TABLE OF CONTENTS

LIST OF TABLES

1.0 INTRODUCTION

This report is a brief review of the North Star floppy (micro-) disk system. This device, including its attending software, very nicely matches the needs of a small (less than $4000) microcomputer system.

The discussion is divided into several progressive sections. The following section deals with hardware aspects of the North Star disk system, along with some spot comparisons with other products. The next section provides a user-oriented description of North Star's very adequate disk operating system software. This software is capable of accommodating the needs of most small system users. Also included is a description of how to "patch" in personalized hardware, with the MITS 88-SIO as a specific example. The next section considers North Star's BASIC software. This software has both good and bad features, along with at least one major glitch which should have been caught before distribution.

2.0 NORTH STAR DISK HARDWARE

It is probably generally true that in the development of a small micro- (and perhaps mini-) computer system the first mass storage device is either a PROM, a magnetic tape cassette recorder or a paper tape reader/punch, or all simultaneously. The next step up is a floppy disk, the micro-disk representing the low end of the improvement.

The North Star floppy disk hardware appears to have good features for the price. For example, PERTEC Computer Corporation (owner of MITS, Inc.) advertises a disk drive at twice the retail price of the North Star and emphasizes that their new micro-disk system has a head disengage feature which breaks contact after 5 seconds of no disk I/O*. The North Star hardware also has this important property but they do not advertise it as such a feature is taken for granted by professionals. Generally, most microcomputer systems operate with the head in contact with the diskette only perhaps 1% of the time**, and an automatic disengage feature is a must.

As a comparative example of the ability of the North Star system, we may compare it with the Sykes floppy disk (not a micro) system which is

*The manual presently states 6.4 seconds.
**For the North star drive, MTBF is 8000 hours, assuming 25% motor duty cycle. Medium life is said to be 3 million passes, or roughly the drive life.

used with our PDP-8. At many times the North Star price, the Sykes disk has less than twice the data (bit) transfer rate and storage capacity of the North Star. Two North Star's would have been a much better buy than one Sykes, assuming the North Star disk accesses could be interleaved while maintaining the maximum access rate of either disk.

As another comparison point, we also have a PERTEC floppy disk unit (not their newly announced micro-disk which they call a "mini") connected to an Altair 8800a[R]. It took some time to get this system up and running (as opposed to the North Star system which went without a hitch). The final snag encountered was the memory size (in bytes) required to just get the system going: 24K*, in comparison to the North Star requirement of 12.5K. The PERTEC disk system costs more than twice that of the North Star, while also requiring an extra 8K of memory.

Although North Star does not have a large reputation, they are sure to grow. The quality of their product is quite good. One way they could improve their position is to develop a large variety of software to accompany their hardware system. An example might be a BASIC compiler (instead of interpreter) option. We presently have a FORTRAN compiler on the North Star system.

*The newly advertised PERTEC microdisk system is said to require 20K of memory, including BASIC.

3.0 ASSEMBLING A NORTH STAR DISK SYSTEM

The North Star disk drive (manufactured by a Xerox subsidiary, Shugart) is controlled by a printed circuit board which fits into a S-100 bus computer. Included on the controller board are three preprogrammed fusible-link (titanium-tungsten) ROM's. This on-board program is required for the bootstrap loader which starts the system up. Such is not the case with the more expensive PERTEC disk system which has an optional 1702 PROM bootstrap, requiring an additional board.

Included in the standard hardware documentation* is a checkout procedure for the controller board which largely consists of looking for pulses of the right shape and repetition rate. However, there is no hardware trouble-shooting guide to handle the situation in which a failure occurs in the checkout procedure! However, we have had no failures to date.

The power requirements for the controller board are taken care of by the computer bus supply. The disk drive requires +5 and +12 volts for operation. In principle, these could also be obtained from the computer bus. In particular, the +5-volt demand in less than one

*North Star optionally supplies additional hardware and documentation for disk drive head alignment, timing and general repair.

ampere, and is thus no problem. However, the +12-volt demand is between one and two amperes, which is a little high. North Star offers the option of no power supply, a regulator board which accepts unregulated +8 and +18 volts, or a complete power supply.

Before hooking everything up, some board jumper changes are required in the eventuality that more than one drive is to be driven by the one controller board. Up to three drives can be connected in parallel and controlled. Also, it appears that multiple controller cards can be used to handle more disks with only minor modifications of hardware and software. These alterations are well laid out in the manual and are simple.

Connecting the controller to the disk drive P.C. board is relatively simple. A nice 4-conductor ribbon cable is supplied with already installed end connectors. The directions for plugging into the controller board are crystal clear. However, the connection to the disk drive board is a little confusing. North Star's directions are technically correct, but not immediately interpretable. As an aid in choosing the correct one of two orientation possibilities, note that if the drive unit is oriented with the drive board at the top, the cable connection will be such that the cable will want to naturally (no bends in the ribbon) come out the bottom of the drive. However, there are obstacles to bringing the cable neatly out the

bottom of the unit and the ribbon ends up getting bent back up towards the top of the drive in something short of an "S".

Once the controller, power supply, and drive are connected, North Star provides two simple checkouts. The first is the running of the following program:

TABLE 1: Sector Test Program

Location (Hex)	Operation (Hex)	
2000	3A 80EB	LDA EB80
2003	C3 0320	JMP 2003

Running this loop presumably turns on the drive motor for 5 seconds.

The next test is a little more visible. In this case you examine address EB20. This results in the sector position field* (0-9) being displayed on the lower four bytes of the data bus.

The first real test of operation is to be found in operating the ROM bootstrap loader. This is done by running from location E900 (the ROM bootstrap smartly happens to be out of the way of the standard Processor Technology or MITS 2K PROMS which tend to be placed in the last 2K of memory for other software uses). The

*The diskette has 10 sector holes in it which are used for locating fields. The corresponding PERTEC drive/controller uses a 16 sector diskette. The two are definitely not interchangeable.

head immediately engages with a "loud click" (more appropriately described as a "clack"), followed by "muted clicks" as the head steps to track "0" for referencing. Now we cross the boundary between hardware and software.

4.0 THE NORTH STAR DISK OPERATING SYSTEM (DOS)

Once the hardware is up and going, the task of putting together an operating system begins. Fortunately the DOS software and instructions supplied by North Star make this an easy job. To proceed in an organized manner, we will first discuss how to interface user I/O hardware to the North Star DOS, and then briefly consider the electrical structure of the diskette, file format, and some ideas on how to manipulate files.

Patching your own I/O routines into North Star's DOS is very simple and straightforward. The following four called routines are required:

COUT: Character output routine. This routine assumes that the DOS has placed the ASCII character to be outputted in register B. It then should test the status of the output I/O device etc. Finally, the routine should return after also placing the outputted character in the accumulator. This is the routine one deals with when interfacing to a VDM-1 video terminal, or other output devices.

CIN: This routine is required to place the 7-bit ASCII character to be input into the accumulator and then return. This routine should also do status checks, etc. Most likely a keyboard will be the target device. However, one can also patch to a cassette interface for linking subroutines by making the computer think it is seeing a terminal.

TINIT: This is a terminal initialization routine. It is meant to handle I/O devices which are required to be set up prior to computer operation. For example, the IMSAI 2SIO and MITS* boards require such an initialization. This routine has no effect on the DOS, so it can also be used for any utility function which might be performed upon disk startup, such as starting a clock to keep track of disk hours, etc.

CONTC: This routine's job is to detect the presence of a "CONTROL-C". It does this by setting the zero flag if "CONTROL-C" occurred. To avoid hanging the computer up in a loop, this routine should check to see if a "CONTROL-C" was typed, and not wait until one is typed.

The actual software patching shown below is for the specific case of an (old) MITS SIO, REV. 1. The changes to be made for Processor Technology's VDM-1 video display are described in the section on BASIC; a "poking" program is used.

Before patching into the DOS, the DOS must be loaded into the computer's active memory. This is done by placing the software diskette provided by North Star into the drive. The disk bootstrap (on PROM) is then run from location E900. The DOS will load and the computer will go into a loop at TINIT; the software provided has self-loops at the patch points and will not operate until it is

*A trademark of PERTEC Computer Corporation.

"personalized". Stop the machine and make the changes shown on Table 2A (Hex) and Table 2b (Octal). Observe that we are changing self-loops to unconditional jumps.

TABLE 2a

PATCH POINT CHANGES (HEX)

Address (Hex)	Was (Hex)	Change to (Hex)	Reason
200D (COUT)	C30D20	C30029	To jump to a memory region having enough room for the I/O routines. This could also be a location in a PROM monitor, such as the TDL monitor which is used for I/O by all TDL software.
2010 (CIN)	C31020	C31029	
2013 (TINIT)	C31320	C33029	
2016 (CONTC)	C31620	C32029	

There are several items to note at this point. First, the self-loop locations in the first column are branched to by the DOS using a "call". Thus the software routines we have just arranged to jump to must contain "returns", and not "jumps" as their concluding operations. Also note that the TINIT routine is implied to be at 2930, above the other routines. As this routine has some alternate uses, it should have some breathing room. This extra room extends to just below 2A00, where the North Star BASIC interpreter (see the next section) starts.

TABLE 2b

PATCH POINT CHANGES (OCTAL)

Address (Octal)	Was (Octal)	Change to (Octal)
010, 015	303	303
	015	000
	010	051
010, 020	303	303
	020	020
	010	051
010, 023	303	303
	023	060
	010	051
010, 026	303	303
	026	040
	010	051

The next step in personalizing is to toggle in the I/O subroutines starting at the locations just branched to. These programs are shown on Tables 3a through 3d. A MITS SIO, Rev. 1 is assumed*.

TABLE 3a

COUT ROUTINE

Location Hex (Octal)	Code:Hex (Octal)	Function
2900 (051,000)	DB00 (333,000)	In (Input status from port 0).
2902 (051,002)	E680 (346,200)	ANI (Check D_7*. For MITS SIO, active low is a "go" condition).
2904 (051,004)	C20029 (302,000,051)	JNZ (If D_7 is not low, try again).
2907 (051,007)	78 (170)	MOV A,B (Move the character from the B register to the accumulator).
2908 (051,010)	D301 (322,001)	OUT (Output character to port 1).
290A (051,012)	C9 (311)	Return (Return to original call).

*D_0 is the least significant bit; D_7 the most significant; MITS convention.

*Similar information is supplied in the DOS manual. However, the above patches are specific to the SIO and differ in some respects from the North Star

TABLE 3b

CIN ROUTINE

Location Hex (Octal)	Code Hex (Octal	Function
2910 (051,020)	DB00 (333,000)	IN (Input status from port 0)
2911 (051,022)	E601 (346,001)	ANI (Check D_0)
2914 (051,024)	C21029 (302,020,051)	JNZ (Loop if D_0 not low)
2917 (051,027)	DB01 (333,001)	IN (Input character to accumulator)
2919 (051,031)	E67F (346,177)	ANI (Mask to 7 bit ASCII character)
291A (051,033)	C9 (311)	Return

TABLE 3c

CCONT ROUTINE

Location Hex (Octal)	Code Hex (Octal)	Function
2920 (051,040)	DB00 (333,000)	IN (Input status to check if character has been typed)
2922 (051,042)	E601 (346,001)	ANI (Test to see if D_0 low)
2924 (051,044)	C0 (300)	RNZ (Return if no character typed)
2925, (051,045)	DB01 (333,001)	IN (Input character to accumulator)
2927 (051,047)	FE03 (376,003)	CPI (Test to see if character was CONTROL-C; set zero flag if yes)
2929 (051,051)	C9 (311)	Return

TABLE 3d

TINIT ROUTINE

Location Hex (Octal)	Code Hex (Octal)	Function
2930 (051,060)	C0 (311) or	Return
Any machine language routine (with a size constraint), using any registers, followed by a return.		

Another DOS change which is optional is to place 01 in location 202B. This causes the disk software to verify every write operation with a read.

Once the changes shown on Tables 2 and 3 are made, the DOS may be properly entered and operated by running from TINIT (examine 2930 Hex and run). An asterix prompt character should appear on your output display. At this point it is a wise idea to start a new diskette and place your new DOS on it. This is done using the following commands. These commands are typed in following the asterisk prompt, followed by a carriage return.

TABLE 4

COPYING THE NEW DOS

Command	Purpose
*IN 1	Initializes diskette (prepares it for future operation).
*CR DOS 10	This creates a file called DOS having a length of ten 256 byte blocks.
*SF DOS 2000	The DOS software starting at 2000Hex and running 2560 bytes is loaded into the DOS file opening on the diskette.

*In the new versions of the North Star DOS, the DOS should be loaded into some other region of memory, adjusted, and saved from there to avoid problems which occur when the DOS tries to save itself.

While you are creating your first disk file operating system, you may as well also include North Star's BASIC. This is done by removing your new diskette, putting in the software diskette supplied by North Star, and doing the following (Table 5).

TABLE 5

CREATING DOS/BASIC FILE
(Version 6, Release 2, 3)

Command	Purpose
*LF BASIC 2A00	This loads the 40-block BASIC program from the software diskette into memory starting at 2A00.*
--Remove software diskette. Insert your new file diskette--	
*CR BASIC 40	Create a 40-block-long file table entry on your new diskette.
*TY BASIC 1	This creates a table entry which tells the DOS that the BASIC file is in machine language.
*GA BASIC 2A00	The DOS is told that the "go address" of this machine language file is 2A00.
*SF BASIC 2A00	The 40-block BASIC program is saved.

The above is a blow-by-blow description of how to get your disk software going. We now consider the logical structure of the diskette being used and the general operating characteristics of the DOS.

The diskette is used on one side only in a single density mode** in which there are 35 concentric tracks numbered 0 to 34. Each track has ten hard

*How the DOS automatically knows BASIC is 40 blocks long will be discussed shortly.

**As opposed to Shugart's new double-density micro-disk drives. PERTEC also has announced a two-sided drive system.

sectors which are physically located according to ten holes in the diskette. Each track in each of these sectors contains a string of overhead data plus 256 bytes of "user" space. That is, in each sector you can place 256 bytes worth of information, not including the accounting information which the DOS takes care of automatically. The North Star convention is to number the sectors from 000 to 349.

The DOS uses sectors 000 to 003 to store (again, automatically) a table or file directory which contains the file name (up to 8 characters long, excluding blanks and commas), type (default is "0") and possibly a "go address". Up to 64 files can be placed on one diskette, the limitation caused by the file directory length. The file types are explained in Table 6.

TABLE 6

FILE TYPES*

TYPE	PURPOSE
0	Default. With this type one can manually (LF and SF) move files to active memory and vice versa.
1	Machine language file which can have a "go address"
2	BASIC program file.
3	BASIC date file.
>3	Assembly language, etc. files.

*The XEK^{C} Assembler/Disassembler software available from the Westminster (CA.) Byte Shop also uses type "8" to denote assembly language files.

North Star's description of the DOS commands is simple and complete. These commands can accomplish the following:

- Initialize or test a diskette. Initialization takes about 10 seconds on the North Star, as compared to eight minutes on a PERTEC unit which is about 4X larger in capacity (the larger of the two PERTEC floppies).

- List directory (file name, size and type, but not "go address"). Newer versions list "go address".

- Create a file space/name.

- Delete same.

- Define a file type.

- Compact a diskette set of files (useful after deletions).

- Load a file into active memory starting at a specific address.

- Save a file similarly.

- Set "go address" of a machine language file. Specified in TYPE command in newer versions.

- Move one file into another on the same diskette.

- Read or write to or from a file or RAM address in integral numbers of blocks.

The last function may sound limited to those who want to move individual bytes around. However, the command structure of North Star BASIC allows one to address any byte position within a block, thus opening up some versatile file management capabilities which are useful in data handling.

Loading BASIC with the new software diskette is easy. Once the DOS is loaded in and running, type "GO BASIC" and roughly two seconds later "READY" will appear, signifying that BASIC has loaded. This response time is not surprising. The rotation speed of the diskette is 300 rpm, or 50 blocks/second. This is the equivalent of an average data transfer rate of roughly 12.5K bytes/second*. North Star BASIC occupies about 10K bytes (more in the newer version) of memory, and therefore should take less than a second to load. The head engagement and drive motor start-up account for a good fraction of the observed response time.

Once BASIC has been entered, the DOS is still alive in the approximately 2.5K of memory below BASIC. The DOS can be returned to by typing in "BYE". This arrangement is convenient, but also causes some small difficulty unless guarded against. For example, if one has written a BASIC program too large to fit on the present diskette, there might be a temptation to test a new diskette, initialize it and try to load the

*The 300 rpm value is from the PERTEC manual; both PERTEC and North Star use the Shugart 400 disk drive. PERTEC states its transfer rate to be 125,000 bits/second. As their disk capacity (user available) is 71,680 data bytes, versus 88,832 data bytes for North Star (first 3 blocks not counted), PERTEC must be specifying a peak, not average, transfer rate. Such is the game of "specsmanship".

file to be saved onto that empty diskette. That approach will not work. The DOS test and initialization functions both use 2.5K bytes of memory above the DOS, cutting into BASIC and destroying it along with your program. Another approach could be to compact the disk being used, hoping to free up enough space for the program. Unfortunately, the compacting function also uses memory above the DOS.

The reason for loading the DOS and BASIC on every diskette, including the spares, is a matter of convenience and safety. From the convenience perspective, giving up 50 blocks out of 350 is a small price to pay to avoid the nuisance of switching diskettes around. From the safety point of view, it is possible to wipe out a complete diskette file if the protect notch on the diskette is not covered and a mistake is made. When such happens, it is reassuring to have extra copies of the DOS and BASIC around. Also, duplicate your files on separate diskettes. Incidentally, if you are going to create a diskette having the DOS on it, make the DOS the first file. The ROM bootstrap on the interface board expects to find the DOS file starting at sector 004.

It is apparent from the memory address references given above that the DOS/BASIC combination starts at 2000 (HEX), leaving the lowest 8K bytes of memory possibly unused. The software package provided by North Star uses the memory above BASIC for program storage, leaving the 0-8K region free. This region may be instead used for probram storage by making the following changes in BASIC.

TABLE 7

CHANGES WHICH ALLOW USE OF 0-8K FOR PROGRAM STORAGE (VERSION 2 BASIC)

Location Hex (Octal)	Code Hex (Octal)
2A06 (052,006)	00(000) Low order byte of the beginning address of program storage space.
2A07 (052,007)	00(000) High order byte.
2A09 (052,011)	FF(377) Low order byte of the end address of program storage space.
2A0A (052,012)	1F(037) High order byte.

The above modification restricts one to a maximum of 8K bytes of program memory. This does not represent a significant limitation in program storage for most applications. However, a programmer might prefer to leave memory space free below the DOS/BASIC for special use. For example, if interrupts are ever to be employed, it is important to note they use service routines located at the beginning of memory (8080 case; the Z-80 microprocessor has other interrupt mode). Thus, it is convenient to leave this space free for eventual interrupt service routines. User machine language routines may also be conveniently placed in this region, and in the next section a VDM-1 video display routine is presented which resides in memory starting at the 2K boundary. Another potential use for the 0-8K region is found in observing that MITS 8K BASIC can be

resident in active memory at the same time as the North Star DOS and BASIC. This is helpful in that North Star's BASIC lacks some features that the MITS 8K BASIC has. This will be discussed in the next section. As a further application, it is possible to use the 0-8K region as byte-wise data storage space, though this can be costly in both software overhead and computing time.

To conclude this section, a few examples of file creation using the DOS are in order. For the first example, consider the storage of MITS 8K BASIC. The actual length of this software is a little more than 6K bytes and is assembled starting at 0000 (Hex). It can comfortably reside in the 8K of memory below the DOS. leaving almost 2K bytes of program storage space between its top and DOS's bottom. One way to put 8K BASIC into the file system is shown below:

1) Load in the DOS
2) Create the 8K BASIC file entry:

```
*CR 8KBASIC 33
*TY 8KBASIC 1
*GA 8KBASIC 0000 (older versions)
```

Observe that 8K BASIC is a machine language file (type 1), 33 blocks long, having 0000 as its "go address".

3) Stop the computer.

4) Load in the MITS 8K bootstrap loader (see the MITS manual. Cassette tape is assumed).

5) Set sense switches (A15 up for MITS SIO terminal at ports 0,1; 88ACR at 6,7).

6) Start tape. Run computer from 0000.

7) When "MEMORY SIZE" is requested by 8K BASIC, stop the computer.

8) Enter into and run the DOS from location 2000 Hex. This brings the DOS back into operation.

9) Save 8K BASIC as follows (remembering to unprotect the diskette);

*SF 8KBASIC 0000

10) Protect the diskette.

8K BASIC is now part of the disk file system. To run it, simply type "GO 8KBASIC". However, remember to set the sense switches before "going"; it's easy to forget those switches. Also, when 8K BASIC asks "MEMORY SIZE?", responding with more than 8191 will cut into and destroy the North Star DOS, which you may not care about anyway.

As a second example, consider adding MITS's excellent 12K BASIC (Extended BASIC) to your file system. This software is also assembled to start at 0000, but it is 10312 bytes long, not including program storage space. Obviously, loading Extended BASIC in will overrun the DOS. In fact, the DOS cannot be used

*Audio Cassette Record Interface

directly to load Extended BASIC as it would be wiped out before completing the program transfer. To deal with this situation one may use two very short machine language files called UPMOVE and DOWNMOVE. The machine language listings of these programs are shown on Tables 8a and 8b. UPMOVE is a routine which shifts the first 63 blocks (16K-256 bytes) of active memory upwards to an address region starting at 16K + 256. UPMOVE itself resides starting at 16K. DOWNMOVE does the reverse. Also, when DOWNMOVE has completed the downwards transfer, it unconditionally jumps to 0000 Hex, thus starting operation of the machine language program it loaded there. To see the mechanics of this procedure, assume you have already created two one-block-long machine language files, UPMOVE and DOWNMOVE, having "go addresses" of 4000 Hex. These "go addresses" are not used in this application, but have been established for future utility. Also assume you have created a machine language file space called 12BASIC which is 42 blocks long (INT(10312/256) + 1). The steps taken in adding Extended BASIC to your repertoire are as follows:

TABLE 8a

UPMOVE ROUTINE

Address Hex (Octal)	Code Hex (Octal)	Function
4000 (100,000)	210041 (041,000,101)	LXI (Load H&L registers with 16K+252 address boundary)
4003 (100,003)	010000 (001,000,000)	LXI (Load B&C registers with OK address boundary)
4006 (100,006)	7C (174)	MOV A,H (Move H to A)
4007 (100,007)	E680 (346,200)	ANI (Test to see if 32K boundary has been reached)
4009 (100,011)	C21340 (302,023,100)	JNZ (Jump to finish loop if move has been completed)
400C (100,014)	0A (012)	LDAX (Load accumulator with data stored at address given in B&C register)
400D (100,015)	77 (167)	MOV (Move accumulator to address specified by H&L registers)
400E (100,061)	03 (003)	INX (Increment B&C)
400F (100,017)	23 (043)	INX (Increment H&L)
4010 (100,020)	C30640 (303,006,100)	JMP (Jump to H register test)
4013 (100,023)	C31340 (303,023,100)	JP (Loop when finished)

TABLE 8b

DOWNMOVE ROUTINE (Similar to UPMOVE)

Address Hex (Octal)	Code Hex (Octal)	Function
4000 (100,000)	210041 (041,000,101)	LXI H&L
4003 (100,003)	010000 (001,000,000)	LXI B&C
4006 (100,006)	7C (174)	MOV A,H
4007 (100,007)	E680 (346,200)	ANI
4009 (100,011)	C21340 (302,023,100)	JNZ
400C (100,014)	7E (176)	MOV (Mov to accumulator data at address specified by H&L registers)
400D (100,015)	02 (002)	STAX (Store accumulator at address specified by B&C registers)
400E (100,016)	03 (003)	INX B&C
400F (100,017)	23 (043)	INX H&L
4010 (100,020)	C30640 (303,006,100)	JMP
4013 (100,023)	C30000 (303,000,000)	JMP(Jump to beginning of program just moved)

1) Load (but do not run) UPMOVE: *LF UPMOVE 4000

2) Stop the computer; toggle in the 12K BASIC bootstrap loader (see the MITS manual); set the sense switches; load the Extended BASIC software into the computer.

3) When "MEMORY SIZE" is requested, stop the computer, examine 4000 Hex and run. The upwards shift of 12K BASIC will take only a moment and completion will be apparent from the address lights on the computer's front panel.

4) Again stop the computer and reload the DOS (run from E900).

5) Replace UPMOVE with DOWNMOVE: *LF DOWNMOVE 4000.

6) Place the combination of DOWNMOVE + Extanded BASIC onto the diskette. *SF 12BASIC 4000.

Extended BASIC may now be loaded and jumped to (remember to set the sense switches beforehand) by typing "GO 12BASIC" when the DOS is active.

The above examples may be applied with variation to create disk files of almost any software.

As a third and very simple (but important) example we consider moving files between diskettes, such as might be done in copying a diskette or reorganizing. This is done as follows:

1) Load the file directly into free memory (e.g., *LF FILEX 0000)

2) Remove original (or source) diskette and insert the new (or destination) diskette.

3) Create an identical file entry on the new diskette.

4) Directly save file (e.g., *SF FILEX 0000)

The flexibility of the North Star DOS, combined with some user creativity, is conducive to the generation of very powerful applications. In the next section we will discuss the North Star BASIC interpreter which operates in conjunction with the DOS. However, it should be noted that sufficient information is supplied in the North Star documentation to allow some measure of linking of the DOS to other BASIC interpreters having user defined machine language subroutines.

5.0 NORTH STAR BASIC

North Star BASIC, Version 6, Release 2 and higher, is a more than adequate interpreter. It has some features which the industry standard, MITS Extended BASIC, does not have, and vice versa. Instead of describing all its features, I will concentrate on the differences between North Star BASIC and MITS Extended BASIC, taking the latter as a benchmark. Also presented in this section is a program for use with the North Star system which does a complete job of linking Processor Technology's VDM-1 video display control board to the DOS; just load the program, type "RUN" and the VDM-1 display device will be part of the microcomputer-disk system.

We first consider North Star BASIC's instruction set. See Tables 9a, b, c, d and 10.

TABLE 9a

DIRECT COMMANDS

RUN	<opt. line no.>
LIST	<opt. line no.> , <opt. line no.>
SCR	(≡ NEW) : Scratch or delete
REN	<opt. beginning value> , <opt. increment> : remember
CONT:	continue after "stop" or "CONTROL-C"
LINE	<no. of characters> (≡ WIDTH = < >) : line width control
NULL	<no.>
LOAD	<file name> : Disk operation*
Save	<file name> : Disk operation*
Edit	<line no.>
BYE:	Return to DOS

*Note, however, that MITS BASIC has similar commands for tape files.

TABLE 9b

STATEMENTS

LET (optional)

IF/THEN/ELSE (can cascade)

FOR/STEP/NEXT

GOTO

EXIT (GOTO out of a FOR/NEXT loop)

ON/GOTO

STOP

END (stop w/o message)

REM (remark)

READ/DATA/RESTORE

INPUT/INPUT1 (no carriage return)

GOSUB/RETURN

PRINT (Formats: nFm, nI, nEm, default)
(also zero suppress, commas, auto $)

FILL (≡ POKE)

OUT

DIM

DEF (function definition)

TABLE 9c

FUNCTIONS

FN (name) (X_1,X_2...) : User defined function

FNEND: End statement for multiple line user defined function.

FREE (0): Remaining free storage

CALL (,): Machine language subroutine call

TYP ( ): Gives type of information in next disk file element.

INP ( ): Input from specified port

EXAM ( ): ≡ PEEK ( )

ABS ( ): Absolute value

SGN ( ): Value/Abs (Value)

INT ( ): Integer

LEN ( ): String length

CHR$ ( ): Decimal → ASCII conversion

ASC ( ): ASCII → Decimal conversion

VAL ( ): Value of number string

STR$ ( ): Number → string conversion

SIN ( ): Sine

COS ( ): Cosine

RND ( ): Random number generator

LOG ( ): Natural Logarithm

EXP ( ): Exponent

SQRT ( ): Square root

TABLE 9d

BASIC FILE ACCESS COMMANDS

Open # <file no.> ,<file name>	: assigns number to an active file; sets file position pointer to file beginning.
Close # <file no.>	: Deactivates access to file and empties write buffer into diskette file.
Write # <file no.> , <list of items>	
Read # <file no.> , <list of variables>	
Write # <file no.> , % <file pointer> , <list of items>	
Read # <file no.> , % <file pointer> , <list of variables>	

TABLE 10

EDIT COMMANDS

Character	Function
Control D < >	Copy to specified character
Control Z	Erase character
Control Q	Backspace
Control A	Copy old character
Control G	Copy rest of old line
Control Y < >	Insert specified character
Control N	Re-edit new line

Starting with the direct command set (Table 9a) we find pretty much the same list as exists in MITS Extended BASIC*, with the following notable exceptions:

- REN in MITS BASIC can start at a specific statement; the North Star analog renumbers the whole program.
- LINE controls the output from the "PRINT" command, and not the "LIST" command. Thus, long program statements attempt to print out full width. This is annoying when using a narrow width printer.
- NULL in North Star's BASIC is limited in argument size to 32 in Release 2, and does not work in Release 4 and 5; MITS BASIC handles NULL's up to 255. At 30 characters/second, the delay time difference between 32 and 255 is one second maximum vs. about 8 seconds maximum. Delays are useful for slow carriage return terminals and in some protocol situations.
- LOAD, SAVE and BYE are all disk commands which have counterparts in the disk version of MITS BASIC.
- EDIT in North Star BASIC is technically a little more versatile than its counterpart in MITS BASIC as the line number may also be changed; the old line is directly treated as a template.

North Star's statement complement (Table 9b) is very similar to that of MITS Extended BASIC, the exceptions being:

*Version 3.2.

- IF in North Star BASIC considers the next line to be the statements added to the same physical line with a ":" or "/". MITS IF uses the next physical line. The MITS version is more flexible. The next line feature of the MITS version can be defeated with an "ELSE"; IF < > THEN < > ELSE < >: Next statement.

- GOTO in North Star is not tolerant of a space between GO and TO. Also, in the North Star version GOTO can not be used as a direct statement; RUN <line number> fulfills this function.

- EXIT in North Star allows a graceful exit from a FOR/NEXT loop leaving a correct stack arrangement.

- INPUT1 stops the carriage return echo after the user's reply. This has some formatting advantages.

- FILL equals MITS's POKE. It's curious that these statements are different only in name as North Star did make allowances in its interpreter to translate ":" (line delimiter) and ";" (no carriage return), which are used in MITS BASIC, into North Star's equivalent "\" and ",". No such translation is done for PEEK and POKE.

The function list (Table 9c) is as complete as any, including both numeric and string functions with the very notable exception being the absence of an inverse trig function in Release 2 (the newer versions contain it). Also, one of the functions, log (X), has a problem (at least in North Star Version 6, Release 2). This error can be programmed around, but is nonetheless annoying.

North Star offers accuracy options ranging from two to fourteen digits. The option must be specified when ordering the software; it is not adjustable once received.

Although the MITS and North Star functions are very similar, there is a very significant difference between the string variable forms. In MITS Extended BASIC, string variables can be up to 255 characters long, which is a limitation I have encountered. These string variables may be dimensioned in matrices such as A$(I,J,K). The power of this structure is that string variable matrices can be created which are very useful in heavy I/O oriented programs. To subsequently manipulate these strings, the MITS software provides functions such as LEFT$, RIGHT$ and MID$.

The North Star string form is much different. String variables are limited in length only by available memory. The VDM-1 video display load program presented later has a Hex string, A$, 678 characters long, which would be illegal in MITS BASIC. However, North Star's string variables can not be subscripted. The equivalent MITS BASIC string functions are implemented in North Star BASIC as shown on Table 12.

TABLE 11

LOGARITHM TEST

```
LIST

10 FOR X=9 TO -9 STEP -1
20 PRINT %10F7,LOG(10↑X)/LOG(10),
30 PRINT "  CORRECT = ",%2I,X
40 NEXT X
READY
RUN

 9.0000000  CORRECT =  9
 8.0000001  CORRECT =  8
 7.0000001  CORRECT =  7
 6.0000002  CORRECT =  6
 5.0000002  CORRECT =  5
 4.0000000  CORRECT =  4
 3.0000000  CORRECT =  3
 2.0000000  CORRECT =  2
 1.0000000  CORRECT =  1
  .0000000  CORRECT =  0
-1.0000000  CORRECT = -1
-1.0000000  CORRECT = -2
-2.0000000  CORRECT = -3
-3.0000000  CORRECT = -4
-4.0000000  CORRECT = -5
-5.0000002  CORRECT = -6
-6.0000002  CORRECT = -7
-7.0000001  CORRECT = -8
-8.0000001  CORRECT = -9
READY
```

TABLE 12

COMPARISON OF MITS AND NORTH STAR STRING FUNCTIONS OPERATING ON A 20 CHARACTER STRING

MITS	North Star Equivalent
LEFT$ (A$,9)	A$(1,9)
RIGHT$ (A$,18)	A$(18,20)
MID$ (A$,9,1)	A$(9,10)

The North Star version is technically equivalent to that of MITS, with perhaps a little more conciseness on the part of North Star. However, the lack of dimensioning capability leads to programming difficulty.

Another difference between the two BASICS is the allowed form for variable names. Having worked largely with FORTRAN IV and MITS BASIC, the use of variable names such as "TEMP", "TIME" and so on is a convenience one has difficulty giving up. In North Star BASIC (and that on the Xerox Sigma 7/9), variable names are restricted to <letter> <optional number> formats. However, perhaps asking for a BASIC which behaves like a FORTRAN interpreter is unfair. Or is it?

Both MITS Extended BASIC and North Star BASIC have editing capabilities (Table 10). Those employed by North Star are explained in the North Star BASIC manual. The two editors are different, and on the whole the MITS version is perhaps more convenient. The differences are:

- The North Star edit structure allows one to copy the old line (with changes) into a new line having a different number. This might be considered North Star's answer to MITS's "PRINTUSING" format tool which North Star does not have.

- In the MITS version, by typing 7SA after the edit command, the old line will be copied up to the 7th occurrence of the character A. There is a similar convenience for changing groups of characters. The North Star edit allows one to search for the next occurrence of a given character only.

- In the North Star editing system typing a "CONTROL-D" plus a character presumably copies the old line up to the specified character.

- The North Star editing system is physically more difficult to use than that of MITS, requiring two separate keys to be pressed simultaneously. Typing "CONTROL-N" with one hand gives me a charley horse of the forth finger.

MITS Extended BASIC is an exceptionally good piece of software, and, in the comparison given above, it significantly outperforms North Star BASIC. This is true without consideration of the operation which MITS has and North Star does not. See Table 13.

TABLE 13

ADDITIONAL MITS EXTENDED BASIC OPERATIONS

SWAP variables

TRACE program steps

ERASE variables

DELETE specified line number intervals (available in new North Star versions)

MOD arithmetic

Integer division

Double precision (arithmetic, not functions)

Clear variables

WAIT

SPC(I) space printing

As mentioned earlier, North Star has provided a very well defined patching structure for interfacing user I/O software drivers. This is well documented and includes examples. The interfacing documentation which comes with Processor Technology's video interface, VDM-1, is equally well documented. Shown on Table 14 is a program written in North Star BASIC which links the VDM-1 to the North Star DOS, which in turn links to North Star BASIC. Observe that statements 310 through 480 create a single string of 678 hexadecimal characters. This string represents the VDM-1 machine language routine which is loaded into memory in type size groups starting at 0800 Hex (001,000 Octal), running upwards for 339 bytes.

After loading this driver, the program then "pokes"("fills" in this language) memory changes to accommodate the MITS 88-SIO status bit structure (low active; least significant bit in status byte). The DOS output routine is then patched to the VDM-1 driver by using a call, moving the outputted character to the accumulator, and making a return.

Running this program will automatically set up the VDM-1 such that all subsequent output will be to the video display. The video driver can be later escaped by reconstructing the original patches. This could be done using a "repair" program.

This load program can also be stored as a file on the North Star disk as part of a routine operating system. For convenience, corresponding system diskettes could start with the following file structure:

TABLE 14: NORTH STAR BASIC LOADER FOR PROCESSOR TECHNOLOGY'S VDM-1

```
10 REM ***************************
20 REM VDM-1 MEMORY LOADER
30 REM 0800 TO 09B6 USED
40 REM ***************************
50 GOSUB 300
60 FOR I=1 TO 439
70 J=2*I
80 B$=A$(J-1,J-1)
90 GOSUB 500
100 M=B*16
110 B$=A$(J,J)\GOSUB 500
120 M=M+B
130 FILL 8*256+I-1,M
140 NEXT I
150 REM   ADJUST VDMLOAD STATUS
160 FILL (8*256+70),194
170 FILL (8*256+125),194
180 FILL (8*256+153),194
190 FILL (9*256+170),001
200 REM   PATCH VDM TO DOS CALL
210 REM     CALL 0800(HEX)
220 FILL 10496,205
230 FILL 10497,000
240 FILL 10498,008
250 REM    AFTER RETURN, MOVA,B
260 FILL 10499,120
270 REM    RETURN TO DOS INTERIOR
280 FILL 10500,201
290 END
300 DIM A$(1000)
310 A$=""
320 A$=A$+"22B709210000393 1F009E5D5C5F5CD1A08F1C1D1E1F92AB709C9"
330 A$=A$+"78E67FFE7FC8FE5FCA2D09FE5BCACE08FE5DCA4109FE5CCA8108"
340 A$=A$+"FE0DCA1509FE20D8F53AB60967 2E80CDA709CA6E08CDAC09CD52"
350 A$=A$+"08C36D08FE3AD27708FE31DA7708E60F4FAF370DCA690817C361"
360 A$=A$+"0832B609C92B7CB7C26D08F1C300009FE20C0CDA709CA7A08C9"
370 A$=A$+"CD150921BA087EFE58CA9608E5CD0009E123C38708CDA709CA96"
380 A$=A$+"08CDAC09FE3AD2B408FE31DAB408F5CD0009CD1509F1C3520821"
390 A$=A$+"CC08C3870820 4E4557205350454544202831 2D39293F20205821"
400 A$=A$+"00CC7CC6043620 23BCC2D408AF32B40932B50932B2092F32B309"
410 A$=A$+"3E0F32B109CDF208C93AB4090707070721B509B6D3C8C94F3AB2"
420 A$=A$+"09473AB109CD7009713AB2093CFE40C223093AB209473AB109CD"
430 A$=A$+"9F09CD5209973 2B209473AB109C3830 93AB209473AB109CD9F09"
440 A$=A$+"2B3620053AB109C3830 93AB3092F32B3093AB209473AB109C383"
450 A$=A$+"0921B509E57E3496010000CD7009014020702C0DC26209E17EE6"
460 A$=A$+"0F77C3F2086F3AB50985 0F0F6FE602C6CC677DE6C0806FC9E60F"
470 A$=A$+"32B109CD7009783 2B2093AB309B77ECA9B09F68077C9E67F77C9"
480 A$=A$+"CD70097EE67F77C9DB00E640C9DB01E67FC9000001000006"
490 RETURN
500 B=0
510 IF B$="A" THEN B=10
520 IF B$="B" THEN B=11
530 IF B$="C" THEN B=12
540 IF B$="D" THEN B=13
550 IF B$="E" THEN B=14
560 IF B$="F" THEN B=15
570 IF B>=10 THEN RETURN
580 B=VAL(B$)\RETURN
READY
```

Name	# Blocks
Directory	3
DOS	10
BASIC	40
VDM	8

This means that an overhead of 58 blocks (out of 350) is carried for the sake of operating convenience. Other users may have different values and not want to have such file duplication.

6.0 CONCLUSIONS

The previous sections have served to demonstrate how much power and convenience is available through the use of a micro-disk system such as that provided by North Star.

If there were a single area in the North Star system to be examined for improvement, it would be the software beyond the DOS. Besides having the logarithm error, North Star BASIC is generally not as capable as MITS Extended BASIC, which must be considered a benchmark. North Star has a good piece of hardware which is partially sold by its software back-up, which at this point is good, but not great. Specific areas for improvement and expansion of their BASIC interpreter are:

- Fix logarithm error (apparently done in the latest releases)
- Provide string variable subscripting
- Include on on-line double precision option
- Add starting statement choice to "renumber"
- Allow line width control of all printing
- Use single key edit commands
- Allow longer variable names
- Implement a "trace" for debugging

The general software frontiers presently being pushed are a BASIC compiler and PASCAL. The features which are important for a BASIC compiler are:

- Compile/Interpret option for program building and debugging
- Disk save and load functions for object code
- Fast running object code (which will end the argument that BASIC is inefficient)
- Disk oriented linking of programs in both compile and interpret* modes.

*The PASCAL software is just now becoming available, and will be considered in a later report.