT STATUS ROUTINES
OUTPUT PORT STATUS ROUTINES
PRINTER RELEASE ROUTINE (TIMESHARING ONLY)

The next section is the MicroDoZ buffer region. The buffer region is 1/2 K (512 bytes) and is used to buffer MicroDoZ disk accesses such as disk initialization.

The last section of MicroDoZ is the disk primitive section. This section contains the routines that are specific to a particular disk drive and controller. The bootup sequence is also defined here since the bootup sequence can be very different from one system to another. In addition, the file lock out routines and the hard disk patches are located in this section but are only used in timesharing or hard disk computers.

At boot time, the boot routines are located in the sector immediately preceding the command processor section. These routines are needed only for the boot and the space is available for use by other programs as soon as the boot process is completed.

At bootup, MicroDoZ loads itself into the high memory of the computer. The control bytes in low memory are written and MicroDoZ begins operation. In a 48K system, MicroDoZ begins at AAØØH and ends at BFFFH, occupying about 5 1/2 K.

The image of MicroDoZ on the disk is different from the image that operates in RAM. On the disk, the primitive section is first, followed by the command processor section, followed by the I/O section. As MicroDoZ boots, each section is loaded into its proper place.

Since the disk image is not the same as the working image, the working image can never be saved back to the disk by the Save File (SF) command. MicroDoZ may by "Load Filed" (LF), modified, and then saved back on the disk using the Save File (SF) command. Since the I/O section is exactly 512 bytes long, it can be overlayed in the file by executing a Write Disk command with the appropriate arguments.

5.1 Configure Block

The configure block is code that is specific to the users hardware. The Configure Block Source should be consulted while reading this section to clarify the position of the various flags and routines in this block.

The first byte of this block is the **PAGE** byte. This byte determines the number of lines to display upon the default I/O device (console device) before the PRESS RETURN TO CONTINUE message is output to the display of the device on a LIst of the disk directory. The default value for this byte is 24 (lines per page).

The second byte of this block is the **VERIFY** flag byte. If this byte is set to zero (Ø), MicroDoZ will not verify writes to disk files. If the flag is set to one (1), all file writes will be verified. The default for the verify flag is one (verify enabled).

The third byte of this block is the **CONFIG**ure byte. This byte is set to "tell" MicroDoZ if any quad density five inch disk drives are "on line". The high nybble of this byte determines if a quad capacity is present while the low nybble determines which drives have "fast stepping" capability. The bits from 7 to 4 control drives 1 to 4 respectively with a bit being 1 indicating that a quad drive is on line. Bits Ø to 3 control the fast stepping capability of drives 1 to 4 respectively with a bit value of 1 representing fast stepping. The default value is zero (Ø).

The fourth byte of this block is the **D**isplay **SYS**tem (DSYS) flag byte. If this byte is a non zero value, all files classified as SYSTEM files will not display when the directory is LIsted. The default value is zero (Ø).

The following two locations are both structured in a similar manner. These two locations control the clear screen and cursor addressing sequence for your CRT (**CSCR** and **GOTO**). If the first byte of each of these locations is FFH, MicroDoZ will assume that a machine language jump follows which jumps to the location of a routine which accomplishes the clear screen and cursor addressing for your CRT.

Any other value will be interpreted by MicroDoZ to be the number of characters following which are to be output by the "built in" clear screen and cursor addressing routines of MicroDoZ. The default for both of these locations is two (2) indicating that 2 bytes are to be output to the CRT to accomplish the clear screen and cursor addressing prefix.

The next two bytes following the cursor addressing codes location are the row and column offset values (**XOFF** and **YOFF**). The default for both these locations is 31.

The next byte is the turnkey flag byte (**TKEY**). This byte determines if MicroDoZ is to continue booting other program(s) upon bootup, or if MicroDoZ is to terminate bootup in its own command mode. The default for this byte is zero (Ø) indicating MicroDoZ is not in the turnkey mode. If a turnkey command is in the buffer, the TKEY byte should be the number of characters in the command.

Immediately following the turnkey byte is the turnkey buffer. This buffer is 8Ø bytes long and is transferred upon bootup to the common command buffer located at 8ØH. Multiple commands may be "left" in the turnkey buffer as long as they are separated by a separator character (\).

After the turnkey buffer is the backspace (**BKSP**) string. This string is the sequence of bytes that are to be output when a backspace is detected. The default sequence is to output a backspace (ASCII 8) followed by a space (ASCII 32) followed by another backspace. The backspace "message" must end with a zero.

The last location defined in the configure block is the backspace recognition key (BKSPSET) codes. These are the codes which MicroDoZ is to recognize as a backspace. The default values are ASCII 8 (backspace), 95 (rub or delete), and 17 (control Q).

In summary, the tags are listed, as well as the Hex and Decimal address of where the byte or location may be found in relation to the ORIGin of the configure block of MicroDoZ. The ORIGin for this section in a 48K version of MicroDoZ would be B800H.

TAG NAME	HEX OFFSET	DECIMAL OFFSET	DEFAULT VALUE
PAGE	ORG + 0	ORG + 0	24
VERIFY	ORG + 1	ORG + 1	1
CONFIG	ORG + 2	ORG + 2	0
DSYS	ORG + 3	ORG + 3	0
CSCR	ORG + 4	ORG + 4	ESC + "E"
GOTO	ORG + 14	ORG + 20	ESC + "Y"
XOFF	ORG + 24	ORG + 36	31
YOFF	ORG + 25	ORG + 37	31
TKEY	ORG + 26	ORG + 38	13
(tkeybuffer)	ORG + 27	ORG + 39	BAZIC\CSUB
BKSP	ORG + 78	ORG + 120	
BKSPSET	ORG + 80	ORG + 128	

5.2 Configure Block Source

```
B800            10 *MDOZIO
B800            20 *
B800            30 *NOTE THE IO SECTION MUST BE RELOCATABLE CODE
B800            40 *TO REMAIN COMPATABLE WITH FUTURE PLANS
B800 18         50 PAGE DEFB 24 *LINES PER CRT PAGE
B801 01         60 VERIFY DEFB 1 *READ AFTER WRITE IF 1
B802 00         70 CONFIG DEFB 0 *CONFIGURATION BYTE FOR QUAD 5 INCH
                                 *DRIVES
B803 00         80 DSYS DEFB 0 *IF NON ZERO DON'T LIST SYSTEM FILES
B804            90 *CSCR AND GOTO 1ST BYTE IS STRING LENGTH
B804           100 *IF 1ST BYTE = 0FFH THEN MACHINE CODE MUST FOLLOW
B804 02        110 CSCR DEFB 2 *CLEAR SCREEN
B805 1B        120      DEFB 27
B806 45        130      DEFB "E"
B807 00        140      DEFB 0
B808           150      ORG CSCR+16
B814 02        160 GOTO DEFB 2 *POSITION CURSOR
B815 1B        170      DEFB 27
B816 59        180      DEFB "Y"
B817 00        190      DEFB 0
B818           200      ORG GOTO+16
B824           210 *XOFF AND YOFF OFFSET ADDED TO VALUE TO POSITION
                   *CURSOR
B824 1F        220 XOFF DEFB 31
B825 1F        230 YOFF DEFB 31
```

```
B826            240 *TKEY TURNKEY TO DOS HERE
B826            250 *1ST BYTE IS LENGTH OF STRING
B826            260 *MUST END WITH ØDH (CARRIAGE RETURN)
B826 ØØ         270 TKEY DEFB Ø
B827            280      DEFS 8Ø
B877 ØD         290      DEFB ØDH
B878            300 *BKSP THE STRING USED TO BACKSPACE ENDS WITH Ø
B878 Ø8         310 BKSP DEFB 8
B879 2Ø         320      DEFB 32
B87A Ø8         330      DEFB 8
B87B ØØ         340      DEFB Ø *BACKSPACE MSG ENDS WITH ZERO
B87C            350      DEFS 4

B88Ø            360 *BKSPSET PARSE KEYBOARD CHARACTERS USED TO
                    *BACKSPACE
B88Ø FE Ø8      370 BKSPSET CP 8
B882 C8         380         RET Z
B883 FE 5F      390         CP 95
B885 C8         400         RET Z
B886 FE 11      410         CP 11H
B888 C8         420         RET Z
B889 FE 7F      430         CP 127
B88B C9         440         RET
```

5.3 I/O Block

The I/O block begins at the next page boundary immediately following the Configure block. The two blocks together constitute one 512 byte sector. The I/O block contains all of the user written input and output routines as well as the control C detect, initialization routines, the input and output status routines and the printer release routine which are used in the timesharing versions. All of the following routines must be written in relocatable code.

At the beginning of this section is a jump table. The jumps are defined with the following tags in the source listing:

INP	(input routines)
OUT	(output routines)
PANIC	(control C detect)
TINT	(port initialization)
INSTAT	(input port status routines)
OUTSTAT	(output port status routines)
PRINTREL	(printer release code)

For the **INP**ut routines, the device number must be supplied in the A register. The value input from the port must have the seventh (high) bit stripped by an ANI 7FH instruction. Only the A register and the flags may be modified during this routine.

The **OUT**put routines should be defined such that the character to be output is in the B register and the device number is in the A register. The character output should be in the A register upon return. No registers may be modified except the A register and the flags.

The **PANIC** or control C detect routine is used to determine if the user has pressed a control C. The routine should first determine if a character has been typed. If a character has not been typed the routine should return with the A register clear and the carry flag not set.

If a character has been typed, the carry flag should be set and the routine should determine if the character was a control C. The character should be input, stripped of the seventh (high) bit, and compared with an ASCII 3. If the character is a control C, the Z flag should be set and the routine return. The A registers only can be modified during this routine.

The initialization routine (**TINT**) is normally used to initialize USARTs but can be used to initialize any device that needs this attention upon bootup. The initialization usually consists of outputting the correct values to specific ports to set the USARTs for proper working conditions. All registers may be used by this routine. If your system requires no initialization, simply insert a RET for this routine.

The **INSTAT** routines are used to determine if a particular input device has a character ready to send to the computer. The device number is passed in the A register. If the device passed is a nonexistent device, the routine should return with the carry set and a FFH in the A register. If the status is not ready, the routine should RETurn with the Z flag clear (Z=Ø). If the status is ready the should be set (Z=1).

The OUTSTAT routines are very similar to the INSTAT routines except the OUTSTAT routines check to see if an output device is ready to accept a character. The device number is passed in the A register. If the device passed is not defined, the routine should set the carry flag and return with a FFH in the A register. If the status is not ready, the routine should RETurn with the Z flag clear (Z=Ø). If the status is ready the should be set (Z=1).

The last routine to define is the printer release routine. This routine should be only a RET for a non timesharing system.

5.4 I/O Block Source

```
                        *
                        *JUMP TABLE AT BEGINNING OF I/O BLOCK
                        *
B889            440          ORG IOORG
B900 18 0C      450 INP JR INPUT1       *DEFINE THE JUMPS
B902 18 15      460 OUT JR OUTPUT
B904 18 30      470 PANIC JR PANICS
B906 18 3D      480 TINT JR TINTO
B908 18 6E      490 INSTAT JR INSTATS
B90A 18 57      500 OUTSTAT JR OUTSTATS
B90C 00         510 PRINTREL DEFB 0 *USED IN TIMESHARE SYSTEM
B90D C9         520          RET
                        *
                        *INPUT ROUTINE
                        *
B90E DB 03      530 INPUT1 IN A,(3) *INPUT THROUGH STATUS PORT
B910 E6 02      540        AND 2    *AND WITH MASK
B912 28 FA      550        JR Z,INPUT1 *LOOP UNTIL PORT READY
B914 DB 02      560        IN A,(2) *PORT READY SO GET CHARACTER
B916 E6 7F      570        AND 7FH *STRIP HIGH BIT
B918 C9         580        RET *END ROUTINE
                        *OUTPUT ROUTINES
                        *
B919 FE 01      590 OUTPUT CP 1 *COMPARE OUTPUT DEVICE NUMBER
B91B 28 0F      600        JR Z,PRINTER *PRINT #1 CASE
B91D FE 02      610        CP 2 *COMPARE WITH 2
B91F CC 2C B9   620        CALL Z,PRINTER *PRINT ON CRT & PRINTER
                        *
                        *CRT OUTPUT
                        *
B922 DB 03      630 CRT IN A,(3) *INPUT STATUS OF CRT PORT
B924 E6 01      640     AND 1 *AND WITH MASK
B926 28 FA      650     JR Z,CRT *LOOP UNTIL PORT READY
B928 78         660     LD A,B *READY SO PUT CHARACTER IN A
B929 D3 02      670     OUT (2),A *OUTPUT CHARACTER TO PORT
B92B C9         680     RET *RETURN UPON COMPLETION
                        *
                        *PRINTER OUTPUT
                        *
B92C DB 05      690 PRINTER IN A,(5) *INPUT STATUS OF PRINTER PORT
B92E E6 01      700         AND 1 *AND WITH MASK
B930 28 FA      710         JR Z,PRINTER *LOOP UNTIL PRINTER READY
B932 78         720         LD A,B *READY SO PUT CHARACTER IN A
B933 D3 04      730         OUT (4),A *OUTPUT THE CHARACTER
B935 C9         740         RET *RETURN UPON COMPLETION
```

```
                        *
                        *CONTROL C DETECT
                        *
B936 DB 03      750 PANICS IN A,(3) *INPUT THROUGH STATUS PORT
B938 E6 02      760        AND 2 *AND WITH MASK
B93A EE 02      770        XOR 2 *SET FLAGS
B93C C0         780        RET NZ *RETURN IF NO CHARACTER
B93D DB 02      790        IN A,(2) *CHARACTER READY SO INPUT IT
B93F E6 7F      800        AND 7FH *MASK HIGH BIT
B941 FE 03      810        CP 3 *IS IS A CONTROL C
B943 37         820        SCF *YES SO SET CARRY FLAG
B944 C9         830        RET *RETURN
                        *
                        *INITIALIZATION
                        *
B945 AF         840 TINTO XOR A *CLEAR A
B946 D3 06      850       OUT (6),A *SET UP PORTS
B948 D3 06      860       OUT (6),A *THESE EXAMPLES ARE FOR THE
B94A D3 06      870       OUT (6),A *STANDARD HORIZON PORTS
B94C D3 06      880       OUT (6),A
B94E 3E CE      890       LD A,0CEH
B950 D3 03      900       OUT (3),A
B952 D3 05      910       OUT (5),A
B954 3E 37      920       LD A,037H
B956 D3 03      930       OUT (3),A
B958 D3 05      940       OUT (5),A
B95A DB 02      950       IN A,(2)
B95C DB 04      960       IN A,(4)
B95E 3E 30      970       LD A,30H
B960 D3 06      980       OUT (6),A
B962 C9         990       RET
                        *
                        *OUTSTATUS ROUTINES
                        *
B963 FE 01     1000 OUTSTATS CP 1 *IS IT DEVICE ONE?
B965 28 0C     1010          JR Z,OUTSTATS1 *YES SO GO OUTSTATS1
B967 B7        1020          OR A *SET FLAFS
B968 28 04     1030          JR Z,OUTSTATSO *DEVICE TWO
B96A 3E FF     1040          LD A,-1 *DEVICE NOT IMPLEMENTED
B96C 37        1050          SCF *SET CARRY FLAG
B96D C9        1060          RET *RETURN TO CALLING ROUTINE
                        *
                        *OUTSTATUS FOR DEVICE 1
                        *
B96E DB 03     1070 OUTSTATS0 IN A,(3) *INPUT THE STATUS PORT
B970 E6 01     1080           AND 1 *MASK THE BIT
B972 C9        1090           RET *RETURN
                        *
                        *OUTSTATUS FOR DEVICE 2
                        *
B973 DB 05     1100 OUTSTATS1 IN A,(5) *INPUT STATUS PORT
B975 E6 01     1110           AND 1 *MASK BIT
B977 C9        1120           RET *RETURN
```

```
                    *
                    *INSTATUS
                    *
B978 B7       1130  INSTATS OR A *CLEAR A
B979 28 04    1140          JR Z,INSTATSØ *DEVICE Ø
B97B 3E FF    1150          LD A,-1 *DEVICE NOT DEFINED
B97D 37       1160          SCF *SET CARRY
B97E C9       1170          RET *RETURN
                    *
                    *STATUS OF DEVICE Ø
                    *
B97F DB Ø3    1180  INSTATSØ IN A,(3) *INPUT STATUS PORT
B981 E6 Ø2    1190           AND 2 *MASK
B983 ØF       1200           RRCA *SET BIT
B984 C9       1210           RET *RETURN
```

5.5 Bootup Sequence

MicroDoZ boots by the following proceedure:

When the computer is turned on, reset, or a JPE8ØØ is executed from MicroDoZ or Monitor, control is transferred to the ROM bootstrap program contained on the floppy disk controller. This program causes sector 4 on the disk in drive 1 to be loaded into the RAM of the computer. Control then passes to the code just loaded which proceeds to load sectors 5 and 6 immediately after itself.

The remaining sectors of MicroDoZ are then loaded below sector 4 (the first sector loaded). At this time the MicroDoZ jump is written to bytes ØØØØH to ØØØ2H, the I/O default byte is written to byte ØØØ3H, the default drive byte is written to byte ØØØ4H, and the MicroDoZ CALL jump is written to bytes ØØØ5H to ØØØ7H.

Once these actions are accomplished, control is passed to MicroDoZ itself by jumping to the first byte of MicroDoZ. At this point, MicroDoZ checks the turnkey byte flag to determine if MicroDoZ is to take some additional action. If the turnkey flag is zero (Ø), no action is taken except to display the prompt and MicroDoZ stops in the command mode waiting for a command.

If the turnkey flag is non zero, MicroDoZ transfers the number of bytes represented by the turnkey flag from the bootup buffer to the Common Command Buffer (CCB) located at ØØ8ØH. From this point, MicroDoZ attempts to excecute the commands in the CCB.

DOS2000

DOS2000 is a short assembly language program that "imitates" North Star DOS located at 2000H. The purpose of this program is two fold: first, to demonstrate the use of the MicroDoZ machine language interface, and second, to provide a program that allows other programs written for North Star DOS to function under MicroDoZ.

The anotated source is included in this section to allow programmers to "see" how MicroDoZ is interfaced.

6.1 Source Listine

```
* INTERFACE TO OLD NORTH STAR DOS
DOSORG EQU 2000H     *Origin of program
MICRODOZ EQU 5       *MicroDoZ CALL location
         ORG DOSORG
         JP DOZ      *Jump to DOS intry
         JP DOZ1
         DEFS 1
         RET         *Often routine
         DEFW 0
         JP DOS      *Boot jump
COUT JP COUT1        *Address of character out
CIN JP CIN1          *Address of character in
TINIT JP TINIT1      *Address of initialization
CONTC JP CONTC1      *Address of control C
HDERR JP HDERR1      *Address of hard disk error routine
DLOOK JP DLOOK1      *Address of DLOOK routine
DWRIT JP DWRIT1      *Address of DWRITe routine
DCOM JP DCOM1        *Address of DCOM routine
LIST JP LIST1        *Address of LIST routine
DOS JP DOZ
RWCHK DEFB 0
DOSERR JP DOSERR1
DENSITY DEFB 0
AUTO DEFB 1
IBLOC DEFW IBUFF
PAGES DEFB 0
ZPOINT DEFW 80H
DOZ1 LD HL,80H
         LD (ZPOINT),HL
         LD A,E
         OR A
         JP Z,DOS1
         JP ZIPPO
```

```
COUT1 PUSH BC
        LD C,1 *OUTPUT A CHARACTER
        CALL MICRODOZ
        POP BC
        RET
CIN1    PUSH BC
        LD C,Ø *INPUT A CHARACTER
        CALL MICRODOZ
        POP BC
        RET
TINIT1  RET *IO PORTS INITIALIZED BY MICRODOZ ALREADY
CONTC1  PUSH BC
        LD C,2 *MICRODOZ PANIC
        CALL MICRODOZ
        POP BC
        RET
PROMPT1 LD SP,STACK
        LD HL,PROMPT1
        PUSH HL
        JP PROMPT
DOZ LD C,3Ø *GET COMMAND BUFFER POINTER IN HL
        CALL MICRODOZ
        LD (ZPOINT),HL
DOS1 LD HL,PROMPT1
        PUSH HL
        LD SP,STACK
        PUSH HL
        LD HL,HDERR1
        LD (HDERR+1),HL
        LD HL,DOSERR1
        LD (DOSERR+1),HL
        LD HL,ODØ
        LD C,23 *EXECUTE COMMAND POINTED TO BY HL
        CALL MICRODOZ
ZIPPO LD C,5 *CLEAR SCREEN
        CALL MICRODOZ
        LD HL,(ZPOINT)
        LD A,(HL)
        CP ØDH
        JR Z,CKAUTO
        INC HL
        JP CR1
CKAUTO LD A,(AUTO)
        OR A
        JP Z,AUTOS
PROMPT LD B,"+"
        XOR A
        CALL COUT *OUTPUT THE PROMPT
        LD HL,IBUFF *IBUFF= THEN INPUT BUFFER
        LD DE,79*256 *D=MAX LEN E=CHARACTER COUNT
```

```
ILOOP XOR A
        CALL CIN *INPUT A CHARACTER
        CP 32 *CHECK FOR LESS THAN 32 IE A CONTROL CHARACTER
        JR C,CONTROLS *GO DO CONTROL CHARACTER
        LD (HL),A *SAVE CHARACTER IN BUFFER
        INC HL *BUMP THE BUFFER POINTER
        INC E *BUMP THE CHARACTER COUNTER
        LD B,A *PREPARE TO OUTPUT
        XOR A
        CALL COUT *OUTPUT THE CHARACTER
        DEC D *SEE IF BUFFER OVERFLOW
        JR NZ,ILOOP *IF NOT THEN MORE ELSE DROP THRU TO CARRIAGE RET
CR LD C,14 *DO A CARRIAGE RETURN LINE FEED
        CALL MICRODOZ
        LD (HL),ØDH *PUT CARRIAGE RET ON END OF BUFFER
        LD HL,IBUFF *POINT TO BEGINING OF BUFFER
        LD (ZPOINT),HL *SET COMMAND POINTER TO BUFFER
CR1 LD C,23 *EXECUTE THE DOS COMMAND POINTED TO BY HL
        CALL MICRODOZ
        JR NC,PROMPT *DO ANOTHER LINE
        OR A *SET FLAGS FROM A
        *PARSE FILE OR HARD DISK ERROR
        JP Z,DOSERR
        CP 7
        JP NC,DOSERR
        JP HDERR
*PARSE CONTROL CODES
CONTROLS CP ØDH
        JR Z,CR
        CP 8
        JR Z,BKSP
        CP 5FH
        JR Z,BKSP
        CP 7FH
        JR NZ,ILOOP
*DO A BACKSPACE
BKSP    LD A,E
        OR A
        JR Z,ILOOP *IF CHAR COUNT=Ø THEN DON'T BACKSPACE
        PUSH HL *SAVE POINTER
        LD HL,BKSP1 *POINT HL TO BKSPACE MESSAGE
        LD C,15 *OUTPUT THE MESSAGE
        CALL MICRODOZ
        POP HL *RESTORE BUFFER POINTER
        DEC HL *DELETE THE CHARACTER
        DEC E *DECREMENT THE CHAR COUNT
        INC E *INCREMENT THE MAX LENGTH
        JR ILOOP *DO MORE INPUT
BKSP1 DEFB 8 *THE BACKSPACE MESSAGE
        DEFB 32
        DEFB 8
        DEFB Ø *MESSAGES END WITH Ø
```

```
LIST1 PUSH HL *SAVE HL,DE,BC
        PUSH BC
        PUSH DE
        ADD 3ØH *A=CHR$(A+ASC("Ø"))
        LD (LIST2+5),A *FORM A DOSCMD STRING
        LD A,L
        ADD 3ØH
        LD (LIST2+3),A
        LD HL,LIST2 *POINT TO COMMAND JUST COMPILED
        LD C,23 *EXECUTE DOZCMD POINTED TO BY HL
        CALL MICRODOZ
        POP DE *RESTORE REGISTERS
        POP BC
        POP HL
        RET NC *NO PROBLEM IN LIST
        OR A *PARSE FILE OR HARD DISK ERRORS
        JP Z,DOSERR
        CP 7
        JP NC,DOSERR
        JP HDERR
LIST2 "LI#Ø 1"
       DEFB ØDH
*
DCOM1   PUSH AF
        LD A,C
        LD C,24 *SELECT THE DRIVE AND DENSITY
        CALL MICRODOZ
        POP AF
        LD C,25 *EXECUTE THE DCOM CALL
        CALL MICRODOZ
        RET NC
        OR A *PARSE FILE OR HARD DISK ERRORS
        JP Z,DOSERR
        CP 7
        JP NC,DOSERR
        JP HDERR
DEV DEFB Ø
*
*DLOOK IS DIFFERENT FOR MIRCODOZ
DLOOKØ INC HL
        LD A,(HL)
        CP 32
        JR Z,DLOOKØ
        CP "1"
        JP C,DOSERR
        CP 35H
        JP NC,DOSERR
        SUB "Ø"
        LD (DEV),A
        JP DLOOP1
```

```
DLOOK1 LD A,1
       LD (DEV),A
       LD A,(HL)
       CP 32
       JR Z,DLOOK4 *LOOKING FOR MT DIRECTORY SPACE
       XOR A
       LD (LFLAG),A
       LD DE,FNAME
       LD B,8
*MOVE FILENAME TO FNAME
DLOOP LD A,(HL)
       CP 32
       JR Z,DLOOP1
       CP ØDH
       JR Z,DLOOP1
       CP ","
       JR Z,DLOOKØ
       LD (DE),A
       INC HL
       INC DE
       DJNZ DLOOP
DLOOP1 EX DE,HL
       LD (HL),ØDH
       LD A,(DEV)
       ADD 3ØH
       LD (DD),A
       LD HL,DD1
       LD C,23 *EXECUTE DOSCMD DEFAULT DRIVE
       CALL MICRODOZ
       LD HL,FNAME
       LD C,26 *DODLOOK
       CALL MICRODOZ
       PUSH HL
       PUSH DE
       PUSH AF
       LD HL,DDSET
       LD C,23 *EXECUTE DOSCMD
       CALL MICRODOZ
       POP AF
       POP DE
       POP HL
       PUSH AF
       AND 8ØH *STRIP DIRECTORY DENSITY
       LD (DENSITY),A
       POP AF
       JR C,DLOOK2
       LD DE,8
       ADD HL,DE
       LD A,(DEV)
       RET
```

```
HDERR2 "HARD "
       "DISK "
       "ERROR "
       DEFB ØDH
       DEFB ØAH
       DEFB Ø
ODØ "ODØ"
       DEFB 5CH
       "IDØ"
       DEFB 5CH
       "DD1"
       DEFB ØDH
DD1 "DD"
DD     DEFB 1
       DEFB ØDH
DDSET "DD1"
       DEFB ØDH
```

```
DLOOK2 AND 3FH *STRIP OFF DIRECTORY DENSITY
        OR A
        JR Z,DLOOK3
        CP 7 *PARSE FILE OR HARD DISK ERROR
        JP C,HDERR
        JP DOSERR
DLOOK3 LD (FREE),HL
        EX DE,HL
        LD A,(DEV)
        SCF
        RET
FNAME DEFS 9
LFLAG DEFB 1
FREE DEFW Ø
DLOOK4 LD A,(LFLAG)
        OR A
        JP NZ,DOSERR
        LD A,-1
        LD (LFLAG),A
        LD HL,(FREE)
        LD DE,8
        ADD HL,DE
        LD A,(DEV)
        RET
DWRIT1 LD C,27 *WRITE DIRECTORY BUFFER TO DISK
        CALL MICRODOZ
        RET NC
        OR A *PARSE FILE OR HARD DISK ERRORS
        JP Z,DOSERR
        CP 7
        JP NC,DOSERR
        JP HDERR
IBUFF EQU 8ØH *DOS INPUT BUFFER LOCATION
        DEFS 8Ø
STACK DEFS Ø
AUTOS LD HL,TKEY
        JP CR1
TKEY "GOBASIC"
        DEFB ØDH
        DEFS 8Ø
        DEFB ØDH
DOSERR1 LD HL,FILERR
        LD C,15 *PRINT FILERR MESSAGE
        JP MICRODOZ
HDERR1 LD HL,HDERR2
        LD C,15 *PRINT HARD DISKERROR MESSAGE
        JP MICRODOZ
FILERR "FILE ERROR"
        DEFB ØDH
        DEFB ØAH
        DEFB Ø
```

MONITOR

The Monitor is an assembly language program designed to provide the user with liminted abilities to view, change, and manipulate the internal memory of the computer. Provisions are made to compare memory, fill memory, search memory, test memory, display memory in various modes, jump to assembly language programs, as well as the ability to input and output directly to Z8Ø ports.

As an added convience, the Monitor allows the execution of all MicroDoZ commands without the need to exit to MicroDoZ. The command syntax under the Monitor is similar to MicroDoZ.

7.1 Commands

This section details the Monitor commands and their proper usage. Each command is followed by the command syntax, a description of the command, and appropriate examples.

No action is taken on any command until the carriage return is entered and the Monitor has validated the syntax of the command. All commands may be edited using the standard editor as described in the **baZic** manual.

All commands are two or three letter sequences, alternately followed by one or more arguments. All commands must be in upper case only. All arguments passed are separated by spaces.

The arguments to commands are expressed by the following "rules":

All numbers passed are assumed to be hexadecimal unless they are followed by a "T" which means the number is decimal. Hex numbers may have an "H" following them but it is not required.

An ADDRESS must be a number in the range of ØØØØH to FFFFH or Ø to 65535T (decimal).

A BLOCK can be defined in one of three ways:

An ADDRESS is a BLOCK that is one byte long.

Two ADDRESSes separated by a hyphen (-) form a BLOCK that begins at the first ADDRESS and continues to the last ADDRESS. The last ADDRESS should not be less than the first.

An ADDRESS followed by a number separated by a comma. This BLOCK begins at the first address and continues the number of bytes specificed by the number.

A BYTE is any value which occupies a single byte of space. A BYTE can be any number from ØØH to FFH or Ø to 255T. Alternately, a BYTE can be any printable (non control) character that is enclosed in quotes.

7.1.1 Compare Memory

CM <BLOCK> <ADDRESS>

This command causes the memory defined by BLOCK to be compared with the memory starting at ADDRESS. The comparison is made on the same number of bytes as the BLOCK beginning with the ADDRESS location. When any pair of bytea at corresponding locations does not compare, the address and value are displayed.

As an example, if we wanted to compare 4K of memory starting at 4ØØØH with 4K of memory at FØØØ, the command would be issued as follows:

```
>CM 4ØØØ-4FFF FØØØ
>
```

If the entire memory of the two regions specified was exactly equivalent, the display would appear as demonstrated by the previous example and the Monitor prompt would immediately follow the command entered. If any memory locations were not equal, the address and memory content would be listed immediately following the command. An example of this condition follows:

```
>CM 4ØØØ,1ØØT FØØØ
4Ø45 E5 66 FØ45
4Ø46 E5 89 FØ46
>
```

The preceeding example indicates the Monitor found an E5 at address 4Ø45H and found a 66 at address FØ45H. Likewise, the address 4Ø46H contained an E5 while the address FØ46H contained a hexadecimal value of 89.

7.1.2 Fill Memory

FM <BLOCK> <BYTE>

This command causes the specified BLOCK to be filled by the specified BYTE. Each byte within the range is filled by the specified value. As an example, if we wanted the 4K region from FØØØ to FFFF to be filled with a BYTE value of FF, the command would appear as follow:

```
>FM FØØØ-FFFF FF
>
```

The printing of the prompt immediately after the command indicates the command was completed sucessfully.

7.1.3 Move Memory

MM <BLOCK> <ADDRESS>

The Move Memory command is used to move the contents of the memory defined in BLOCK to the location specified in ADDRESS. All moves are performed correctly reqardless of whether the move is "up" or "down" in memory or if the specification cause an overlap between the BLOCK and the ADDRESS.

As an example, if we wanted to move the block of memory from 4000H to 6000H to 1000H, the command would appear as follows:

```
>MM 4000-6000 1000
>
```

The printing of the prompt immediately after the command indicates the command was completed sucessfully.

7.1.4 Search Memory

SM <BLOCK> <BYTE> [,<STRING OF BYTES>]

The Search Memory command is used to search the specified BLOCK of memory for one or more BYTES. If more than one BYTE is specified, each BYTE must be separated from the others by a comma (,). In addition, if a STRING OF BYTES is given, the Monitor will only search for the series of values that you have specified and not for the occurrence of any one value.

As an example, if we wanted to search the region from F000H to FFFFH for an ASCII space, the command would appear as follows:

```
>SM F000-FFFF "P","A","S"
>
```

The preceeding example indicates that the specified values were not located within the range of memory specified. If the value had been found, the command and output from the monitor would appear as follows:

```
>SM F000,100T "f"
F045
>
```

The preceeding example indicates the Monitor "found" the requested value at memory location F045H. If the value had been found at other locations, these would have been listed also.

The Search Memory command also has the ability to use the Boolean operator NOT. This operation is performed by placing an N before any BYTE that is to be acted upon by the Boolean NOT.

As an example, the NOT operator can be used to find areas of Read Only Memory by the following sequence:

```
>FM E800,100 20
>SM E800,100 N20
```

In the example, we first fill the memory area in question with any BYTE such as 20 and then issue the Search Memory command to look for all occurrences of "NOT 20" (memory that was not set to the value specified).

7.1.5 Test Memory

TM <BLOCK> <BYTE>

This command is used to test the region of memory as defined by BLOCK. Each byte in this region is written to with an ascending pattern and after an amount of time approximately equal to the value of BYTE, the location is read and compared to the value written. If the two values do not compare, the address is reported as bad.

After each pass through the specified region, the results of the all passes are reported. The starting value is decremented and another pass is initiated.

An example of a memory test reporting no errors follows:

```
>TM 4000-5000 1
WRITING
READING
    1  PASSES COMPLETE      0  BAD PASSES
WRITING
READING
    2  PASSES COMPLETE      0  BAD PASSES
```

An example of a memory test reporting errors follows:

```
>TM E000,2 0
WRITING
READING
E000 FF  READ AS C3
E000 FE  READ AS 61
    1  PASSES COMPLETE      1  BAD PASSES
WRITING
READING
E000 FE  READ AS C3
E000 FD  READ AS 61
    2  PASSES COMPLETE      2  BAD PASSES
>
```

7.1.6 Display memory Hexadecimal

DH <BLOCK>

The Display Hex command is used to display the hexadecimal contents of the range of memory as specified in BLOCK. Two hex digits are displayed per byte with sixteen pairs displayed on each line.

As an example, if you wanted to display the contents of ten memory locations starting at address F000, the command and resulting display would appear as follows:

```
>DH F000,10T
F000   FE 7F FB E3 BE FE F7 FD E5 FF

>
```

7.1.7 Display memory Ascii

DA <BLOCK>

This command displays the memory of the specified BLOCK but shows the ASCII representation of the memory address directly below the actual value. This command is useful in finding string of characters or messages that may be imbedded in assembly code. Any control characters encountered are displayed as blanks (ASCII 32) and any values encountered that have the seventh (high order bit) high are printed with a minus sign directly preceeding the value.

If you wanted to display the ASCII value of the block of memory that started at F000H and continued for 10H bytes, the command and resulting display would appear as follows:

```
>DA F000,10
F000   E5 E5  2 47 45 66 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5
       -e -e    G  E  f -e -e -e -e -e -e -e -e -e -e

>
```

7.1.8 Display memory and Substitute

DS <ADDRESS>

The Display and Substitute command is usefull for determining the value at any single location and having the option of replacing that value with one selected by yourself. After each memory location is displayed, the contents can be changed by entering a hexadecimal number in the range of 0 to FF. If the value is not to be changed, the space bar may be pressed to see the next memory location. Entering a carriage return in response to the Monitor display of a memory value will terminate the command.

The Display and Substitute command was used to create the example as shown in the Display Ascii command. After first Filling Memory in the range of FØØØ,1ØØ with the value E5, the pattern of the previous example was created by the following sequence:

```
>DS FØØ2
FØØ2   E5 Ø2
FØØ3   E5 "G"
FØØ4   E5 45
FØØ4   E5 66
FØØ5   E5
>
```

7.1.9 Jump Program

JP <ADDRESS>

The Jump Program command is used to transfer control from the Monitor to another assembly language program. The ADDRESS specified must be the proper address for executing the program.

7.1.1Ø Operating System

OS

This command causes control to be passed back to MicroDoZ (the disk operating system). No parameters are passed to this command. Remember that all MicroDoZ commands can be executed from the monitor without returning to the operating system.

7.1.11 Initial Load

IL

The Initial Load command is used to "re-boot" the computer. This command has the same affect as if the operator had got up from his/her chair and physically rebooted the computer by pressing the red reboot switch. This command performs a Jump Program command with E8ØØH (the disk controller ROM) as the argument.

7.1.12 INPut a byte from a port

INP <PORT ADDRESS>

The INPut command is useful for determining the value that currently resides at one of the 256 Z8Ø ports. The value will be displayed directly below the command as shown in the following example:

```
>INP 6
35
>
```

7.1.13 OUTput a byte to a port

OUT <PORT ADDRESS> <VALUE>

The OUTput command is used to output a byte value directly to the specified port. This command will not be checking the status of the port as is the normal sequence in printer output but will issue the value to the port regardless of its status. As an example, the value 12 (Form Feed) is output to the "printer port" which is port 4 in this example.

```
>OUT 4 12

>
```

The value OUTput to the port must be a BYTE value in the range of Ø to FFH. Care must be taken in using this command because the outputting of the wrong BYTE to the wrong port can cause disturbing results. If an incorrect value is output to many status ports, the port can "lock down", resulting in the user having to reboot the computer to again do I/O.

7.2 Line Editor

To allow the easy creation and changing of commands, Monitor has a "built in" line editor. This editor may be used while entering a command or any time before the carriage return is entered.

Internally in the Monitor are two line buffers. As you type in any command, you are typing into the editor input buffer. This means that any line of program just typed is available for editing.

To keep track of the changes in a command, an internal pointer is used to "point" to characters in both the editor buffer and the input buffer. As every character of text (except the editor commands) is typed in, both pointers are advanced one character at a time so that the pointer to the input buffer is always pointing to the current character position. The editor keeps track of both pointers and allows the programmer to transfer information from the editor buffer to the input buffer.

The following commands are available in the line editor and are used to copy characters or sets of characters from the editor buffer to the input buffer: ^G (control G), ^N, ^A, ^Q, ^Z, ^D, and ^Y. All of the editor commands are control characters (the control key is pressed at the same time the character is pressed).

7.2.1 Control G

Control G copies the entire contents of the editor buffer from the current cursor position within the line to the input buffer. Control G may be used to view the editor buffer after a command has been placed in the editor buffer. After the control G has been executed, the user should be able to see the command that was in the editor buffer and the cursor will be at the end of the line.

At this point the can take one of two actions: press the return key which executes the command or press the control N command which leaves the command in the editor buffer and returns the cursor to the beginning of the line. This procedure is very useful when viewing a command prior to doing the actual editing.

If the pointer is already at the end of the command in the editor buffer, the bell will be sounded if a control G command is issued by the programmer.

7.2.2 Control N

The control N command is partially discussed in the control G section. The purpose of the control N command is to allow the programmer to restart the editing of a command by cancelling the command presently on the screen and returning the cursor to the beginning of the line for further editing. An "@" sign is printed when the control N command is typed to indicate to the programmer the line has been cancelled.

7.2.3 Control A

The control A command is used to copy one character from the editor buffer to the input buffer. The pointers can be pointing to different characters in each buffer so the command "takes" the character pointed to in the editor buffer and places it in the input buffer.

The character is also printed to the CRT as if the user had typed the character into the input buffer. Both pointers are incremented after this command. If no character is in the editor buffer, the "bell" is sounded on the CRT to let the user know that the command was illegal.

7.2.4 Control Q

This is the backspace command. It is identical to the backspace key on many CRTs or the control H key. Both pointers are decremented by this command.

When a backspace command is detected a backspace is printed followed by a space (ASCII 32) followed by another backspace. If one or both pointers are at the beginning of a line, the command sounds the "bell" of the CRT to let the programmer know of the mistake.

7.2.5 Control Z

This command is used to erase one character at a time from the input buffer. The command prints a "%" sign to inform the programmer that the character position occupied by the "%" sign has been erased and is no longer in the input buffer. If the input buffer pointer is already at the beginning of the line, the bell is sounded to inform the programmer of the error.

7.2.6 Control D

The control D command is the search and find command. Upon executing the command, Monitro will wait for one additional character to be input. Once this character is input, the editor buffer is searched until the first occurrence of the specified character is found and the contents of the editor buffer up to but not including that character is copied to the input buffer. If the character is not located in the editor buffer, nothing is copied to the input buffer and the bell is sounded.

7.2.7 Control Y

The control Y command is used to "turn on and off" the insert mode. By executing the control Y command to turn on the insert mode, characters may be inserted into the input buffer that were not in the editor buffer. Once the characters have been entered, control Y can be toggled off again to allow other characters to be copied or deleted from the input buffer.

When the insert mode is toggled on, a "less than" character (<) will be printed to inform the programmer that the editor is in the insert mode. When the insert mode is toggled off, a "greater than" character (>) is printed to inform the programmer that the insert mode is off. These characters are **not** part of the command itself but are placed in the line shown on the CRT so the user will know the status of the insert mode.

7.3 Execute MicroDoZ Commands

As has been previously mentioned, the monitor is capable of executing all of the MicroDoZ commands. This feature is extremely "handy" because files can be loaded, chages made through the use of Monitor commands, and the file saved back on the disk without the user having to swith from one program to another. The following is a list of the MicroDoZ commands that are available from the Monitor:

LIst the directory
INitialize a disk
WRite disk or Read Disk
Save File or Load File
Jump Program
GO program
CReate a file

TYpe a file
DElete a file
Output Device or Input Device
Define Drive
REname a file
Read Only or Write Enable
Set System or Non System
Attribute Field

When a command is issued from the Monitor, the Monitor searches its internal command list. If the command is not found in its command table, the Monitor calls the DOZCOMMAND location (ØØØ5h) with the users input in the command buffer. The MicroDoZ command table is then searched to determine if the command is a legal MicroDoZ command. If the command is found in the MicroDoZ command table, the command is executed.

If MicroDoZ does not find the command in its command table, it assumes the command is an implied GO command, so the proper disk directory is searched for a type 1 file with the same name as the command. If such a file is found, it is immediately loaded at its GO address and control is transferred to the program. Thus, any other program can be executed directly from the Monitor.

7.4 Source Listing

To facilate the use of MicroDoZ, the complete source listing is included to the Monitor program. Most of the features of the machine language interface are demonstrated in the Monitor program. This source listing shows how easy it is to write programs while using the resources of MicroDoZ.

```
*MONITOR
*1-14-81
*COPYRIGHT (C) 1981, MICRO MIKE'S, INC.
        ORG 1ØØH
        JP BEGIN
                        *DEFINE MICRODOZ ROUTINES
IOBYTE EQU 3            *DEFAULT I/O LOCATION
MDOS EQU 5              *MICRODOZ CALL LOCATION

INPUT LD C,7    *INPUT CHARACTER INTO A REG USING IOBYTE FOR DEVICE
        JP MDOS
OUTPUT LD C,8   *OUTPUT CHARACTER IN B TO DEVICE DEFINED BY IOBYTE
        JP MDOS
PANIC LD C,9    *PANIC CHECK CONSOLE STATUS FOR CONTROL C
        JP MDOS
CLS     LD C,12*CLEAR THE CRT SCREEN HOME CURSOR
        JP MDOS
CRLF LD C,14    *OUTPUT A CARRIAGE RET LINE FEED TO IOBYTE DEVICE
        JP MDOS
MSG LD C,15     *OUTPUT TO IOBYTE DEVICE
        JP MDOS*THE MESSAGE POINTED TO BY HL REGISTERS
                *MESSAGE ENDS WHEN A Ø SEEN
```

```
HEX16 LD C,16  *CONVERT BINARY NUMBER IN HL REGISTER PAIR TO
       JP MDOS*4 DIGIT HEX NUMBER SUPPRESS LEADING ZEROS
              *PRINT NUMBER ON IOBYTE DEVICE WITH TRAILING SPACE
HEX8 LD C,17   *SAME AS HEX16 BUT A 2 DIGIT HEX NUMBER
       JP MDOS
DEC16 LD C,18  *SAME AS HEX16 BUT DECIMAL
       JP MDOS
DEC8 LD C,19   *SAME AS HEX16 BUT DECIMAL Ø..255
       JP MDOS
TAB LD C,22    *TAB CURSOR THE VALUE IN THE A REGISTER
       JP MDOS


DODOS LD C,23  *DO A DOS COMMAND POINTED TO BY HL REGISTER PAIR
       JP MDOS*COMMAND MUST END WITH ØDH (CARRIAGE RETURN)
EXIT JP Ø

              *END MICRODOZ ROUTINE DEFINITIONS
              *
              *BEGIN MONITOR

BEGIN LD SP,STACK        *SETUP STACK
       CALL CLS          *CLEAR THE SCREEN
       CALL IBUFFINIT    *INITIALIZE INPUT BUFFER
       CALL CRLF         *PRINT CARRIAGE RET LINE FEED
       LD HL,SIGNON      *LOAD HL WITH ADDRESS OF SIGN ON MESSAGE
       CALL MSG          *PRINT SIGN ON MESSAGE
BEGINØ CALL CRLF
BEGIN1 LD HL,PROMPT      *LOAD HL WITH ADDRESS OF PROMPT MESSAGE
       CALL MSG          *PRINT THE PROMPT
       XOR A
       LD (IBFLAG),A     *SET FLAG FOR COMMAND INPUT
       CALL IBUFF        *INPUT A LINE OF COMMANDS
       CALL NC,EXECUTE   *IF INPUT OK THEN GO PARSE COMMANDS
       JR NC,BEGIN1      *DO ANOTHER COMMAND LINE
       LD HL,ERROR
       CALL MSG          *PRINT ERROR MESSGAE
       CALL CRLF
       JR BEGIN1         *DO ANOTHER COMMAND LINE

*SIGNON MESSAGE
SIGNON "MONITOR "
       "COPYRIGHT "
       "(C) "
       "198Ø "
       "MICRO "
       "MIKE'S "
       "INC."
       DEFB Ø            *MESSAGE ENDS WITH Ø

*PROMPT MESSAGE
PROMPT ">"
       DEFB Ø
```

```
*ERROR MESSAGE
ERROR "?"
        DEFB Ø

*STACK SPACE
        DEFS 8Ø
STACK DEFS Ø

*ECHO THE FOLLOWING WHEN A BACKSPACE IS INPUT
BKSPSTR DEFB 8
        DEFB " "
        DEFB 8
        DEFB Ø
IBFLAG DEFB Ø
IBUFFINIT RET              *UNUSED ROUTINE

*
*PARSE COMMANDS
*

EXECUTE LD DE,COMMDS       *DE POINTS TO COMMAND TABLE
        LD C,NCMDS         *C= NUMBER OF COMMANDS IN TABLE
EXECØ LD A,(DE)
        CP (HL)            *COMPARE 1ST BYTE OF COMMAND
        JR NZ,EXEC1        *JMP IF NOT A MATCH
        INC HL
        INC DE
        LD A,(DE)
        CP (HL)            *COMPARE SECOND BYTE OF COMMAND
        DEC HL
        DEC DE
        JR Z,EXEC2         *JMP IF COMMAND MATCH
EXEC1 INC DE               *MOVE DE TO NEXT ENTRY IN CMD TABLE
        INC DE
        INC DE
        INC DE
        DEC C              *HOW MANY ENTRIES
        JR NZ,EXECØ        *JMP IF MORE ENTRIES IN CMD TABLE
        JP DODOS           *CMD NOT IN MON GIVE IT TO MICRODOZ
*
*FOUND COMMAND IN MONITOR TABLE NOW EXECUTE IT
*
EXEC2 INC HL               *POINT HL TO CHARACTER IN IBUFF AFTER CMD
        INC HL
        INC DE             *POINT DE TO ADDRESS OF COMMAND IN TABLE
        INC DE
        PUSH HL            *SAVE IBUFF POINTER
        EX DE,HL           *HL= ADDRESS OF ADDRESS OF ROUTINE
        LD E,(HL)
        INC HL
        LD D,(HL)          *PUT COMMAND ADDRESS IN DE
        EX DE,HL           *MOVE COMMAND ADDRESS TO HL
        EX (SP),HL         *PUT COMMAND ADDRESS ON STACK
                           *HL = IBUFF POINTER
        RET                *RETURN TO COMMAND ADDRESS
```

```
*
*COMMAND TABLE
*TWO CHARACTER COMMAND FOLLOWED BY ADDRESS OF COMMAND
*
COMMDS "CM"
        DEFW CM
        "FM"
        DEFW FM
        "MM"
        DEFW MM
        "SM"
        DEFW SM
        "TM"
        DEFW TM
        "DH"
        DEFW DH
        "DA"
        DEFW DA
        "DS"
        DEFW DS
        "OS"
        DEFW OS
        "IL"
        DEFW IL
        "IN"
        DEFW INP
        "OU"
        DEFW OUT
NCMDS EQU 12            *NUMBER OF COMMANDS IN THE TABLE
*END OF COMMAND TABLE
*

*COMPARE MEMORY
CM CALL BLOCK    *BLOCK RETURNS WITH DATA ON THE STACK IF CARRY=Ø
        RET C           *ERROR IN BLOCK
        CALL ADDRESS *ADDRESS RETURNS WITH DATA IN DE REGISTER PAIR
        JP C,BLKERR     *ERROR IN ADDRESS
CM1 POP HL              *THE START OF THE DATA BLOCK
        LD A,(DE)       *A= BYTE OF <ADDRESS>
        CP (HL)         *COMPARE WITH BYTE OF <BLOCK>
        INC HL          *BUMP <BLOCK START>
        INC DE          *BUMP <ADDRESS>
        PUSH HL         *SAVE <BLOCK START>
        JR Z,CM2        *IF <BLOCK BYTE>=<ADDRESS BYTE>
        PUSH DE         *SAVE <ADDRESS>

*ADJUST <BLOCK START> AND <ADDRESS> TO POINT TO
*BYTE INEQUALITY
        DEC DE
        PUSH DE
        DEC HL
        PUSH HL
        CALL HEX16      *PRINT HEX <BLOCK START>
        POP HL
        LD A,(HL)
```

```
        LD L,A
        LD H,Ø
        CALL HEX8          *PRINT HEX VALUE OF BYTE AT <BLOCK START>
        POP HL             *HL= <ADDRESS>
        PUSH HL
        LD A,(HL)
        LD L,A
        LD H,Ø
        CALL HEX8          *PRINT HEX VALUE OF BYTE AT <ADDRESS>
        POP HL
        CALL HEX16         *PRINT HEX OF <ADDRESS>
        CALL CRLF          *DO A CARRIAGE RET LINE FEED
        POP DE
CM2 EXX                    *EXCHANGE HL HL' DE DE' BC BC'

*LIMIT CHECKS FOR CONTROL C
*AND IF <BLOCK START> > <BLOCK END> ON STACK
        CALL LIMIT
        EXX
        JR NC,CM1          *DO SOME MORE
CEXIT OR A
        POP HL             *CLEAR <BLOCK PARAMETERS FROM STACK>
        POP HL
        RET

*
*FILL MEMORY

FM CALL BLOCK
        RET C
        CALL BYTE
        JR NC,FMØ          *JMP IF VAILID BYTE
        CP 22H             *IS IT A QUOTE
        JP NZ,BLKERR       *NOT A QUOTE REPORT ERROR
        INC HL
        LD A,(HL)
        LD E,A       *PUT LITERAL VALUE OF BYTE AFTER QUOTE IN E
FMØ LD A,E                 *PUT VALUE TO FILL IN A
FM1 POP HL                 *<BLOCK START>
        LD (HL),A          *FILL <BLOCK START>,A
        INC HL             *BUMP <BLOCK START>
        PUSH HL            *PUT BACK ON STACK

        CALL LIMIT         *LIMIT CHECKS PANIC AND BLOCK LIMITS
        JR NC,FM1          *DO MORE
        JR CEXIT           *CLEAR STACK AND RET

*
*MOVE MEMORY

MM CALL BLOCK
        RET C              *ERROR IN BLOCK
        CALL ADDRESS
        JP C,BLKERR        *ERROR IN ADDRESS
```

```
*THE FOLLOWING DETERMINES WHICH DIRECTION TO MOVE
        POP HL              *START
        EX DE,HL            *DE=START HL=ADDR
        EX (SP),HL          *HL=END STK=ADDR
        PUSH DE             *START

*NEGATE DE FOR SUBTRACTION
        LD A,E
        CPL
        LD E,A
        LD A,D
        CPL
        LD D,A
        INC DE
        ADD HL,DE           *HL=LENGHT
        POP DE              *START
        EX (SP),HL          *HL=ADDR STK=LEN
        LD A,D
        CP H
        JR NZ,MM1
        LD A,E
        CP L
MM1 EX DE,HL                *DE=DEST HL=START
        POP BC              *LENGTH
        JR C,MM2            *START < DEST

*INRCREMENTING MOVE
        LDIR
        RET
MM2 ADD HL,BC
        DEC HL
        EX DE,HL
        ADD HL,BC
        DEC HL
        EX DE,HL
*DECREMENTING MOVE
        LDDR
        OR A
        RET

*
*SEARCH MEMORY
SM CALL BLOCK
        RET C
        CALL LIST

*LIST COMPILES THE BYTES TO SEARCH FOR ALONG WITH
*THE BOOLEAN OPERATOR
*THE LIST IS COMPILED AT BUFFER
*FORMAT BOOLEAN,BYTE
*C REG = NUMBER OF BYTES TO COMPARE
```

```
        JR NC,SM1
BLKERR POP HL           *CLEAR BLOCK PARAMETERS FROM STACK
        POP HL
        RET
SM1  POP DE             *DE=<BLOCK START>
        PUSH DE
        PUSH BC
        LD HL,BUFFER
SM2  LD B,(HL)          *LOGIC SENSE
        INC HL
        LD A,(DE)       *MEMORY BYTE
        CP (HL)         *LIST BYTE
        INC DE
        INC HL
        JR NZ,SM3       *NOT A MATCH
        LD A,B
        OR A
        JR NZ,SM5       *NOTMATCH
SM6  DEC C
        JR NZ,SM2       *DO NEXT LIST BYTE

*MATCH PRINT IT
        POP HL          *OLD BC
        LD (TEMP),HL    *SAVE IT
        POP HL
        PUSH HL         *ADDRESS
        CALL HEX16      *PRINT HEX ADDRESS OF MATCH
        CALL CRLF
SM4  POP HL             *BUMP <BLOCK START>
        INC HL
        PUSH HL
        CALL LIMIT
        LD HL,(TEMP)    *RESTORE BC VALUE
        PUSH HL
        POP BC
        JR NC,SM1       *DO MORE
        POP HL          *CLEAR STACK OF BLOCK PARAMETERS
        POP HL
        OR A        *CLEAR CARRY SO ERROR MESSAGE DOESN'T DISPLAY
        RET
TEMP DEFW Ø             *STORAGE FOR BC IN SM ROUTINES
SM3  LD A,B
        OR A
        JR NZ,SM6
SM5  POP HL
        LD (TEMP),HL
        JR SM4
```

```
*
*DISPLAY HEX
DH CALL BLOCK
        RET C               *ERROR IN BLOCK
DH1 POP HL                  *<BLOCK START>
        LD (ZIPPER),HL
        PUSH HL
        CALL HEX16          *PRINT HEX ADDRESS
        LD A,2
        CALL TAB            *TAB 2 SPACES

* SETUP TO PRINT 16 BYTES
        LD A,16
        LD (COUNT),A
DH2 POP HL
        LD A,(HL)           *BYTE AT <BLOCK START>
        INC HL              *BUMP <BLOCK START>
        PUSH HL
        LD L,A
        LD H,Ø
        CALL HEX8           *PRINT HEX VALUE OF BYTE
        CALL LIMIT
        JR C,DH3            *END OF BLOCK
        LD A,(COUNT)
        DEC A
        LD (COUNT),A
        JR NZ,DH2           *16 BYTES NOT FINISHED
        CALL DA1            *IF DA THEN DISPLAY CHR$(BYTE)
        JR DH1
DH3 CALL DA1
        POP HL              *CLEAR BLOCK PARAMETERS
        POP HL
        CALL CRLF
        OR A                *RESET CARRY SO ERROR DOESN'T DISPLAY
        RET
COUNT DEFB Ø

*
* DISPLAY ASCII USING DISPLAY HEX WITH DAFLAG SET
DA LD A,-1
        LD (DAFLAG),A
        CALL DH
        LD A,Ø
        LD (DAFLAG),A
        RET
DAFLAG DEFB Ø
```

```
DA1 CALL CRLF
        LD A,(DAFLAG)
        OR A
        RET Z
        POP HL
        POP DE
        PUSH DE
        PUSH HL
        LD HL,(ZIPPER)
        PUSH HL
        LD A,6
        CALL TAB
DA2 LD A,1
        CALL TAB
        POP HL
        PUSH HL
        LD A,(HL)          *BYTE TO DISPLAY

*IF BYTE>127 THEN PRINT "-" ELSE PRINT " "
        AND 80H
        LD B," "
        JR Z,DA3
        LD B,"-"
DA3 CALL OUTPUT
        POP HL
        LD A,(HL)          *BYTE TO DISPLAY
        AND 7FH            *STRIP BIT 7
        LD B,A

*IF BYTE IS LESS THAN A SPACE IT MUST BE
*A CONTROL CHARACTER SO PRINT A SPACE
*ELSE PRINT CHR$(BYTE)

        CP " "
        JR NC,DA4
        LD B," "
DA4 CALL OUTPUT
        INC HL
        LD A,E
        CP L
        JP Z,CRLF
        PUSH HL
        JR DA2
ZIPPER DEFS 2

*
*DISPLAY SUBSTITUTE
DS CALL ADDRESS
        RET C
        EX DE,HL           *HL= <ADDRESS>
        LD A,-1
        LD (IBFLAG),A      *SET IBUFF FLAG FOR DS  MODE
```

```
DS1 PUSH HL
        CALL HEX16          *PRINT HEX ADDRESS
        LD A,2
        CALL TAB
        POP HL
        PUSH HL
        LD A,(HL)
        LD L,A
        LD H,Ø
        CALL HEX8           *PRINT HEAX VALUE OF BYTE AT <ADDRESS>
        CALL IBUFF          *INPUT VALUE OR RESPONSE
        JR C,DS2            *ERROR IN IBUFF
        LD A,(HL)           *LOOK AT INPUT STREAM
        CP 32
        JR Z,DS4            *IF SPACE THEN NEXT ADDRESS
        CP ØDH
        JR NZ,DS3           *IF NOT CARR RET THEN IS A VALUE
DS2 POP HL
        RET
DS3 CP 22H                  *IS INPUT A QUOTE
        JR NZ,DS5

*QUOTE GET LITERAL VALUE OF BYTE IN E
        INC HL
        LD E,(HL)
        JR DS6
DS5 CALL BYTE               *CONVERT HEX TO BINARY IN E
        RET C
DS6 POP HL                  *SUBSTITUTE BYTE AT <ADDRESS>
        LD (HL),E
        INC HL              *BUMP <ADDRESS>
        JR DS1
DS4 POP HL                  *NO SUBS JUST BUMP <ADDRESS>
        INC HL
        JR DS1

*
*EXIT TO MICRODOZ
OS JP EXIT

*EXIT TO BOOT PROM
IL JP ØE8ØØH

*
*LIMIT
*LIMIT ASSUMES THAT BLOCK PARAMETERS ARE ON THE STACK
*LIMIT CHECKS FOR A CONTROL C (LIMIT1 DOESN'T)
*LIMIT & LIMIT1 CHECKS FOR <BLOCK START> > <BLOCK END>
*RETURN CARRY=1 IF CONTROL C OR LIMIT PASSED
LIMIT PUSH AF
        CALL PANIC
        JR C,LIM1           *FOUND A CONTROL C
        POP AF
```

```
LIMIT1 POP BC              *RETURN FOR LIMIT
       POP HL              *<BLOCK START>
       POP DE <BLOCK END>
       PUSH DE
       PUSH HL
       PUSH BC
       PUSH AF

*NEGATE DE REGISTER PAIR
       LD A,E
       CPL
       LD E,A
       LD A,D
       CPL
       LD D,A
       INC DE
       POP AF

*SUBTRACT EFFECTIVLY HL=HL-DE
       ADD HL,DE
       RET
LIM1 POP AF
       SCF
       RET

*
*LIST ON RETURN, REG C=NUMBER OF BYTES IN LIST
*FORMAT BOOLEAN,BYTE
*BOOLEAN=Ø1 THEN MATCH BYTE
*BOOLEAN=-1 THEN MATCH NOT BYTE

LIST LD DE,BUFFER
       LD C,Ø
LIST1 CALL SKIPSP
       CP "N"              *CHECK FOR NOT
       LD A,Ø
       JR NZ,LIST2
       INC HL
       LD A,-1
LIST2 LD (DE),A
       INC DE
       CALL SKIPSP
       CP 22H              *IS IT A QUOTE VALUE
       JR NZ,LIST3
       INC HL
       LD A,(HL)
       LD (DE),A
       INC DE
       INC C
       INC HL
       CALL SKIPSP
       CP 22H
       SCF
       RET NZ
       INC HL
```

```
LIST4 CALL SKIPSP
        INC HL
        CP ","
        JR Z,LIST1          *MORE ITEMS IN LIST
        OR A
        RET
LIST3 PUSH DE               *DO A HEX OR DECIMAL VALUE
        CALL BYTE
        LD A,E
        POP DE
        RET C
        LD (DE),A
        INC DE
        INC C
        JR LIST4

*BYTE CONVERTS HEX OR DECIMAL INPUT TO BINARY IN DE
*CARRY=1 IF VALUE > 255 DECIMAL
*HL POINTS TO INPUT STREAM BUFFER

BYTE CALL SKIPSP
        CALL NUMB
        RET C
        LD A,D
        OR A
        RET Z
        SCF
        RET

*
*ADDRESS CONVERTS HEX OR DECIMAL TO BINARY IN DE
ADDRESS CALL SKIPSP
        JP NUMB

*
*SKIP SPACES
SKIPSP LD A,(HL)
        CP " "
        RET NZ
        INC HL
        JR SKIPSP

*BLOCK
*HL POINTS TO STRING
*CARRY=1 ERROR
*CARRY=Ø THEN STACK IN ORDER OF POPS
*CONTAINS BEGINNING ADDRESS AND ENDING ADDRESS
```

```
BLOCK CALL SKIPSP       *SKIP SPACES
        CALL NUMB       *EVALUATE A NUMBER
        RET C
        EX DE,HL
        EX (SP),HL
        PUSH HL
        EX DE,HL
        CALL SKIPSP     *SKIP SPACES
        CP ØDH          *LOOK FOR CARRIAGE RETURN
        JR NZ,BLOCK1
        POP HL
        POP DE
        PUSH DE
        PUSH DE
        PUSH HL
        OR A
        RET
BLOCK1 CP "-"           *LOOK FOR BLOCK SEPARATOR
        JR NZ,RANGE     *NO, SEE IF RANGE
        INC HL
        CALL SKIPSP     *SKIP SPACES
        CALL NUMB       *EVALUATE SECOND NUMBER
        JR NC,BLOCK2
RERR POP HL             *ERROR CONDITION
        POP DE
        PUSH HL
        SCF             *SET CARRY FLAG
        RET             *AND RETURN
BLOCK2 EX DE,HL
        POP BC          *RET
        EX (SP),HL      *HIGH TO STK
        PUSH HL         *LOW TO STK
        PUSH BC         *RET TO STK
        EX DE,HL
        OR A
        RET
RANGE CP ","            *LOOK FOR RANGE SEARATOR
        JR NZ,RERR      *NO, SO ERROR CONDITION
        INC HL
        CALL SKIPSP     *SKIP SPACES
        CALL NUMB       *EVALUATE SECOND NUMBER
        JR C,RERR       *IF CARRY SET THEN ERROR
        POP BC          *RET
        EX (SP),HL      *HL = LOW
        PUSH BC         *RET
        LD B,H          *BC=LOW
        LD C,L
        ADD HL,DE
        LD D,B
        LD E,C          *DE=LOW
        POP BC          *RET
        EX (SP),HL      *HIGH TO STK HL= POINTER
        PUSH DE
        PUSH BC
        RET
```

```
*NUMB
*NUMBER PART OF MONITOR
*HL POINTS TO STRING OF NUMBERS
*CKNUM RET C=1 IF NOT "Ø".."9"
CKNUM CP "Ø"
        RET C
        CP ":"
        CCF
        RET

*CKHEXLET RET C=1 IF NOT HEX LETTER
CKHEXLET CP "A"
        RET C
        CP "G"
        CCF
        RET
NUMBER CALL CKNUM          *SEE IF VALID NUMBER
        JR NC,NUMBER1
        CALL CKHEXLET      *SEE IF VALID HEX LETTER
        RET C
        SUB "A"-1Ø
        OR A
        RET
NUMBER1 SUB "Ø"
        OR A
        RET

*NUMB COUNTS DIGITS IN C
*PUSHES BINARY OF VAILID DIGIT ONTO STACK
NUMB LD C,Ø
        LD A,(HL)
        CALL NUMBER
        RET C
NUMB1 PUSH AF
        INC C
        INC HL
        LD A,(HL)
        CALL NUMBER
        JR NC,NUMB1

*NUMBER IS ON STACK NOW ASSEMBLE INTO 16 BIT BINARY
        CP "T"             *LOOK FOR DECIMAL INDICATOR
        JR Z,DECIMAL
HEX CP "H"                 *LOOK FOR HEX INDICATOR
        JR NZ,HEX1
        INC HL
HEX1 POP AF
        LD D,Ø
        LD E,A
        DEC C
        JR Z,NRET
        POP AF
        CALL ROT
        OR E
        LD E,A
```

```
        DEC C
        JR Z,NRET
        POP AF
        LD D,A
        DEC C
        JR Z,NRET
        POP AF
        CALL ROT
        OR D
        LD D,A
        DEC C
        JR Z,NRET
        SCF
        RET
NRET OR A
        RET
ROT RLCA
        RLCA
        RLCA
        RLCA
        AND ØFØH
        RET
DECIMAL INC HL
        POP AF
        LD E,A
        LD D,Ø
        DEC C
        JR Z,NRET
        LD B,1
DEC1 POP AF
        PUSH HL
        LD L,A
        LD H,Ø
        PUSH DE
        PUSH BC
        CALL X1Ø
        POP BC
        POP DE
        ADD HL,DE
        EX DE,HL
        POP HL
        INC B
        DEC C
        JR Z,NRET
        LD A,B
        CP 5
        CCF
        RET C
        JR DEC1
```

```
X1Ø LD D,H
        LD E,L
        ADD HL,HL
        ADD HL,HL
        ADD HL,DE
        ADD HL,HL
        DEC B
        RET Z
        JR X1Ø

*TEST MEMORYØ
*PART OF MONITOR
TM CALL BLOCK           *RANGE TO TEST
        RET C
        CALL BYTE       *VALUE OF TIME DELAY
        JR NC,TM1
        POP HL
        POP HL
        RET
TM1     LD A,E
        INC A
        LD (TIME),A
        XOR A
        LD (ZIP),A
        LD HL,Ø
        LD (TMPASS),HL
        LD (TMBAD),HL
TM2     LD HL,WRITING
        XOR A
        LD (TMBADF),A
        CALL MSG
        CALL CRLF
        LD B,-1
        LD (CUP),BC
        POP HL
        POP DE
        PUSH DE
        PUSH HL
        PUSH DE
TM3 PUSH HL
        LD (CUP),BC
        LD A,(ZIP)
        ADD B
        LD (HL),A
        POP HL
        INC HL
        PUSH HL
        CALL LIMIT1
        JR C,TM4
        CALL PANIC
        JP Z,TMX        *4 POPS RET
        LD BC,(CUP)
        POP HL
        DJNZ TM3
        LD B,-1
```

```
        LD (CUP),BC
        JR TM3
TM4     LD A,(TIME)
TM5 DEC A
        JR Z,TM6
        LD B,1
X1 LD H,Ø
X2 LD L,Ø
X3 EX (SP),HL
        EX (SP),HL
        DEC L
        JR NZ,X3
        DEC H
        JR NZ,X2
        DEC B
        JR NZ,X1
        JR TM5
TM6 CALL PANIC          *CHECK FOR CONTROL C
        JP Z,TMX
        LD HL,READING   *LOAD HL WITH ADDRESS OF READING MESSAGE
        CALL MSG        *DISPLAY MESSAGE
        CALL CRLF       *OUTPUT CARRIAGE RETURN AND LINE FEED
        LD B,-1
        LD (CUP),BC
        POP HL
        POP HL
        POP HL
        POP DE
        PUSH DE
        PUSH HL
        PUSH DE
TM7 PUSH HL
        LD (CUP),BC
        LD A,(ZIP)
        ADD B
        CP (HL)
        JR Z,TM8
        PUSH HL
        PUSH BC
        CALL HEX16
        POP BC
        LD L,B
        LD H,Ø
        CALL HEX8
        LD HL,READAS
        LD A,1
        LD (TMBADF),A
        CALL MSG
        POP HL
        LD L,(HL)
        LD H,Ø
        CALL HEX8
        CALL CRLF
```

```
TM8  POP HL
         INC HL
         PUSH HL
         CALL LIMIT1
         JR C,TM9
         CALL PANIC
         JP Z,TMX
         LD BC,(CUP)
         POP HL
         DJNZ TM7
         LD B,-1
         LD (CUP),BC
         JR TM7
TM9  LD HL,(TMPASS)
         INC HL
         LD (TMPASS),HL
         LD A,(TMBADF)
         OR A
         JR Z,TM90
         LD HL,(TMBAD)
         INC HL
         LD (TMBAD),HL
TM90 LD HL,(TMPASS)
         CALL DEC16
         LD HL,PASSCOM
         CALL MSG
         LD HL,(TMBAD)
         CALL DEC16
         LD HL,BADPASS
         CALL MSG
         CALL CRLF
         POP HL
         POP HL
         LD A,(ZIP)
         INC A
         LD (ZIP),A
         JP TM2
ZIP  DEFB 0
CUP  DEFW 0
TIME DEFB 0
BADPASS " BAD PASSES"
         DEFB 0
PASSCOM " PASSES "
         "COMPLETE "
         DEFB 0
READAS " READ "
         "AS "
         DEFB 0
READING "READING"
         DEFB 0
WRITING "WRITING"
         DEFB 0
```

```
TMX POP HL
        POP HL
        POP HL
        POP HL
        OR A
        RET
TMPASS DEFW Ø
TMBAD DEFW Ø
TMBADF DEFB Ø

*
*INPUT
INP LD A,(HL)
        CP "P"
        JR Z,INP1

*NOT INP SO GIVE IT TO MICRODOZ
        DEC HL
        DEC HL
        JP DODOS
INP1 INC HL
        CALL BYTE
        RET C
        LD C,E          *C= PORT NUMBER
        LD B,E
        IN L,(C)
        LD H,Ø
        CALL HEX8
        CALL CRLF
        OR A
        RET

*
*OUTPUT PORT,VALUE
OUT LD A,(HL)
        CP "T"
        JR Z,OUT1

*NOT OUT GIVE IT TO MICRODOZ
        DEC HL
        DEC HL
        JP DODOS
OUT1 CALL CRLF
        INC HL
        CALL SKIPSP
        CALL BYTE       *GET PORT NUMBER
        RET C
        PUSH DE
        CALL SKIPSP
        CP 22H
        JR NZ,OUT2
        INC HL
        LD A,(HL)       *QUOTED VALUE
```

```
OUT3 POP BC
        OUT (C),A
        CALL CRLF
        OR A
        RET
OUT2 CALL BYTE          *GET HEX OR DECIMAL BYTE VALUE
        LD A,E
        POP DE
        RET C
        PUSH DE
        JR OUT3

*IBUFF
BUFFER DEFS 80          *MAIN INPUT BUFFER
        DEFB 0DH
BUFFER1 DEFS 80         *EDITOR BUFFER
        DEFB 0DH
B1LEN DEFB 0
B1POINT DEFB 0
IBUFF LD HL,BUFFER
        LD DE,80

*D=CHARS INPUT E= SPACE REMAINING
IBUFF1 CALL INPUT
        CP 32
        JR C,CCODES
*NOT A CONTROL CODE
        JR NZ,IBUFF2
        LD A,(IBFLAG)

*IBFLAG<>0 THEN SPACE ACTS LIKE CARRIAGE RETURN
        OR A
        LD A,32
        JR NZ,CR
IBUFF2 CP 95
        JR Z,BKSP

*PUT CHARACTER IN BUFFER
        LD (HL),A
        INC HL
        LD B,A
        CALL OUTPUT
        LD A,(CYFLG)
        OR A
        CALL Z,INX
        INC D
        DEC E
        JR NZ,IBUFF1
        SCF
        RET
```

```
*PARSE CONTROL CODES
CCODES CP 8
        JR Z,BKSP
        CP 11H
        JR Z,BKSP
        CP ØDH
        JR Z,CR
        CP 3
        JP Z,BEGINØ
        CP 1
        JR Z,CONTA
        CP 7
        JR Z,CONTG
        CP ØEH
        JP Z,CONTN
        CP 19H
        JP Z,CONTY
        CP 1AH
        JP Z,CONTZ
        CP 4
        JP Z,CONTD
        JR IBUFF1

*BACKSPACE
BKSP LD A,D
        OR A
        JR Z,IBUFF1
        PUSH HL
        LD HL,BKSPSTR
        CALL MSG
        POP HL
        DEC HL
        LD A,(B1POINT)
        DEC A
        LD (B1POINT),A
        INC E
        DEC D
        JR IBUFF1

*CARRIAGE RETURN END OF INPUT
CR LD (HL),A
        CALL CRLF
        LD A,D
        OR A
        JR Z,CR1
        LD (B1LEN),A
        LD HL,BUFFER
        LD DE,BUFFER1
        LD C,A
        LD B,Ø
        LDIR
CR1 XOR A
        LD (B1POINT),A
        LD HL,BUFFER
        RET
```

```
*CONTROLA (BUFFER)=(EDITBUFFER)
CONTA CALL CKEND
        JR C,CONTA1
BELL LD B,7
        CALL OUTPUT
        JP IBUFF1
CONTA1 CALL BPOINT
        JP IBUFF2
CONTG CALL CKEND
        JR NC,BELL
CONTG1 CALL BPOINT
        LD (HL),A
        INC HL
        LD B,A
        CALL OUTPUT
        CALL INX
        INC D
        DEC E
        SCF
        RET Z
        CALL CKEND
        JR C,CONTG1
        JP IBUFF1
CONTN LD B,64
        CALL OUTPUT
        CALL CRLF
        CALL CR
        LD DE,80
        JP IBUFF1
CONTZ CALL CKEND
        JR NC,BELL
CONTZ1 LD B,"%"
        CALL OUTPUT
        LD A,(B1POINT)
        INC A
        LD (B1POINT),A
        JP IBUFF1
INX LD A,(B1POINT)
        INC A
        LD (B1POINT),A
        RET
CKEND LD BC,(B1LEN)
        LD A,B
        CP C
        RET
BPOINT LD A,(B1POINT)
        LD IX,BUFFER1
        LD C,A
        LD B,0
        ADD IX,BC
        LD A,(IX)
        RET
CYFLG DEFB 0
```

```
CONTY LD A,(CYFLG)
        OR A
        JR Z,INSERT
        XOR A
        LD (CYFLG),A
        LD B,">"
        CALL OUTPUT
        JP IBUFF1
INSERT DEC A
        LD (CYFLG),A
        LD B,"<"
        CALL OUTPUT
        JP IBUFF1
CONTD PUSH DE
        PUSH HL
CONTD1 CALL INPUT
        CP 32
        JR NC,CONTD2
        POP HL
        POP DE
        JP IBUFF1
CONTD2 LD (CHAR),A
CONTD5 CALL CKEND
        JR C,CONTD3
        POP HL
        POP DE
        JP BELL
CONTD3 CALL BPOINT
        LD B,A
        LD A,(CHAR)
        CP B
        JR NZ,CONTD4
        POP BC
        POP BC
        JP IBUFF1
CONTD4 LD (HL),B
        INC HL
        CALL OUTPUT
        CALL INX
        INC D
        DEC E
        JR NZ,CONTD5
        POP HL
        POP DE
        SCF
        RET
CHAR DEFB Ø
        LD DE,8Ø
        JP IBUFF1
CONTZ CALL CKEND
        JR NC,BELL
```

```
CONTZ1 LD B,"%"
        CALL OUTPUT
        LD A,(B1POINT)
        INC A
        LD (B1POINT),A
        JP IBUFF1
INX LD A,(B1POINT)
        INC A
        LD (B1POINT),A
        RET
CKEND LD BC,(B1LEN)
        LD A,B
        CP C
        RET
BPOINT LD A,(B1POINT)
        LD IX,BUFFER1
        LD C,A
        LD B,Ø
        ADD IX,BC
        LD A,(IX)
        RET
CYFLG DEFB Ø
CONTY LD A,(CYFLG)
        OR A
        JR Z,INSERT
        XOR A
        LD (CYFLG),A
        LD B,">"
        CALL OUTPUT
        JP IBUFF1
INSERT DEC A
        LD (CYFLG),A
        LD B,"<"
        CALL OUTPUT
        JP IBUFF1
CONTD PUSH DE
        PUSH HL
CONTD1 CALL INPUT
        CP 32
        JR NC,CONTD2
        POP HL
        POP DE
        JP IBUFF1
CONTD2 LD (CHAR),A
CONTD5 CALL CKEND
        JR C,CONTD3
        POP HL
        POP DE
        JP BELL
```

```
CONTD3 CALL BPOINT
        LD B,A
        LD A,(CHAR)
        CP B
        JR NZ,CONTD4
        POP BC
        POP BC
        JP IBUFF1
CONTD4 LD (HL),B
        INC HL
        CALL OUTPUT
        CALL INX
        INC D
        DEC E
        JR NZ,CONTD5
        POP HL
        POP DE
        SCF
        RET
CHAR DEFB Ø
```