NORTH STAR ★ COMPUTERS, INC.
2547 Ninth Street • Berkeley, California 94710 • (415) 549-0858

North Star BASIC

Version 5

REVISION 1

INTRODUCTION

North Star BASIC was implemented by Dr. Charles A. Grant and Dr. Mark Greenberg of North Star Computers, Inc. This manual describes Version 5, an extended BASIC with such features as multiple-dimensioned arrays, strings, multiple-lined functions, formatted output, and machine language subroutine capability. Version 5-FPB has been specially assembled for use with the North Star Floating Point Board, for increased speed and reduced memory requirements. It is possible to use Version 5-FPB on an 8K read-only-memory, if desired. Version 5 has been assembled for 8-digit precision. For accuracy of 2, 4, 6, 10, 12, or 14 digits, a special order may be made. Version 5 is assembled with origin at 8192. Special orders may also be made for different origins.

Every large software system has errors in it. Although Version 5 has been extensively tested, it will be no exception to this rule. Updates will be mailed to purchasers who request this service.

This manual assumes familiarity with some version of BASIC on the part of the reader. There are many tutorial and advanced publications on BASIC available in both stores and libraries.

Version 6 of North Star BASIC will permit random and sequential disk file operations compatible with the North Star MICRO-DISK SYSTEM which uses the Shugart Mini-floppy disk drive.

INPUTTING A PROGRAM

Every program line begins with a line number. Any line of text typed to BASIC in command mode that begins with a digit is processed by the editor. There are four possible actions which may occur:

1) A new line is added to the program. This occurs if the line number is legal (range is Ø thru 65535) and at least one character follows the line number in the line.

2) An existing line is modified. This occurs if the line number matches the line number of an existing line in the program. That line is modified to have the text of the newly typed-in line.

3) An existing line is deleted. This occurs if the typed-in line contains only a line number which matches an existing line in the program.

4) An error is generated. If the line number is out of range, or the line is too long, or the memory would become full, then an error message is generated and no other action is taken by BASIC.

Blanks

Blanks preceding a line number are ignored. The first non-digit in a line terminates the line number (even blanks). Multiple blanks are permitted anywhere in a line for indentation purposes, but not within reserved words or constants.

Multiple statements per line

Multiple program statements may appear on a single line, if separated by a (\) back slash. A line number must appear only at the beginning of the first statement on the line.

Typing mistakes

If a typing mistake occurs during the entering of any line of text to BASIC, there are two possible corrective actions available:

When the user types an (@) at-sign character, BASIC completely ignores all input on the current line being typed in, and types a carriage return. The correct line may then be typed to BASIC.

When the user types a left-arrow (under-line on some keyboards), BASIC will ignore the previously typed character.

Compatibility

Certain characters, when they appear in programs, are automatically translated into other characters. This is done to minimize the effort of converting programs written for other BASIC systems. In particular, left bracket ([), right bracket (]), colon (:), and semi-colon (;) are converted to left paren, right paren, back slash (\), and comma (,) respectively. This conversion is not done within quoted strings in a program.

COMMANDS

RUN <optional line number>
Begin program execution either at the first line of the program or else at the optionally supplied line number.

LIST <optional line number>,<optional second line number>
If no arguments are supplied, then print the entire existing program. If one line number is supplied, then print the program from the specified line number to the end. If two line numbers are supplied, then print the program in the region between the two line numbers.

SCR
Delete (scratch) the existing program and data, in preparation for entering a new program.

REN <optional beginning value>,<optional increment value>
Renumber the entire existing program. If the first argument is not supplied, then 1Ø is used as the initial statement renumber value. If the second argument is not supplied, then 1Ø is used as the increment value.

CLEAR
Clear all variables. This command deletes all arrays, strings and functions, and initializes all scalar variables to zero.

CONT
This command causes execution of a running BASIC program to continue after a STOP statement or after a control-C stop.

LINE <number of characters>
This command defines the line length of the user terminal. No input line will be accepted longer than the specified value, and no output line will be printed longer than the specified value. The maximum value is 132. The initial value is 72.

NULL <number of nulls>
This command specifies the number of ACSII null characters to output following the output of each carriage return character. The initial value is zero.

CONSTANTS

Magnitude range: .1E-63 thru .99999999E+63

Constants appearring in programs are rounded to 8 digits if necessary. Internal representation of numbers is binary-coded-decimal.

NAMES

All user defined names are one or two characters long: a letter of the alphabet optionally followed by any digit. For example: A, Z0, and Q9 are legal names. The same name may be used to identify different values, as long as the values they identify are of different types. For example, it is possible to have a scalar variable named A1, an array named A1, a string named A1$ and functions named FNA1 and FNA1$. There is no relationship between these entities.

OPERATORS

Numeric: +, -, /, *, ↑ (or ^ on some keyboards)

Relational: =, <, >, <>, >=, =>, <=, =<
A relational operation gives a 1 (true) or 0 (false) result.

Boolean: AND, OR, NOT
A Boolean operand is true if non-zero, and false if zero. The result of a boolean operation is 1 or 0.

STATEMENTS

Only some statements listed below are accompanied by discussion. File accessing statements are described in a later section. Consult the example programs in Appendix 1 for questions about the use of a particular type of statement.

LET

The LET is optional in assignment statements. Multiple assignments are not allowed. The statement A=B=Ø assigns true or false to A depending on whether or not B equals Ø.

IF

An IF statement may optionally have an ELSE clause. A THEN or ELSE clause may be a LET statement, a RETURN statement, another IF statement or a GOTO, for example. If either the THEN clause or the ELSE clause is a simple GOTO, then the GOTO reserved word may be optionally omitted.

```
100 IF A=B THEN 150 ELSE A=A-1
```

FOR

FOR loops may be multiply nested. The optional STEP value may be positive or negative. It is possible to specify values such that the FOR loop will execute zero times. For example,

```
100 FOR J=5 to 4 \ PRINT J \ NEXT
```

NEXT

A NEXT statement may optionally specify the control variable for the matching FOR statement, as a check for proper nesting.

GOTO

ON

The ON statement provides a multi-branched GOTO capability. For example,

```
100 ON J GOTO 500,600,700
```

will branch to 500, 600 or 700 depending on the value of J being 1,2, or 3 respectively.

EXIT

The EXIT statement is identical to a GOTO except that it has the effect of terminating any active FOR loops and reclaiming the associated internal stack memory. It should be used for branching out of a FOR loop.

STOP

END

REM

READ

DATA

RESTORE

The RESTORE statement may optionally include a line number, specifying where the READ pointer is to be restored to. In the absence of the optional line number, the READ pointer is set to the first line of the program.

INPUT
INPUT1

The INPUT or INPUT1 statement may optionally specify a literal string which is typed on the terminal as a prompt for the input instead of a question mark. To inhibit the echoing of the carriage return at the end of user input, use the INPUT1 statement.

```
100 INPUT "TYPE VALUE: ",V
```

GOSUB

RETURN

PRINT

The PRINT statement may include a list of expressions, variables, or constants separated by (,) commas. Note that if the list of variables is terminated by a comma, then a carriage return is not typed. The null PRINT statement will cause only a carriage return to be typed. A semi-colon is equivalent to a comma in the print list. All values are printed in free format, separated by a blank, unless formatting is specified. If a value will not fit on the current output line, then it is printed on the next output line. Advancement of the printer to a specified output position may be accomplished with the TAB function. Formatting may be accomplished by including a "format string" in a print statement (see below).

FILL

This statement permits filling a specified byte in the computer memory with a given expression value. For example, FILL 100,J+3 will fill memory byte 100 with J+3.

OUT

This instuction permits doing an 8080 OUT instruction. For example, OUT 5,3 will perform an OUT 5 instruction with 3 in the 8080 accumulator.

ARRAYS

Arrays may be dimensioned with any number of dimensions, limited only by available memory, e.g.,

```
100  DIM A(1), B7(5,2,3,4,5,6)
```

Array indexing starts at element 0. Array A in the above example actually has two elements, A(0) and A(1). Use of an undimensioned array causes automatic dimensioning to a one-dimension, 10 element array. Arrays may not be re-dimensioned within a program.

STRINGS

Strings of 8-bit characters may be dimensioned to any size, limited only by available memory, e.g.,

```
100 DIM A$(1),A1$(10000)
```

Note that a string name is a varaible name followed by a ($) dollar sign. Substrings may be accessed as A$(N,M) which is the substring of characters N thru M. For example, if A$ is "ABCDEF" then A$(3,5) is "CDE". Alternatively, A$(N) identifies the substring including characters N thru the last character in the string. The concatenation operator is a plus sign.

If an assigned value is larger than the destination string or substring, then it is truncated to fit. If an assigned value to a substring is shorter than the substring, then the extra characters of the substring are left unmodified. A string variable used before being DIMensioned is given the default dimension of 10. Strings may not be redimensioned within a program.

Strings, substrings and string expressions may be used in conjunction with: LET, READ, DATA, PRINT, IF, and INPUT statements. The string IF statement does alphabetic comparisons when the relational operators are used, .e.g.

```
100  IF A$+B$<"SMITH" THEN 50
```

When string variables are INPUT, they must not be quoted. When strings appear in DATA statements, they must be quoted.

USER DEFINED FUNCTIONS

User-defined functions (either of type string or numeric) may be 1-line or multiple line functions. There may be any number of numeric arguments. Parameters are "local" to a particular call of a function. That is, the value of the variable is not affected outside of the execution of the function.

Functions are defined before execution begins (at RUN time), so definitions need not be executed, and functions may be defined only once.

Multiple line functions must end with a FNEND statement. A multiple-line function returns a value by executing a RETURN statement with the value to be returned, for example:

```
100 DEF FNA(X,Y,Z)
200 IF Z=1 THEN RETURN X
300 X=Y*Z+X*3
400 RETURN X
500 FNEND
600 PRINT FNA(1,2,X+Y)
```

BUILT IN FUNCTIONS

```
FREE(Ø)      returns number of bytes remaining in free storage.
ABS(expr)    returns the absolute value of the expression
SGN(expr)    returns 1, Ø, or -1 if the value is +, Ø, or -
INT(expr)    returns the integer portion of the expression value
LEN(string name)     returns the length of the specified string
CHR$(expr)    returns a string with the specified character
VAL(string expr)  returns the numeric value of the string
STR$(expr)  returns a string with the specified numeric value
ASC(string name)  returns ASCII code of first character in string
SIN(expr)    returns SINE of the expression
COS(expr)    returns the COSINE of the expression
RND(expr)    returns a random number between Ø and 1
LOG(expr)    returns the natural log of the expression
EXP(expr)    returns the value of e raised to the specified power
SQRT(expr)  returns the positive square root of the expression
CALL(expr,optional expr)  see below
EXAM(expr)  return contents of addressed memory byte
INP(expr)    return result of 8Ø8Ø IN to specified port
```

MACHINE LANGUAGE SUBROUTINE INTERFACING

The built-in function CALL takes a first argument which is the address of a machine language subroutine to call. The optional second argument is a value which is converted to an integer and passed to the machine language subroutine in DE. The CALL function returns as value the integer which is in HL when the machine language subroutine returns.

FORMATTED OUTPUT

If no format string is present in a PRINT statement, then all numeric values will be printed in the "default format". (The default format is initially set to be free format.) A format string appears anywhere in the print list and must begin with a per cent (%) character, e.g.

```
PRINT %$1Ø F3,J
```

A format string consists of optional format characters followed optionally by a format specification. The format characters are:

C place commas to the left of decimal point as needed
$ put a dollar sign to the left of value
Z suppress trailing zeroes
make this format string the default specification

Format specifications (similar to FORTRAN) are:

nFm F-format. The value will be printed in a n-character field, right justified, with m digits to the right of decimal point.

nI I-format. The value will be printed in a n-character field, right justified, if it is an integer. (Otherwise an error message will occur.)

nEm E-format. The value will be printed in scientific notation in a n-character field, right justified, with m digits to the right of the decimal point.

All printed values are rounded if necessary. A null format string will print values in free format.

Control-C

Typing the control-C character (ETX on some keyboards) has the effect of prematurely interrupting BASIC from whatever it is doing. If a LIST is in progress, the listing will be terminated at the completion of the output of the current line. If a RUN or CONT is in progress, then execution will stop after the completion of the currently executing statement.

DIRECT STATEMENTS

When BASIC is in command mode, certain statements may be typed for immediate execution. This is typically used for examining the values of certain variables to diagnose a programming error. Note that an exclamation point (!) may be used as a shorthand way of typing the PRINT reserved word. No direct statement is permitted which transfers control to the BASIC program. Also, DATA, DEF, FOR, NEXT, INPUT, and REM are forbidden.

SAVING AND RELOADING PROGRAMS

Two methods are available to save and reload BASIC programs. One method is to LIST the program with a paper-tape punch turned on. If this is done, then sufficient nulls must be used (see NULL command) that the program can be successfully reloaded. Alternatively, a program (either machine language or BASIC) can be written to copy the contents of the memory where the BASIC program resides to a secondary storage medium. BASIC programs can be similarly be reloaded.

LOADING AND USING BASIC

We expect users to load BASIC from paper tape once, and copy it to a more convenient storage medium, although this is not necessary.

Version 5

> The system software and temporaries are assembled with origin at 8192. Memory is used through (approximately) cell 17000. See Appendix 2 for the exact value. Memory for BASIC programs and data may reside anywhere else in the memory of your computer (see below).

Version 5-FPB

> The system software (read-only storage) resides within addresses 8192 and 16383. Temporary values (read-write storage) reside between 16384 and approximately 16500. Note that there is unused read-only memory between 16300 and 16383 in case you need it. Memory for BASIC programs and data may reside anywhere else in the memory of your computer (see below). BASIC assumes that you use the suggested values for the two jumpered FPB addresses. If for some reason you select other values, certain BASIC memory cells must be modified. Consult the factory if you decide to do this.

The paper tape must be loaded with the supplied loader (see Appendix 3). Every paper tape has been automatically verified before shipment. Your tape is guaranteed to be free of any punching errors.

Before creating your final version of BASIC, you must interface BASIC to your terminal. Entry point addresses and calling conventions for the four I/O routines used by BASIC are listed in Appendix 2. You must JMP from the entry points to your own routines elsewhere in memory.

There are two main entry points to BASIC. The value of the symbol ORIGIN is 8192 unless you have made a special order:

ORIGIN+0: This is the continue address. It may be used to re-enter a previously saved version of BASIC, or to continue after an interruption in the use of BASIC.

ORIGIN+4: BASIC is initialized, and is started with no existing BASIC program.

The version of BASIC on paper tape assumes that BASIC will be used in one contiguous 12k region of memory beginning at 8192. BASIC programs and data begin directly following BASIC temporary storage. The second LXI instruction at the entry point (see Appendix 2) must be modified to load the value of the last cell in memory you wish to use for BASIC programs and data. Should you wish your BASIC programs and data to reside elsewhere, change both LXI instuctions (see Appendix 2) to delineate the region you desire.

Appendix 1: SAMPLE PROGRAMS.

```
100 REM  A NUMERIC SORT PROGRAM
110 REM
120 DIM A(15)
130 PRINT "INPUT FIFTEEN VALUES, ONE VALUE PER LINE"
140 FOR J=1 TO 15
150 INPUT A(J)
160 NEXT
170 REM  DO EXCHANGE SORT UNTIL ALL IN ORDER
175 F=0 \ REM THIS FLAG USED TO SIGNAL WHETHER ARRAY IN ORDER YET
180 FOR J=2 TO 15
190 IF A(J-1)<=A(J) THEN 220
200 T=A(J)\ A(J)=A(J-1)\ A(J-1)=T\ REM EXCHANGE A(J) AND A(J-1)
210 F=1\ REM SET FLAG
220 NEXT
230 IF F=1 THEN 175\ REM  LOOP IF EXCHANGES HAPPENED
240 PRINT "SORTED ARRAY: ",
250 FOR J=1 TO 15\ PRINT A(J),\ NEXT
```

```
100 REM   CHARACTER SORT
110 REM   EXAMPLE USING STRINGS AND FUNCTION
115 DIM A$(72)
120 INPUT "TYPE A STRING OF CHARACTERS: ",A$
130 IF LEN(A$)=0 THEN 120
140 IF FNA(LEN(A$))=1 THEN 140\ REM CALL FNA UNTIL IT RETURNS ZERO VALUE
150 PRINT "SORTED ARRAY: ",A$
155 END
160 DEF FNA(N)\ REM CHARACTER SORT
170 REM  RETURN 0 IF A$ SORTED, ELSE RETURN 1
175 F=0
180 FOR J=2 TO N
190 IF A$(J-1,J-1)<=A$(J,J) THEN 220
200 T$=A$(J,J)\ A$(J,J)=A$(J-1,J-1)\ A(J-1,J-1)=T$
210 F=1
220 NEXT
230 RETURN F
240 FNEND
```

```
100 REM INPUT A STRING AND CHECK THAT IT IS A LEGAL INTEGER
105 REM
110 DIM A$(72)
115 INPUT "TYPE AN INTEGER: ",A$
120 IF LEN(A$)=0 OR LEN(A$)>8 THEN GOTO 500
130 FOR J=1 TO LEN(A$)
140 IF A$(J,J)<"0" THEN 500
145 IF A$(J,J)>"9" THEN 500
150 NEXT J
155 PRINT "THE INTEGER IS OK: ",VAL(A$)
160 GOTO 115
500 REM
510 PRINT "NOT AN INTEGER!"
520 GOTO 115
```

```
100 REM
110 REM  PRINT A SINE WAVE VERTICALLY ON THE PAGE
115 FOR J=1 TO 100 STEP .1
120 T=SIN(J)
130 S=INT(30*T)
140 PRINT TAB(30+S),"*"
150 NEXT
160 STOP
```

```
100 REM
110 REM  PRINT A TABLE OF FORMATTED VALUES
120 REM
130 FOR J=1 TO 100
140 PRINT %3I,J,
150 PRINT %6F3,SIN(J),%7F4,COS(J),
160 PRINT %10E3,EXP(J),
170 PRINT %12F10,RND(J)
180 NEXT
```

Appendix 2: ENTRY POINTS AND I/O ROUTINES FOR BASIC

```
0000                    ORG 8192        2000 HEX
2000 37         EP0     STC             CONTINUE ENTRY POINT
2001 C30520             JMP EP11
2004 AF         EP1     XRA A           INITIALIZATION ENTRY POINT
2005 210000     EP11    LXI H,ENDBAS    FIRST MEMORY BYTE AFTER TEMPS
2008 110000             LXI D,ENDMEM    LAST AVAILABLE MEMORY BYTE
200B C30000             JMP START
200E                    *
200E                    *
200E                    *INPUT A CHARACTER FROM TERMINAL, AND RETURN IN ACC.
200E                    *CHARACTER MUST BE TRIMMED TO 7-BIT ASCII.
200E                    *ALL REGISTERS MAY BE USED.
200E                    *
200E C30000     CIN     JMP 0           YOUR ROUTINE MUST DO A RET
2011                    *
2011                    *
2011                    *OUTPUT A CHARACTER FROM REGISTER B TO TERMINAL
2011                    *ON RETURN THE CHARACTER SHOULD ALSO BE IN A.
2011                    *NO OTHER REGISTERS SHOULD BE MODIFIED.
2011                    *
2011 C30000     COUT    JMP 0           YOUR ROUTINE MUST DO A RET
2014                    *
2014                    *
2014                    *THIS ROUTINE REPORTS WHETHER A CTRL-C HAS BEEN TYPED.
2014                    *IT RETURNS WITH Z SET IF ONE HAS BEEN TYPED.
2014                    *ALL REGISTERS MAY BE USED.
2014                    *
2014                    *
2014 C30000     CONTC   JMP 0           YOUR ROUTINE MUST DO A RET
2017                    *
2017                    *
2017                    *THIS ROUTINE IS CALLED AT START-UP TIME TO PERFORM
2017                    *TERMINAL INITIALIZATION, IF NEEDED.
2017                    *ALL REGISTERS MAY BE USED.
2017                    *
2017 C30000     TINIT   JMP 0           YOUR ROUTINE MUST DO A RET
201A                    *
201A                    *
201A                    *
```

Appendix 3: HEX CHECKSUM LOADER FOR PAPER TAPE

```
0000           *HEX FORMAT CHECKSUM LOADER
0000           *
0000           *
0000           *RECORD FORMAT IS:
0000           *  A ':' CHARACTER
0000           *  A BLOCK SIZE (TWO HEX CHARACTERS GIVING ONE BYTE)
0000           *  A STARTING ADDRESS (FOUR HEX CHARACTERS)
0000           *  A RECORD TYPE (TWO HEX CHARACTERS)
0000           *  THE RECORD (TWO HEX CHARACTERS PER BYTE)
0000           *THE LAST RECORD IS INDICATED BY A 0 SIZE
0000           *
0000           *
0000                 ORG 16384+3072
4C00                 DS 6              THIS SPACE USED FOR STACK
4C06 31064C    START LXI SP,START
4C09 CD534C    BLOCK CALL CIN          READ A CHARACTER
4C0C FE3A            CPI ':'           WAIT FOR BLOCK START
4C0E C2094C          JNZ BLOCK         TRY AGAIN
4C11 CD3B4C          CALL HBYTE        GET A BYTE (2 HEX CHARS)
4C14 4F              MOV C,A           SHOULD BE RECORD SIZE
4C15 B7              ORA A             SET CONDITIONS
4C16 CA164C    DONE  JZ DONE           LOOP IF DONE
4C19 CD3B4C          CALL HBYTE        GET A BYTE
4C1C 67              MOV H,A           HIGH ORDER ADDRESS BYTE
4C1D CD3B4C          CALL HBYTE        GET A BYTE
4C20 6F              MOV L,A           LOW ORDER ADDRESS BYTE
4C21 CD3B4C          CALL HBYTE        GET A BYTE AND IGNORE IT
4C24 0600            MVI B,0           INITIALIZE CHECKSUM
4C26 CD3B4C    LOOP  CALL HBYTE        GET A DATA BYTE
4C29 77              MOV M,A           STORE IT IN MEMORY
4C2A 80              ADD B             ADD IN CHECKSUM
4C2B 47              MOV B,A           UPDATE CHECKSUM
4C2C 23              INX H             UPDATE ADDRESS
4C2D 0D              DCR C             DECREMENT COUNT
4C2E C2264C          JNZ LOOP          LOOP UNTIL RECORD IS IN
4C31 CD3B4C          CALL HBYTE        GET THE CHECKSUM ON TAPE
4C34 80              ADD B             ADD TO RUNNING CHECKSUM
4C35 C2354C    ERR   JNZ ERR           SHOULD BE 0
4C38 C3094C          JMP BLOCK         GET ANOTHER BLOCK
4C3B           *     THIS ROUTINE READS 2 HEX CHARS AND
4C3B           *     RETURNS THE VALUE IN 1 BYTE
4C3B CD484C    HBYTE CALL HEX          GET 4 BIT VALUE FROM HEX
4C3E 07              RLC               SHIFT
4C3F 07              RLC                THE
4C40 07              RLC                 CHARACTER
4C41 07              RLC                  LEFT
4C42 57              MOV D,A           SAVE IT IN D MOMENTARILY
4C43 CD484C          CALL HEX          GET 4 BIT VALUE FROM HEX
4C46 B2              ORA D             PUT BACK THE HIGH ORDER 4 BITS
4C47 C9              RET
4C48           *
```

```
4C48           *      GET A HEX CHAR AND CONVERT TO 4 BIT VALUE
4C48 CD534C    HEX    CALL CIN        GET CHARACTER
4C4B D630              SUI '0'         ASSUME IT WAS A DIGIT
4C4D FE0A              CPI 10          SEE IF IT WAS A DIGIT
4C4F D8                RC              RETURN IF WAS DIGIT
4C50 C6F9              ADI '0'-'A'+10  ADJUST FOR LETTER CASE
4C52 C9                RET
4C53           *      THIS MUST BE THE INPUT ROUTINE FOR YOUR SYSTEM.
4C53           *      A SAMPLE ROUTINE IS SHOWN. ONLY CLOBBER ACC.
4C53 DB00      CIN    IN 0            ASSUME STATUS PORT IS 0
4C55 E601              ANI 1           ASSUME READY BIT IS BIT 0
4C57 CA534C            JZ CIN          JUMP IF NOT READY
4C5A DB01              IN 1            ASSUME DATA PORT IS PORT 1
4C5C E67F              ANI 177Q        TRIM OFF PARITY BIT
4C5E C9                RET
```