digital group software systems inc.

MAXI BASIC - LEVEL 1.1

Introduction

This manual describes MAXI BASIC - Level I, an extended BASIC with such features as multiple-dimensional arrays, strings, formatted output, and machine language subroutine capability, with plain english diagnostics.

The user of MAXI-BASIC is assumed to be familiar with some version of BASIC. The purpose of this manual is not to teach BASIC but rather to define commands, statements and operating procedures of MAXI-BASIC.

System Size

MAXI-BASIC and operating system reside in the first 63 pages of memory (≃13K). Therefore, the minimum system memory size should be 18K. MAXI-BASIC automatically searches for top of memory and adjusts itself for any size of continous memory.

Inputting a program

Every program line begins with a line number. Any line of text typed to MAXI 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 MAXI 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 program statements

Multiple program statements may appear on a single line, if separated by a (:) colon. A line number must appear only at the beginning of the first statement on the line.
NOTE: The colon (:) must be preceeded by a space for correct operation.

Typing mistakes

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

When the user types an (@) at-sign character, MAXI 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 MAXI BASIC.

When the user types a left-arrow (under-line or RUBOUT on some keyboards), MAXI BASIC will backspace to the previously typed character. (It is not possible to backspace past the beginning of line).

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 ([), and right bracket (]), are converted to left paren, and right paren 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 specified line number. If two line numbers are supplied, then print the program in the region between the two line numbers. If one line number and a comma are typed with no second line number, then print the program from the specified line number to the end.

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, the 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.

digital group software systems inc.
po box 1086, arvada, colorado 80001

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.

SAVE

This command is used to save a program onto a cassette. See saving and reloading programs

LOAD

This command is used to load a program from cassette to memory. See saving and reloading programs

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, ZØ, 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 Λ on some keyboards)

Relational: =, <, >, <>, >=, =>, <=, =<

A relational operation gives a 1 (true) or Ø (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 Ø.

STATEMENTS

Only some statements listed below are accompanied by discussion. 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.

```
1ØØ LET A = A+1: B(J) = B(J-1)
```

IF, THEN, ELSE

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.

```
lØØ IF A=B THEN 15Ø ELSE A=A-1
```

FOR, NEXT

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:

```
lØØ FOR J=5 to 4  PRINT J  NEXT
```

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

GOTO

The GOTO statement is a direct branch to the designated line number.

```
lØØ GØTØ 71Ø
```

ON

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

```
lØØ ON J GOTO 5ØØ, 6ØØ, 7ØØ
```

will branch to 5ØØ, 6ØØ or 7ØØ 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.

```
lØØ IF A (J)→lØØ EXIT 32Ø
```

STOP

The STOP statement halts execution of the program and displays the message "STOP IN LINE XXX". After a STOP has been encountered, the program can be continued starting at the next line by typing CONT.

```
lØØ STOP
```

END

The END statement also halts the execution of the program. However, unlike STOP, there is no way to continue from an END statement. If the END statement is the last line number of the program, it may be optionally omitted.

```
lØØ END
```

REM

The REM statement is used to annotate the program. Any REM statement is ignored by the MAXI-BASIC interpreter.

```
lØØ REM THIS PROGRAM CALCULATES PI
```


digital group software systems inc.
po box 1086, arvada, colorado 80001

READ, DATA

The READ and DATA statements allow the user to input pre-determined data into a program. The READ statement transfers data named in the DATA statement into the varriables or array which have been named by the READ statement.

```
1ØØ DATA 12.17, "VOLTS", 2.4EØ9, "OHMS"
11Ø READ V, V$, 0, 0$
```

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.

```
1ØØ RESTORE 75
```

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.

```
1ØØ INPUT "TYPE VALUE: ",V
```

GOSUB, RETURN

The GOSUB statement branches the program to a subroutine with the starting line number specified in the GOSUB statement. The RETURN statement is the last line of the subroutine, and branches the program to the line following the GOSUB statement.

```
1ØØ GOSUB 13ØØ
1350 RETURN
```

PRINT

The PRINT statement may include a list of expressions, variables, or constants separated by (,) commas, or semicolons (;). Note that if the list of variables is terminated by a comma, or semicolons then a carriage return is not typed. A comma separator will output five spaces between variables. A semicolon separator will output no spaces between variables. The PRINT "" statement will cause a carriage return to be printed. All values are printed in free format, 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). A # sign is interpreted as the word PRINT.

```
1ØØ PRINT "PT=": P: PRINT"": PRINT D,17.5, E
```

FILL

This statement permits filling a specified byte in the computer memory with a given expression value. For example, FILL 1ØØ, J+3 will fill memory byte 1ØØ with J+3.

```
1ØØ FILL 1ØØ, J+3
```


digital group software systems inc.
po box 1086, arvada, colorado 80001

OUT

This instruction permits doing an 8Ø8Ø or Z-8Ø OUT instruction. For example, OUT 5,3 will perform an OUT 5 instruction with 3 in the 8Ø8Ø or Z-8Ø accumulator.

```
1ØØ OUT 5,3
```

ARRAYS

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

```
1ØØ DIM A(1), B7(5,2,3,4,5,6)
```

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

STRINGS (See Appendix 1, Page 3)

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

```
1ØØ DIM A$(1),A1$(1ØØØØ)
```

Note that a string name is a variable 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 1Ø. Strings may not be redimensioned within a program. Strings may not be modified until they have been defined by a LET A$= or INPUT A$ statement.

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.

```
1ØØ  IF A$+B$<"SMITH" THEN 5Ø
```

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

NOTE: A string array is initialized as follows: (Where N = Length of string).

```
For X = 1 to N :A$=A$+" " : NEXT X
```

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:

```
1ØØ DEF FNA(X,Y,Z)
2ØØ IF Z=1 THEN RETURN X
3ØØ X-Y*Z+X*3
4ØØ RETURN X
5ØØ FNEND
6ØØ 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 or Z-8Ø IN to specific port
```

MACHINE LANGUAGE SUBROUTINE INTERFACING

The built-in function CALL takes a first argument which is the decimal 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.

NOTE: CALL is a function and not a verb. Therefore:

```
1Ø  LET X=CALL(1234)  and not
1Ø  CALL(1255)
```

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 %$1ØF3,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.)

digital group software systems inc.
po box 1086, arvada, colorado 80001

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.
*N includes preceeding +or-, and all commas and dollar signs

The general form is PRINT % XY;I
Where X=any combination (or none) of C,$, and Z
Y=any format specification
I=variable or constant
and where the separating comma or semicolon is as in any non-formatted PRINT statement
i.e. PRINT %C$Z12F3;1234.56Ø9
$1,234.561
PRINT %C$Z12F2;1234.56ØØ
$1,234.56

Control-C

Typing the control-C character (ETX on some keyboards) has the effect of prematurely interrupting MAXI 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, and a CONT will continue executing the program.

DIRECT STATEMENTS

When MAXI 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 a pound sign(#) 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

To save a current program onto cassette, the user should turn on his recorder (on record) and type SAVEcr. The CRT screen will indicate that the tape is being written. When finished, the screen will return with the READY message.

To reload a program from cassette, the user should start playing the cassette. When the leader tone is heard, type LOAD cr. The CRT screen will indicate that the tape is being read. When finished the screen will return with the READY message.

All programs written on the Z-8Ø are useable on the 8Ø8Ø version of MAXI-BASIC and vice versa. Also, cassettes written with level 1 MAXI BASIC will be upward compatible on all later levels of MAXI BASIC.

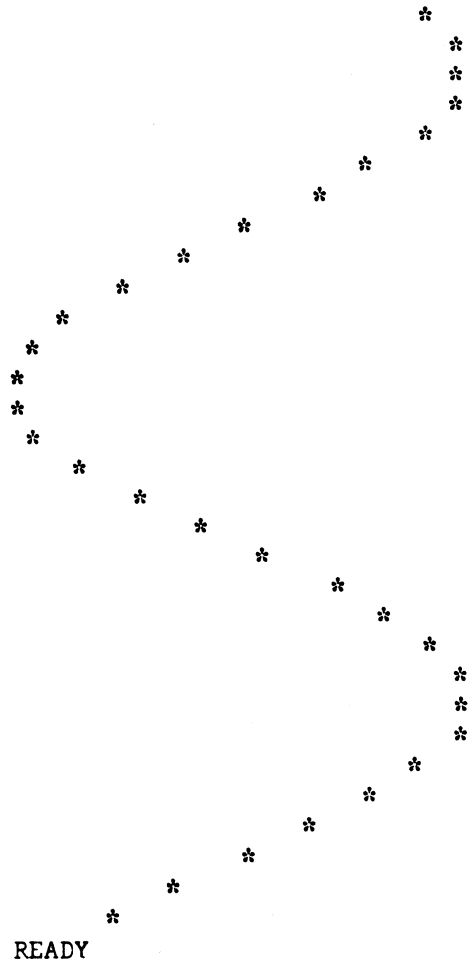
digital group software systems inc.
po box 1086, arvada, colorado 80001

APPENDIX 1

EXAMPLES OF PROGRAMS

```
100 REM   PRINT A VERTICAL SINE WAVE
110 REM
115 FOR J=1 TO 10 STEP .3
120 S=INT(15*(SIN(J)))
140 PRINT TAB(15+S);"*"
150 NEXT J
160 STOP
READY
REN 10,2
READY
LIST

10 REM   PRINT A VERTICAL SINE WAVE
12 REM
14 FOR J=1 TO 10 STEP .3
16 S=INT(15*(SIN(J)))
18 PRINT TAB(15+S);"*"
20 NEXT J
22 STOP
READY
RUN
```



```
READY
```

```
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 J
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
READY
RUN

INPUT FIFTEEN VALUES, ONE VALUE PER LINE
?123
?22
?-37
?0
?2
?-54
?31
?8
?-9.4
?1.54
?-3.8
?36
?21
?-43
?213
SORTED ARRAY:  -54  -43  -37  -9.4  -3.8  0  1.54  2  8  21  22  31  36
 123  213
READY
```

STRING INITIALIZATION

```
10  REM TO INITIALIZE A STRING VARIABLE
20  REM USE THE FOLLOWING ROUTINE
30  REM BEFORE ATTEMPTING TO ALTER THE STRING
40  DIM A$(n)       Where n=string length
50  FOR X=1 to n
60  LET A$=A$+" "
70  NEXT X
```

```
10 REM   CHARACTER SORT
20 REM   EXAMPLE USING STRINGS AND FUNCTION
30 DIM A$(72)
40 INPUT "TYPE A STRING OF CHARACTERS: ",A$
50 IF LEN(A$)=0 THEN 40 : REM MAKE SURE SOMETHING WAS ENTERED
60 IF FNA(LEN(A$))=1 THEN 60 ; REM CALL FNA UNTIL IT RETURNS A ZERO VALUE
70 PRINT"SORTED ARRAY: ";A$
80 END
90 DEF FNA(N) ; REM  CHARACTER SORT
100 REM  RETURN 0 IF A$ SORTED, ELSE RETURN 1
110 LET F=0
120 FOR J=2 TO N
130 IF A$(J-1,J-1)<=A$(J,J) THEN 160
140 T$=A$(J,J) : A$(J,J)=A$(J-1,J-1) : A$(J-1,J-1)=T$
150 F=1
160 NEXT J
170 RETURN F
180 FNEND
READY
RUN

TYPE A STRING OF CHARACTERS: DIGITAL GROUP
SORTED ARRAY:  ADGGIILOPRTU
READY
```

TV DESIGNER

```
ENTER LINE FREQUENCY 60
ENTER HORZ HOLDOFF RATIO 1.5
ENTER VERT HOLDOFF RATIO 1.25
ENTER CHARACTERS/LINE 64
ENTER ROWS OF CHARACTERS 16
ENTER HORZ PEL/CHARACTER 7
ENTER H PEL SPACES 1
ENTER LINES/CHARACTER 12
ENTER LINES DURING JUMP 1
H FREQ 15600   XTAL 11980
VIS H PEL 512  TOTAL H PEL 768
VIS LINES 208  TOTAL LINES 260
H RATIO 1.50  V RATIO 1.25
NS/PEL 83.5  NS/CHAR 667.7
H LINE  64.10  H BLANK  21.37
FRAME 16.67  V BLANK   3.33
        ********
READY
```

```
SCR
READY
LOAD

READY
LIST

10 REM    TV DESIGNER
20 REM
30 FOR A=1TO5 : PRINT"" : NEXTA
40 PRINT TAB(8);"TV DESIGNER"
50 PRINT
60 INPUT"ENTER LINE FREQUENCY ",L
70 INPUT"ENTER HORZ HOLDOFF RATIO ",H
80 INPUT"ENTER VERT HOLDOFF RATIO ",V
90 INPUT"ENTER CHARACTERS/LINE ",C
100 INPUT"ENTER ROWS OF CHARACTERS ",R
110 INPUT"ENTER HORZ PEL/CHARACTER ",P
120 INPUT"ENTER H PEL SPACES ",S
130 INPUT"ENTER LINES/CHARACTER ",X
140 INPUT"ENTER LINES DURING JUMP ",J
150 REM
160 REM        CALCULATIONS
170 REM
180 REM   CORRECTED TOTAL LINES
190 LET A=INT(((X+J)*R*V)+.5)
200 REM   CORRECTED V HOLDOFF RATIO
210 LET V=A/((X+J)*R)
220 REM   HORZ FREQUENCY
230 LET B=A*L
240 REM   VISIBLE H PEL
250 LET D=C*(P+S)
260 REM   CORRECTED EFFECTIVE H PEL
270 LET E=INT((D*H)+.5)
280 REM   CORRECTED HORZ HOLDOFF RATIO
290 LET T=E/D
300 REM   PEL RATE IN MHZ
310 LET F=E*B/1000
320 REM   TIME/PEL IN NS
330 LET G=(1/F)*1000000
340 REM   TIME/CHARACTER IN NS
350 LET I=G*(P+S)
360 REM   HORZ LINE TIME IN US
370 LET K=(1/B)*1000000
380 REM   HORZ BLANKING TIME IN US
390 LET M=K*((E-D)/E)
400 REM   FRAME TIME IN MS
410 LET N=(1/L)*1000
420 REM   VERT BLANKING IN MS
430 LET O=(N*((A-((X+J)*R))/A))
440 REM   TOTAL VISIBLE PEL
450 LET O=D*(P+S)
460 REM   TOTAL VISIBLE LINES
470 LET U=(X+J)*R
480 PRINT"H FREO";INT(B);"   XTAL";INT(F)
490 PRINT"VIS H PEL";D;"  TOTAL H PEL";E
500 PRINT"VIS LINES";U;"  TOTAL LINES";A
510 PRINT"H RATIO ";%4F2;T;"  V RATIO ";%4F2;V
520 PRINT"NS/PEL ";%4F1;G;"  NS/CHAR ";%5F1;I
530 PRINT"H LINE ";%6F2;K;"  H BLANK ";%6F2;M
540 PRINT"FRAME ";%5F2;N;"  V BLANK ";%6F2;O
550 PRINT TAB(10);"********"
READY
```