NORTH STAR COMPUTERS, INC.

2465 FOURTH STREE
BERKELEY, CA 9471
TEL: 415-549-085

North Star MICRO DISK SYSTEM UPDATE NOTICE

February 23, 1977

This note is to inform you of the availability of Release 2 versions of the DOS, BASIC, and documentation for the North Star MICRO DISK SYSTEM. All systems sent prior to 2/23/77 did not receive any of these updates.

BASIC
Several errors have been found in Release 1 BASIC, including:

1. The OUT statement does not work.

2. The GOSUB statement error message may give the wrong line number.

3. The null PRINT statement will not print a carriage return if there is an additional statement on the same line.

4. The NOENDMARK feature does not work.

5. Certain line numbers (e.g., 13 mod 256), when appearring in a BASIC program, can cause undesired errors to occur.

6. The random number sequence is not very random.

7. Control-C interruption of program execution can not be continued if the iterrupted statement was an INPUT or an OUTPUT statement.

8. The SAVE command does not properly update the file directory entry of the saved file. This can cause errors in some cases.

All of the above problems, plus a few others, have been corrected in Release 2 BASIC. Furthermore, the following improvements have been made:

1. The random number generator can now be re-initialized to start at any place in the pseudo-random sequence. RND(Ø) will generate the next random number. RND(X), X>Ø and X<1, will reset the "seed" to X.

2. A newly dimensioned string will now be initialized to a full string of blanks rather than the null string.

3. A vastly improved line editor has been added which permits convenient modification of program statements. See the attached description.

(over)

DOS
Two enhancements have been made to the Release 1 DOS. The DT command now does a substantially better disk test. Also, the read-after-write mode of operation now works properly.

DOCUMENTATION
The three North Star documents (MDS, DOS, and BASIC) have been updated to incorporate the errata and the changes. Also, the documents have been reworked to improve clarity. Some additional checkout steps have been added to the MDS document.

ORDER BLANK

____ Please send a copy of the Release 2 DOS and BASIC on my enclosed diskette. I enclose $5.ØØ.

____ Please send a copy of the Release 2 DOS and BASIC on a new diskette. I enclose $9.5Ø.

____ Please send a copy of the three updated documents. I enclose $7.5Ø.

NAME:

COMPANY:

STREET ADDRESS:

CITY, STATE, ZIP:

Californians please add sales tax.

# MICRO-DISK SYSTEM ERRATTA AND ADDITIONAL INFORMATION

January 21, 1977

1. The holes on the power regulation PC card for mounting the 7805 heat sink may require a slight amount of reaming. Do not over-ream, as the heat sink should fit snugly in the holes for mechanical support.

2. In some cases disk drives may not operate properly if located in the vacinity of TV's, electric motors, or other sources of electrical noise. This may be remedied by moving the disk drive away from the noise source or by installing the disk drive in a grounded cabinet.

3. Identification of PROM's.

| Location | Schematic Label | PROM Label | Standard Label |
|---|---|---|---|
| 7G | PSEL | Sxx-1 | SE8-1, S38-1 |
| 3F | PGML | Lxx-xx-1 | LE8-20-1,L38-20-1 |
| 3E | PGMR | Rxx-xx-1 | RE8-20-1,R38-20-1 |

4. Capacitor C2 should be marked as 6.8uf (not 10uf) on the silk screen legend for the MDS controller board.

5. The following fix must be made to the MDS controller PC board in order to increase the reliability of reading data written using a different drive.

   a. Cut traces to 8E pin 4 and 8E pin 5. These cuts may be made on the solder side. The connection 8E pin 3 to 8E pin 12 to 9F pin 14 should remain.

   b. Add a jumper wire from 8E pin 4 to 9D pin 10.

   c. Add a jumper wire from 8E pin 5 to 1E pin 12.

   d. Modify page 2 of the schematic drawings so that the inputs to the gate 8E are pin 4-BODY/ (instead of TRANS and BIT) and pin 5-BYC3 (instead of WRITE REQ/). Give name "BYC3" to 1E pin 12 (counter output on page 3).

6. A 74LS157 may be substituted for a 74LS257 in location 2F of the controller PC board. Some kits will include a 74LS157 in place of a third 74LS257.

7. The exact values of capacitors on the MDS controller PC board are not critical. Some kits will be shipped with values slightly different than those specified in the instructions. For example, .047uf instead of .05uf.

8. In Appendix 1 of the Disk Operating System manual, there is an error in the comment about the use of registers for the CIN subroutine. Registers B, C, D, E, H, and L must be preserved by your CIN routine. Some sample I/O routines are as follows:

```
CIN    IN Ø             GET KEYBOARD INPUT STATUS
       ANI 1            MASK DOWN TO INPUT STATUS BIT
       JNZ CIN          JUMP IF NO CHAR TYPED YET
       IN 1             KEYBOARD DATA PORT
       ANI 7FH          MASK DOWN TO 7-BIT ASCCI CODE
       RET
COUT   IN Ø             GET STATUS
       ANI 2            MASK DOWN TO OUTPUT STATUS BIT
       JNZ COUT         JUMP IF NOT OK TO OUTPUT CHARACTER
       MOV B,A
       OUT 1            OUTPUT THE CHARACTER
       RET              NOTE THAT CHAR IS IN ACC NOW
CONTC  IN Ø             GET STATUS
       ANI 1            MASK DOWN TO INPUT STATUS BIT
       XRI 1            SET Z FLAG FALSE IF NO CHARACTER TYPED
       RNZ              RETURN WITH Z FALSE IF NO INPUT
       IN 1             READ THE CHARACTER
       ANI 7FH          MASK DOWN TO 7-BIT ASCII
       CPI 3            SET Z IF CHAR IS CONTROL-C
       RET
TINIT  RET              THIS TINIT ROUTINE DOES NOTHING
```

9. The following lines of code should be added to the end of Appendix 1 of the Disk Operating System manual:

```
2Ø2B           *
2Ø2B           *THIS BYTE IS THE "READ-AFTER-WRITE-CHECK" FLAG
2Ø2B ØØ        RWCHK DB Ø
```

1Ø. There are two DOS commands not described in the DOS manual:

CD <source unit #> <destination unit #>
This command will copy the contents of the diskette mounted on the specified source unit to the diskette mounted on the specified destination unit. Note that the 2.5K of RAM immediately following the DOS are required for this command.

CO <unit #>
This command may be used to "compact" the file space on the diskette mounted on the specified unit. Any unused disk space between existing files will be eliminated by moving files toward track Ø. The CO command may be used to reclaim file space when a file is shortened or deleted. Note that this command requires use of the 2.5K RAM area immediately following the DOS.

11. Line number 190 of the last sample program in the Appendix of the BASIC manual should read:

```
190 READ #0%5*X,X2\ REM EACH FP VALUE USES 5 BYTES
```

12. The following errors should be corrected in the assembly manual:

    page 17, step 3: Wave form 4 should read 1D pin 11, 8us.

    page 17, step 3: Wave form 5 should read 2D pin 11, 128us.

    page 19, step 4: The heat sink part number should read 690-3, not 695-3.

    page 23, item 3: The opcode for the LDA instruction should read 3A, not 22.

Please send us feedback about the MICRO-DISK System documentation. We would like to make this documentation as complete, clear, and accurate as possible.

## INTRODUCTION

The North Star Disk Operating System (DOS) was designed and implemented by staff members of North Star Computers, Inc. for use in conjunction with the North Star MICRO DISK SYSTEM. The DOS permits a user to issue various "commands" from a terminal for maintaining and using files on the disk(s). The DOS also provides "library routines" which may be called from user software. These library routines will primarily be of interest to users who will be developing their own system software, as opposed to those users who will primarily use application programs such a BASIC.

The DOS occupies 2.5K (A00 hex) bytes of RAM, including 384 bytes of RAM for user I/O routines. The origin of the DOS is 2000 hex in the standard version.

The North Star DOS is intended for use only with the North Star MICRO DISK SYSTEM, and no license is granted for any other use. Improved copies of the Version 2 DOS, as they become available, may be obtained for a nominal copying charge.

## DISK ADDRESSES

As described in the hardware documentation, information is stored on the disk in 256-byte "blocks". Each diskette consists of 35 concentric "tracks" with 1Ø sector positions per track. A block exists at each sector position. Every block on the disk is identified by a unique "disk address" - an integer from Ø through 349. For example, the block at track 27,sector 3 has disk address 273. Track Ø is the "outermost" track, and track 34 is the "innermost" track.

## FILES

The primary DOS function is to permit the creation, deletion and use of files on disk(s). A file is an integral number of blocks of data with sequential disk addresses. For example, a particular file might occupy disk addresses 17 through 95 on a diskette mounted on unit 2.

The first four blocks on each diskette contain a "file directory" which specifies a symbolic name, base address, length and type information for each file on that diskette. The symbolic name may be up to 8 characters long, and may include any characters except blank and comma. The length of a file may be up to 346 blocks. A directory may contain as many as 64 entries. No two files in a directory may have the same name, but it is possible for files of the same name to be in directories of diskettes mounted simultaneously on separate units in a multi-unit system.

## FILE TYPES

One byte in the file directory entry for each file specifies the "type" of the file. Depending on the specific type, additional bytes in the entry may have special meaning. (The details of file directory entries are given in a later section). Only four of the 256 possible file types have been assigned to date:

type 0 - Default type. All new files are given type 0 until explicitly changed.

type 1 - Machine language program. This file type identifies a machine language program (object code) that may be executed directly from the DOS with the GO command.

type 2 - BASIC program. This type of file is used to identify a BASIC program that can be LOADed or SAVEd from BASIC.

type 3 - BASIC data file. This type of file may be read and written by BASIC programs for data storage and retrieval.

## COMMANDS

Instructions are issued to the DOS by typing "commands". The command format is a 2-letter mnemonic followed by any required arguments. Arguments are separated from the command mnemonic and from each other by a single blank. A command must be terminated by a carriage return before the DOS takes any action. If a typing error occurs during typing of a command, a question mark (?) may be typed to permit re-typing of the command.

When a file name is required as a command argument, the disk unit number (in a multi-unit system) may be specified by immediately following the file name with ",1", ",2" or ",3". Otherwise, unit 1 is assumed. Some sample name formats are:

```
ABC    TEST1234,3    BASIC,1
```

Commands may be typed whenever the prompt character (*) appears at the left margin of the terminal.

LI <optional unit #>
This command will list the entire contents of the directory on the diskette mounted on the specified unit. If no unit is specified, then unit 1 is assumed. For each file, its symbolic name, starting disk address, length and type will be printed. To prematurely terminate a listing, a control-C may be typed.

CR <file name> <length> <optional start address>
This command will create a new file on the unit indicated by the file name. The length argument specifies the number of 256-byte blocks. If no starting address is given, then the file will start after the "last" (innermost) file currently allocated on the diskette. Otherwise, the supplied starting address will be used. This command will only create a file directory entry - no accessing of the file itself will be done.

DE <file name>
This command will delete an existing file directory entry on the indicated unit. No actual accessing of the file blocks will be done.

TY <file name> <file type>
This command is used to change the type of the specified file on the indicated unit.

GO <file name>
This command is used to load the specified file into RAM from the indicated unit and begin execution. The GO command may be used only with type 1 files which have correctly been given a

"go-address" (see the GA command below). The GO command will read the entire file into RAM beginning at the go-address, and then jump to the go-address. Obviously, the first byte of the file must be the entry point of the program.

GA <file name> <hex RAM address>

The GA command is used to specify the "go-address" for a type 1 file. An attempt to do a GO to a file which has not had the go-address properly set can have undesireable effects.

JP <hex RAM address>

This command will cause the computer to jump to the specified RAM address. It provides a way of executing programs which exist in the address space of the computer. Do not confuse this command with the GO command.

LF <file name> <hex RAM address>

SF <file name> <hex RAM address>

These commands may be used to load or save a file. The entire contents of the file will be read or written to or from the specified RAM address.

CF <source file name> <destination file name>

This command may be used to copy one file to another. The two files may be on the same or separate units. The file copy is performed only if the destination file is at least as large as the source file. The file type and the type dependent information are also copied.

RD <disk address> <hex RAM address> <# of blocks>

WR <disk address> <hex RAM address> <# of blocks>

These commands may be used to read or write a specified unit directly to or from RAM. The WR and RD commands should be used with great care, as typing errors can have catastrophic effects. The disk address may optionally be followed by ",1", ",2" or ",3" to indicate a particular unit. Otherwise, unit 1 is assumed. Note that a method of copying one diskette to another in a single drive system would involve repeated use of the RD and WR commands.

IN <optional unit #>

This command should be used to initialize each new diskette to be used in the system. The IN command writes each block on the specified drive with ASCII blank characters (20 hex). This initializes the directory and also guarantees that no "hard disk error" can result from access to an uninitialized file block. The IN command takes about 15 seconds. Needless to say, one should make sure that the proper diskette is mounted before issuing the IN command. Note that the IN command, in order to drive the disk at high speed, uses the 2.5k RAM area immediately following the DOS. Also note that an initialized diskette does not contain a copy of the DOS.

DT <optional unit #>

The DT command may be used to test the unit or to verify the usability of a diskette. This command will continuously write a changing pattern and then read the diskette on the specified unit. Note that all information previously stored on the diskette will be overwritten, and that a tested diskette must be initialized before use. If a hard disk error occurs, then the test will stop and print out the hard disk error message. The command may be stopped by typing a control-C. Note that the 2.5K block of memory immediately following the DOS will be used for this command.

CD <source unit #> <destination unit #>

This command will copy the contents of the diskette mounted on the specified source unit to the diskette mounted on the specified destination unit. Note that the 2.5K of RAM immediately following the DOS are required for this command.

CO <unit #>

This command may be used to "compact" the file space on the diskette mounted on the specified unit. Any unused disk space between existing files will be eliminated by moving files toward track Ø. The CO command may be used to reclaim file space when a file is shortened or deleted. Note that this command requires use of the 2.5K RAM area immediately following the DOS.

## DISK SYSTEM START-UP

After power-on, or when it is desired to re-start the disk system, the 8080 or Z80 computer must be forced to begin execution at the PROM bootstrap program starting address (E900 hex in the standard version). The PROM bootstrap program will read one 256-byte block from unit 1, disk address 4 into RAM at the DOS starting address (2000 hex in the standard version). After reading in the block, the bootstrap will branch to the DOS starting address. The program in the first block of the DOS will proceed to read in the nine blocks from disk addresses 5 through 13. Then the DOS will print the prompt character (*) and await a command from the terminal.

Once the DOS has been started, it is no longer necessary to leave the diskette in unit 1. The DOS is fully resident in RAM, and makes no disk accesses unless asked to do so. Furthermore, the DOS does not maintain any copies of the diskette file directory in RAM between commands. Thus it is possible, for example, to obtain listings of the file directories of several diskettes by inserting them one at a time and then issuing the LI command.

In a multi-drive system, after disk system start-up, the first access to an additional drive must be to track 0. This will happen normally if files are accessed, because the file directory is maintained on track 0.

## PERSONALIZING YOUR VERSION OF THE DOS

The following procedure must be followed the first time you operate the DOS after installing it in your computer system. After you have followed this procedure, the DOS will communicate directly with your terminal immediately after disk system start-up.

The DOS is designed to be able to interface to any conceivable terminal I/O conventions. There are four routines used by the DOS: character input (CIN), character output (COUT), control-C detect (CONTC), and terminal initialization (TINIT). In the DOS which you receive with your MICRO DISK SYSTEM, each of these routines is merely a jump to self loop. The location of these routines is shown in Appendix 1. Thus, when you first perform a disk system start-up sequence, the DOS will be stuck in a branch to self loop at TINIT.

At this point, remove the pre-recorded diskette and insert the second diskette supplied. Now stop the computer and enter your own terminal I/O subroutines in the last 384 bytes of the DOS (from 2880 hex through 29FF hex in the standard version), carefully following the interfacing rules described in Appendix 1. (Unused memory locations in this region can be used to contain any code or data that you wish to have loaded as part of the disk start-up sequence.) Next, patch the four JMP intructions to contain the addresses of your routines.

Now, force your computer to branch to TINIT. The terminal should print out an asterisk (*) and the DOS should be awaiting a command.

Now, initialize the second diskette with the IN command. Be sure the second diskette, and not the pre-recorded diskette is properly inserted in unit 1.

```
*IN 1
```

Next create a file with the name DOS. This will discourage your later allocating a file on top of the disk space that will hold the DOS.

```
*CR DOS,1 10
```

Now write out the DOS from RAM (2000 hex in the standard version) to disk unit 1.

```
*SF DOS 2000
```

You should now be able to start the DOS by branching to the PROM bootstrap start address.

## PERSONALIZING YOUR VERSION OF BASIC

When you have successfully created your personal version of the DOS on the second diskette, you may proceed to creating your personal version of BASIC on the second diskette. First, insert the pre-recorded diskette in unit 1, and read BASIC into RAM at the location where it is intended to be run (2A00 hex in the standard version).

```
*LF BASIC 2A00
```

Now remove the pre-recorded diskette and insert the second diskette. Create an entry in the file directory for BASIC, set the type and set the go-address:

```
*CR BASIC,1 40
*TY BASIC,1 1
*GA BASIC,1 2A00
```

The region in RAM where BASIC allocates user BASIC programs and data is set up in the BASIC initialization sequence (see Appendix 2). If you have a non-standard version of BASIC or you wish to change the region where BASIC allocates programs and data, then you must modify the appropriate LXI instructions in the BASIC software. If you decide to make such modifications, stop your computer at this point and make the appropriate modifications to the copy of BASIC now in RAM. Then re-start the DOS by branching to the bootstrap address. Whether or not you made the above modification, now write BASIC out onto the second diskette:

```
*SF BASIC 2A00
```

It should now be possible to start BASIC by typing

```
*GO BASIC
```

Note that the terminal requirements of BASIC are handled by calling the DOS terminal I/O routines.

## DISK ERRORS

Every disk or verify operation is tried 10 times by the DOS before reporting failure. After the 10 tries, the disk address is printed followed by the message "HD?", and the DOS will await further commands. For example,

```
2 233hd?
*
```

informs of a disk error on unit 2, at track 23, sector 3.

If the operation is a write, a disk error will result if the diskette was write protected.

It is possible to specify to the DOS that after every write operation performed, an attempt is to be made to verify the written data against the data in RAM. This modification will result in slower operation, and most users should find that it is not needed. To make the modification, go through the procedure outlined above for creating a version of the DOS, and change the RWCHK byte (see appendix) from 0 to 1.

## FILE DIRECTORY STRUCTURE

This section gives a detailed description of the format of entries in the file directory on a diskette. The file directory occupies disk addresses 0 though 3, with each of these four blocks holding sixteen 16-byte entries. The symbolic name of the entry is the first 8 bytes of an entry. An empty entry is an entry with 8 blanks (20 hex). Following the symbolic name in an entry, the disk address (2 bytes), the file size (two bytes) and the type (1 byte) follow. The last three bytes of an entry are type dependent. In particular, for a type 1 file (GO file), the two bytes following the type byte contain the go-address, and for a type 2 file (BASIC program) the byte following the type byte specifies how many blocks of the file actually contain valid data.

File directory entry:

| | |
|---|---|
| bytes 0-7 | symbolic name of entry |
| bytes 8-9 | disk address |
| bytes 10-11 | number of blocks in file |
| byte 12 | file type |
| bytes 13-15 | type-dependent information |

## DOS LIBRARY ROUTINES

This section describes how user machine language software may interface to the DOS for the accessing of disk files.

Appendix 1 shows the entry points for each of the routines to be described here. The exact interfacing requirements are described in the appendix. The DOS uses the stack pointer existent at call time, and some of the DOS library routines may require as much as 30 bytes of stack storage. Note that the DOS may be re-entered without using the bootstrap PROM. Now follows a discussion of each library routine.

### HDERR

This routine is unique among the DOS library routines because it does not return. HDERR branches to DOS code that prints an error message and then enters the DOS command processor. HDERR is branched to within the DOS whenever a read attempt is impossible to successfully complete after 10 retries. If your software wishes to retain control in the event of a hard disk error, your software should modify the address of the HDERR JMP instruction (e.g., LXI H,ADDR; SHLD HDERR+1). The stack depth at the time of a branch to HDERR from within the DOS is indeterminate. [Note: Software for dealing with hard disk errors is notoriously difficult. It is suggested that due to the expected low frequency of hard disk errors, for most applications the existing HDERR action will be sufficient. Hard disk errors will result primarily from careless use (e.g. forgetting to initialize a diskette, or from removing a diskette while writing is in progress). Hard disk errors can also result from power failure during writing, or from a hardware system failure.]

### DLOOK

This routine searches for a specified file name in the directory of the indicated disk unit. On failure, HL is set to the value of the first free disk address on the indicated unit following the last file on the diskette.

On success, HL contains a pointer into a buffer in DOS RAM that has a copy of the sought entry. The pointer addresses the first byte following the symbolic name (i.e., byte 8). Also, on return, the ACC specifies the disk unit which was determined from the name passed as argument.

### DWRIT

This routine is used to write back to diskette an updated file directory entry which was previously found using DLOOK. No disk activity may occur between the DLOOK and the DWRIT call.

### DCOM

This routine may be used to issue an arbitrary disk read or write command. On a read request, DCOM will try 10 times for a successful read before giving up and branching to HDERR. DCOM will fail return if the supplied arguments are out of bounds. However, great care should be used before calling DCOM with incorrect arguments.

LIST

This routine will list the file directory of the specified drive. The listing format will be exactly the same as the listing format obtained with the DOS LI command.

# Appendix 1: DOS I/O Routines and Entry Points

```
ØØØØ          *
ØØØØ          *NORTH STAR DISK OPERATING SYSTEM
ØØØØ          *
ØØØØ                ORG 2ØØØH         STANDARD VERSION ORIGIN VALUE
2ØØØ                DS 13             THESE CELLS ARE RESERVED
2ØØD          *
2ØØD          *
2ØØD          *THIS IS THE CHARACTER OUTPUT ROUTINE
2ØØD          *THE CHARACTER TO BE OUTPUT MUST BE IN THE B REGISTER
2ØØD          *ON RETURN THE CHARACTER MUST ALSO BE IN THE ACC
2ØØD          *NO OTHER REGISTERS MAY BE MODIFIED
2ØØD C3ØD2Ø   COUT  JMP COUT          YOUR ROUTINE MUST DO A RET
2Ø1Ø          *
2Ø1Ø          *THIS IS THE CHARACTER IN ROUTINE
2Ø1Ø          *THE 7-BIT ASCII CODE MUST BE RETURNED IN ACC
2Ø1Ø          *ALL OTHER REGISTERS MAY BE USED
2Ø1Ø C31Ø2Ø   CIN   JMP CIN           YOUR ROUTINE MUST DO A RET
2Ø13          *
2Ø13          *THIS IS THE TERMIAL INITIALIZATION ROUTINE
2Ø13          *ALL REGISTERS MAY BE USED
2Ø13          *IF NOT NEEDED, MERELY PATCH IN A RET
2Ø13 C3132Ø   TINIT JMP TINIT
2Ø16          *
2Ø16          *THIS ROUTINE DETECTS A CONTROL-C
2Ø16          *IF Z IS SET ON RETURN, THAT MEANS A CONTROL-C WAS TYPED
2Ø16          *ROUTINE SHOULD RETURN IMMEDIATELY IF NO CHAR WAS TYPED
2Ø16          *ALL REGISTERS MAY BE USED
2Ø16 C3162Ø   CONTC JMP CONTC
2Ø19          *
```

```
2019          *DOS LIBRARY ROUTINE ENTRY POINTS
2019          *
2019          *THIS ADDRESS IS BRANCHED TO ON HARD DISK ERRORS
2019 C30000   HDERR JMP 0        0 IS NOT THE REAL ADDRESS
201C          *
201C          *THIS IS THE FILE DIRECTORY LOOKUP ROUTINE
201C          *ACC MUST CONTAIN THE DEFAULT UNIT NUMBER (NORMALLY 1)
201C          *HL=POINTER TO RAM WHERE THE FILE NAME MUST APPEAR
201C          *FOLLOWED BY EITHER A BLANK OR CARRIAGE RETURN.
201C          *FAILURE IF CARRY SET. ON FAILURE, IF THE ACC=0
201C          *FAILURE IS BECAUSE THE NAME WAS OF BAD SYNTAX.
201C          *IF THE ACC#0, THEN NAME WAS NOT FOUND IN DIRECTORY, AND
201C          *HL=THE ADDRESS OF THE FIRST FREE DISK BLOCK FOLLOWING
201C          *THE "LAST" FILE ON THE DISK.
201C          *ON SUCCESS, ACC=THE DISK UNIT INDICATED, AND HL HAS A POINTER
201C          *TO THE EIGHTH BYTE OF A COPY OF THE ENTRY IN DOS RAM
201C C30000   DLOOK JMP 0        0 IS NOT THE REAL ADDRESS
201F          *
201F          *THIS ROUTINE WILL WRITE A DIRECTORY ENTRY BACK TO DISK
201F          *NO ARGS ARE NEEDED. MUST FOLLOW DLOOK.
201F C30000   DWRIT JMP 0        0 IS NOT THE REAL ADDRESS
2022          *
2022          *THIS ROUTINE MAY BE USED TO ISSUE A DISK COMMAND
2022          *ACC=NUMBER OF BLOCKS
2022          *B=COMMAND (0=WRITE, 1=READ, 2=VERIFY), C=UNIT NUMBER
2022          *DE=STARTING RAM ADDRESS, HL=STARTING DISK ADDRESS
2022          *RETURN WITH CARRY SET MEANS ARGUMENTS WERE ILLEGAL
2022 C30000   DCOM  JMP 0        0 IS NOT THE REAL ADDRESS
2025          *
2025          *THIS ROUTINE MAY BE USED TO LIST A FILE DIRECTORY
2025          *ACC=DISK UNIT
2025 C30000   LIST  JMP 0        0 IS NOT THE REAL ADDRESS
2028          *
2028          *THIS ADDRESS IS AN ENTRY POINT TO THE LOADED DOS
2028 C30000   DOS   JMP 0        0 IS NOT THE REAL ADDRESS
202B          *
202B          *
202B          *
202B          *
202B          *
202B          *
202B          *
202B          *

202B          *
202B          *THIS BYTE IS THE "READ-AFTER-WRITE-CHECK" FLAG
202B 00       RWCHK DB 0
```

# Appendix 2: BASIC Entry Points

```
0000            *
0000            *NORTH STAR BASIC, VERSION 6
0000            *
0000                  ORG 2A00H        STANDARD VERSION ORIGIN
2A00            *
2A00 AF         EP0   XRA A            INITIALIZATION ENTRY POINT
2A01 C3052A           JMP EP11
2A04 37         EP1   STC              CONTINUE ENTRY POINT
2A05 210000     EP11  LXI H,ENDBAS     FIRST CELL OF PRAGRAM REGION
2A08 11FF4F           LXI D,4FFFH      LAST CELL OF CONTIGUOUS MEMORY
2A0B C30000           JMP START        ENDBAS AND START ARE NOT REALLY 0
2A0E            *
```

# Appendix 2: The Line Editor

This section describes an advanced method of editting BASIC program lines. For the purpose of this discussion, the term "old line" will refer to a line of the BASIC program which is to be editted. The EDIT command may be used to designate which program line is to become the old line, e.g.,

```
          EDIT 100
```

Otherwise, the most recently typed-in program line is automatically designated the old line.

When typing in a "new line" to BASIC, the old line may be used as a "template" for creating the new line. Special editting characters may be typed to cause parts of the old line to be used in contructing the new line. Use of the line editor will result in dramatically more convenient program modification.

When a new line is entered with the assistance of the line editor, it is as if the new line was entered in the "normal" manner. Thus a new line can have either the same or a different line number as the old line. For example, if the new-line line number is the same as the old line, then the new line will replace the old line in the program.

A line edit is terminated as soon as a carriage return is typed. If an illegal editting command is attempted, the line editor will ring a bell and perform no action.

During the entering of a new line, there are two "invisible pointers" maintained, one which locates a character position in the old line, and one which locates a character position in the new line. When beginning to enter a new line to BASIC, both pointers locate the first character position. Typing a normal (non-editting) character will advance both pointers by one position. The editting character control-G will cause the characters in the old line to the right of the old line pointer to be printed as part of the new line. Thus, for example, if the line

```
              100 PRINT "**************
```

was entered as part of a program, and the missing end quote mark was discovered, the error could be corrected by typing EDIT 100, then control-G, then a quote mark, and then a carriage return. If it was desired to make line 234 do the same PRINT statement, then the following could be typed to accomplish this: 234 followed by control-G followed by a carriage return. It is suggested that you try these and similar examples before reading on. Now follows descriptions of each of the editting control characters.

Control-G Copy rest of old line to end of new line.
This command will print the characters from the current pointer position in the old line as part of the new line.

Control-A Copy one character from old line.
This command will print one character from the current pointer position of the old line as part of the new line. Both pointers will be advanced one character position.

Control-Q Back up one character.
This command will erase the last character of the new line, and also back up the old line and new line pointers. A left-arrow (under-line on some terminals) will be typed to indicate that this command was typed.

Control-Z Erase one character from old line.
This command advances the old line pointer by one character position. This command would be used to remove undesired characters from the old line. A per cent character (%) is printed to indicate that this command was typed.

Control-D Copy up to specified character.
This command requires a second character to be typed before it is executed. The command will copy the contents of the old line up to, but not including the first occurence of the specified character to the new line.

Control-Y Toggle insert mode.
When entering a new line, insert mode is "off". When insert mode is off, then typing normal characters will advance the old line pointer. When insert mode is "on", then typing normal characters will not advance the old line pointer. Thus, insert mode may be used to add some omitted characters from the old line. A left angle bracket will be typed to indicate entering insert mode, and a right angle bracket (>) will be typed to indicate leaving insert mode.

Control-N Re-edit new line.
This command erases the current new line and permits re-entering the new line. The partially complete new line becomes the old line for subsequent editting. (Of course, the EDIT command could be typed to select a different old line.) An at-sign (@) is typed to indicate that this command was typed.

→ <u>Routine</u> for loading large machine language programs from disk

<u>Problem</u>: DOS starts at 8K. Program running more than 8K will wipe out DOS before it finishes.

<u>Approach</u>: Use 16-32K as a buffer storage. A bootstrap is loader into the first 256 bit block; the program in the succeeding. The entire file is called and a jump is made to the 16K address. The program is then down-loaded to 0K and a jump is made to that address (0).

<u>File creation</u>

```
*CR   NAME  <LENGTH IN 256 bit blocks)
* TY   NAME   1
* GA   NAME  < 16K starting location: 4ØØØ>
```

Can then  * GO NAME  and load will start

- 1st block of buffer is to be bootstrap. Assume originally it is toggled in starting at ØØØØ. We want to load it into the 1st block as follows:

```
*WR  < beginning address of file> <ØØØØ> <1>
```

- The program to be moved is assumed initially loaded at 0, but then moved to 16K (4ØØØ). The latter is done by a file called 16KMOVE. Once there it is loaded into the file starting at the second block:

```
* WR < beginning address of file + 1> < 4ØØØ > <63>
```

- The 16KMOVE file is located at the first 256 block in the 16 K block, or 4ØØØ. This program is simply loaded into position, and a forced jump is made ( so that the program to be moved can be loaded into ØØØØ ). Once this program is in place, the program to be moved to 16K is loaded (cassette; it may wipe out DOS), and the jump from is made. DOS is reloaded and the program moved to file.

The two programs to be keyed in initially are

16K MOVE
and the bootstrap for NAME. They are very similar as they are the inverses of one another.

## 16K MOVE

Starting address 4000 or 100,000
File Creation:

```
    * CR... 16KMOVE 1
    * TY 16KMOVE 1
    * GA 16KMOVE 4000
```

Can then

```
    * LE 16KMOVE 4000   or   * GO 16KMOVE
```

16KMOVE is initially loaded using

```
    * SF 16KMOVE 4000
```

Machine language listing of 16KMOVE

```
100,000   041   LXI   load H&L with 16K address
    001   000
    002   101
    003   001   LXI   load B&C with 0K address
    004   000
    005   000
    006   174   MOVE H to A
    007   346    ANI   200   (test for 32K boundary)
    010   200
    011   302   JNZ  -  jump if 16K loaded  (jump to tight loo
    012   023
    013   100
    014   012   LDAX  load accumulator with data at B&C addres
    015   167   MOV   move data to M  (H&L address)
    016   003   INX   B&C
    017   043   INX   H&L
    020   303   Jump  to H register test
    021   006
    022   100
```

```
023     303    Loop when finished
024     023
025     100
```

At this point, DOS may have to be reloaded.

<u>Bootstrap</u>

moves data down from second 256 bit block

```
                   100,000    041     load H&L
                       001    000
                       002    101
                       003    001    Load  B&C
                       004    000
                       005    000
                       006    174    MOVE H to A
                       007    346    ANI  200
                       010    200
                       011    302    JNZ
                       012    023
                       013    100
different  →           014    176    MOV   Load A with data from H&L address
from 16K MOVE →        015    002    STAX  Store A at B&C address
                       016    003     INX   B&C
                       017    043     INX   H&L
                       020    303     Jump to H&L tst
                       021    006
                       022    100
                       023    303    Go     when finished
                       024    000
                       025    000
```

Can save bootstrap via

```
* SF NAME 4000
```