Micro Basic Users Guide

I originally wrote Micro Basic in 1980 for a University Computer

Club, in which the members were building their own 8085 based systems. I

am including it with CUBIX as a demonstration program for the SIM80

8080 processor simulator utility.

Dave Dunfield

MICRO BASIC USERS GUIDE

TABLE OF CONTENTS

Page

1. INTRODUCTION 1

2. COMMANDS 2

2.1 General commands 2

2.2 Program only commands 5

3. EXPRESSIONS 6

3.1 Numeric operators 6

3.2 Character operators 7

3.3 Numeric conversion 7

3.4 Variables 8

3.5 Special variables 9

4. PROGRAM ENTRY AND EDITING 10

5. CONTROL CHARACTERS 12

6. ERROR MESSAGES 13

7. SOURCE FORMAT 14

8. EXAMPLE PROGRAMS 15

MICRO BASIC USERS GUIDE Page: 1

1. INTRODUCTION

Micro Basic was implemented with the intention of providing the

maximum amount of features and flexibility, in the minimum amount of

memory space. It is intended for use on 8080/8085/Z80 based computers

which are too small to afford the use of larger programming systems.

Currently, the Micro Basic interpreter, is 3K bytes in size.

A minimum of 3K of ram is required if any useful programs are to

be implemented. The interpreter only makes use of memory as it needs

it, and memory can be expanded at any time, to allow for larger

programs or more array space. Micro Basic is quite different from

most other BASIC interpreters, in particular the right to left

execution of expressions, with no operator precedence, and the use

of separate operators for EQUALS and ASSIGNMENT. These implementation

decisions, were in part, based on the language APL, which is a

favorite language of the author.

On the following pages, is a brief description of Micro Basic's

commands and features.

MICRO BASIC USERS GUIDE Page: 2

2. COMMANDS

Operands to commands are as followes:

<e> - Any expression.

<v> - Any variable.

<a> - Any array variable.

<n> - Any numeric expression.

<l> - A line number.

[ ] - Optional operands.

... - Multiple extra operands allowed.

2.1 General commands

The following commands, may be entered directly from the

keyboard, or executed with a BASIC program.

CLEAR

Clears all numeric and character variables, Delete's all

arrays, and resets the control stack and data pointer.

DIM a1(<n1>)[,a2(n2)]...

Dimensions integer arrays a1, a2,... makeing them n1, n2,...

elements each. Arrays may be REDIMENSIONED with the DIM statement,

however this allocates new memory for the array, causing the old

memory used by the old array to be made unusable (until 'NEW',

'CLEAR' or a 'RUN' command is issued). Whenever an array is

dimensioned or redimensioned via DIM, it is cleared to zeros.

NOTE: Array space is allocated in memory, starting at the end of

the program source. As a result, if a line is inserted into

the program, or any line is replaced in the program, any

existing arrays will be deleted.

END

Stops the program. no messages are issued.

EXIT

Terminates BASIC and exits to the operating system.

GOTO <line#>

Transfers program execution to the statement at the beginning

of line <line#>.

GOTO(<n>),<l1>[,<l2>]...

Transfers program execution to the statement at the beginning

of line <l1> if <n>=0, to the beginning of line <l2> if <n>=1,

etc. results in SYNTAX ERROR if <n> is greater than number of line

numbers given.

MICRO BASIC USERS GUIDE Page: 3

INPUT <v>

Requests a value for <v> from the terminal. Prompts with a

question mark "?". If <v> is a character variable, then any text

can be input. If <v> is numeric, then value supplied must be a

number or expression.

INPUT "<text>",<v>

Same as above, but prompts with <text> instead of "?". <text>

can be a null string ( INPUT "",<v> would give no prompt ).

NOTE: The value of a character variable can be used in the prompt,

but must be concatinated with <text>. EG: ' INPUT ""+A$,V '.

LET <v>=<e> (default)

Assigns the value of <e> to the variable <v>. If any lines are

found by the interpreter which do not contain a command, then they

are assumed to be LET.

LIST [<l1>][,<l2>]

Lists the program,if no operands are given, then lists the

entire program. If <l1> is given then only that line is listed. If

<l2> is also given, then lists from <l1> to <l2> inclusive.

LOAD filename

Loads the named file [.BAS] from the disk and makes it the

currently active BASIC program. NOTE: Only .BAS files SAVEd from

BASIC may be LOADed.

NEW

Clears the program, variables, and arrays.

ORDER <line#>

Positions the read pointer to the start of the line <line#>.

This line must begin with a DATA statement, or a DATA ERROR will

occur.

PRINT <e>[,<e>][,<e>]...[,]

Prints the expressions on the terminal. Numeric values will be

printed with a preceding space. If a numeric expression is

preceded by a single '(', then the preceding space is not printed.

( EG: PRINT 12,(12 would display ' 1212' ). If the list of

expressions ends with a trailing comma, then no line-feed carriage

return will be printed, causing the next PRINT statement to

continue at the end of the same line.

MICRO BASIC USERS GUIDE Page: 4

READ <v>[,<v>]...

Reads the values for variables from data statements. An ORDER

statement must be done before the first read in a program, and

anytime that you have read all the data in a data block. A data

block, is a collection of data statements which are located

separately, with no other statements between them. If a read

statement does not read all of the data in a given data statement,

then the next read will pick up where the last one left off. If a

read statement reads beyond the end of a data statement, the it

will advance to the next statement and attempt to read from there.

REM <text>

Comment, the rest of the statement is ignored.

RUN

Clears variables and arrays, then starts the program running. A

running program can be stopped by pressing CONTROL-C.

SAVE filename

Saves the currently active BASIC program to disk under the

specified filename. An extension of .BAS is automatically applied.

The file can be reloaded to memory with the LOAD command.

SIZE

Prints the size of the program in bytes.

STOP

Stops the program, issues message indicating line number where

it was executing.

USR <n1>[,<n2>][,<v>]

Calls a user supplied machine code routine at address <n1>. If

<n2> is given, its value will be passed in the H-L register pair.

If <v> is given, it must be a numeric variable, and will be

assigned the value of H-L after the machine language routine

returns.

MICRO BASIC USERS GUIDE Page: 5

2.2 Program only commands

The following commands, can only be executed within a program.

DATA <e>[,<e>]...

Defines program DATA, to be read by the READ statement, into

program variables. DATA statements are not executed by the

interpreter.

NOTE: Variables can be used in the DATA statements, but the value

will be evaluated as the value of that variable at the time

that the DATA statement is read.

FOR <v>=<n1> TO <n2>

Starts a program loop. The variable <v> will be set to <n1>.and

will be incremented by one every time around the loop. until it

value is equal to <n2>. <v> must be a simple numeric variable. See

also 'NEXT' statement.

GOSUB <line#>

Calls a BASIC subroutine at given line number. (Same as

goto,but stacks return address.) See also 'RETURN' statement.

GOSUB(<n>),<l1>,<l2>...

Same as above, but uses computed line number. See also

'GOTO(<n>)'.

IF <e> THEN <stmt>

Evaluates <e>, If it is true (non-zero) then <stmt> is

executed. If <stmt> is a number, then assumes GOTO <stmt>.

LIF <e> THEN <stmts>

Long IF, same as IF, except that the entire remainder of the

line is executed only if the expression <e> is true.

NEXT <v>

Closes a program loop. <v> must match the <v> in the matching

'FOR' statement.

RETURN

Returns to statement following GOSUB statement. (Terminates a

BASIC subroutine)

MICRO BASIC USERS GUIDE Page: 6

3. EXPRESSIONS

Expressions can be either numeric or character. all expressions,

are evaluated from right to left, with NO operator precedence (as in

the language APL). For example, 1+5\*5 evaluates to 26, but 5\*5+1 will

give the answer 30. Precedence can be forced in numeric expressions

with the use of brackets "()". Brackets can be nested to any depth.

3.1 Numeric operators

FORMAT: "X<operator>Y"

+ Addition.

- Subtraction.

\* Multiplication.

% Division. (Remainder assigned to special variable "R").

& Bitwise logical AND of X and Y.

| Bitwise logical OR of X and Y.

\ Floor. (returns lesser of X and Y).

/ Ceiling. (returns greater of X and Y).

= Assignment. (X takes value of Y).

== Equality. (returns 1 if X equals Y, 0 otherwise).

> Greater than. (returns 1 if X greater than Y, 0 otherwise).

< Less than. (returns 1 if X less than Y, 0 otherwise).

>= Greater equals. (Returns 1 if X GE Y, 0 otherwise).

<= Less equals. (Returns 1 if X LE Y, 0 otherwise).

-= Not equals. (Returns 1 if X not equal to Y, 0 otherwise).

; Null operator, returns value of X. (but executes Y).

Especially useful for doing modular arithmetic.

The expression "A=R;B%123" will divide B by 123 (without

changing B), and then assign the remainder to A. (Right

to left execution).

MICRO BASIC USERS GUIDE Page: 7

3.2 Character operators

FORMAT: "X$<operator>Y$"

+ Concatonation. (Y$ appended to X$).

= Assignment. (X$ takes value of Y$).

== Equality. (only valid in "IF" and Numeric conversion).

-= Not equals. (only valid in "IF" and numeric converson)

The following are other operators which perform useful functions:

( ) Brackets, Force operator precedence.

[ ] Braces, Used to index numeric arrays, E.G. "A[10]"

Also can be used to extract a single character

from a character variable. E.G. "A[0]$" returns

the first character in variable "A$".

(Index starts at zero (0) ).

# Hexidecimal constants. EG. "A=FF#+1" calculates "FF#"

as 255, adds 1 then assigns result (256) to "A".

: Statement separator, can be used to place multiple

statements on a single program line: E.G: "A=10:PRINT A"

, Operand separator, separates operands to some commands.

3.3 Numeric conversion

A character expression can be included in a numeric expression,

but must be contained in brackets "()". If the leftmost operator

in the character expression is one of "==" or "-=", then a 1 or 0

is returned to the numeric (outside) expression. If the leftmost

operator of the character expression is "=",then the value

returned is the ASCII value of the first character of the OLD

value of the character variable. Otherwise the ASCII value of the

first character in the result of the character expression is

returned. The ASCII value of a character, is the decimal value of

it's binary representation. (E.G. " " (blank) is 32). If a null

string ("") is the result of the expression,then the value 255 is

returned (ASCII values for characters can only range from 0 to

127). The expression within the brackets does not have to contain

operators, I.E. " PRINT ("A") " will print a 65. (The decimal

value of an ASCII "A").

MICRO BASIC USERS GUIDE Page: 8

3.4 Variables

There are 26 simple integer variables (A-Z). These variables

always exist and are cleared to zero when BASIC is entered, when a

NEW command is executed, and when a program is RUN. Integers are

positive, with a range of 0 to 65535 (16 bits of data).

There are also up to 26 integer arrays, (A[n] - Z[n]). An array

must be created (via the 'DIM' command) before it exists. Arrays

are cleared to zero's when they are created.

Arrays when dimensioned, (DIM A(n)) have n+1 elements,

subscripts ranging form 0 to n. Subscripts are not checked by the

interpreter, therefore, if you type 'DIM A(10),B(10)' then A[11]

is the same as B[0].

There are 26 character variables, (A$-Z$). These variables

always exist and are cleared to null strings ("") when BASIC is

entered, when a NEW is executed,and when a program is RUN. Each

character variable can hold up to 35 characters. The individual

characters can be read using braces between the character variable

name, and the dollar sign. (ie. A[0]$ to A[34]$). If an index is

greater than 34, a DIMENSION ERROR will result. If an index is

greater than the number of characters currently in the variable,

but less than 35, then a null string ("") will be returned.

Character variables cannot be assigned values in this manner.

The variable names are all separate, you can have A, A[n] and

A$, all in the same program, without interaction between them.

MICRO BASIC USERS GUIDE Page: 9

3.5 Special variables

The simple integer variable 'R' is a special variable because

it will be assigned the remainder whenever a divide operation is

executed.

The following are special variables, unlike 'R', they cannot be

used as 'normal' variables:

@[n] This variable can only be referenced as an array. When read,

it returns the BYTE value (0-255) of the memory location at

its index (n). When assigned a value, that value will be

assigned to the memory location at its index (n). (if the

value assigned is > 255 Then it is divided by 256, and the

remainder is used). This is the Same function as 'PEEK' and

'POKE' in some other BASIC's.

@[n]$ This character variable, can only be referenced with an

index. Its index can range from 0 to 255. It will return the

character which has the binary value of its index. (if 255

is used, it will return a null string. This is the same

function as 'CHR$' in some other basics.

? This variable can only be referenced as a simple integer

variable When read, it returns a random number from 0 to

65535. When given a value, it sets the random seed to that

value. Random numbers can be generated within limits by the

use of modular arithmetic. (EG. to generate a random number

between 0 and 99, and assign it to the variable 'A', use the

command 'A=R;?%100').This is similar to the 'RND' function

of some other basics.

MICRO BASIC USERS GUIDE Page: 10

4. PROGRAM ENTRY AND EDITING

To enter or replace a line, simply enter it's line number starting

in column one, followed by the text for the new line. To delete a

line, just enter it's line number, with no following text.

When a line is entered, (and return is pressed), it is copied into

a buffer. Parts or all of this old line can be included in a new

line, as it is typed in. When the new line is entered,it then becomes

the old line. A pointer is kept, indicating the current position in

the old line. The following functions are available to perform this

'editing'.

CTRL-A

Advance: Copy one character from the old line into the new line,

and advance the pointer to the next character in the old line.

CTRL-C

Cancel: Cancels the (partially) complete new line, and restart

from the beginning. (resets old line pointer).

CTRL-D

Delete: Advances the old line pointer by one character,deleteing

that character from the old line. the new line is not affected. A '\*'

character is printed to indicate this has been done.

CTRL-F

Find: This command requires one extra character to be entered.

When it is, the old line is copied (from the current pointer

position) into the new line, up to but NOT including the first

occurance of that character. The pointer is advanced to point to the

character found. If the character is not found, no action is taken.If

the second character is a carriage return,the remainder of the old

line will be copied into the new line.

CTRL-H

Backup: This backs up one character,deleteing the last character

entered, and backs up the old line pointer in the old line. This

effectively cancels the effect of the last character entered. (The

DELETE key also invokes this function).

CTRL-M

Carriage Return: enters the new line, causing it to become the

(new) old line, and passes it to the interpreter, as input.

MICRO BASIC USERS GUIDE Page: 11

CTRL-I

Insert: Toggles insert mode. a "<" is printed when entering insert

mode a ">" is printed when leaving insert mode. Normally, when you

enter text into the new line, the old line pointer is advanced, so

that the characters you are typing, effectivly replace the characters

in the old line. In insert mode, this does not happen, therefore the

characters you are typing, can be inserted into the old line. (If it

is later copied into the new line).

The line editor can be used to EDIT program lines, When a list

command is executed, the last line listed will be made the old line.

To modify line 50, you would just have to type 'LIST 50'. This would

display line 50, and would also store it in the old line buffer.

Whenever a program stops due to an error, CTRL-C, or a STOP

statement, the line it stopped on will also be stored in the old line

buffer, ready for editing.

MICRO BASIC USERS GUIDE Page: 12

5. CONTROL CHARACTERS

The following control characters (other than the ones recognised

by the line editor) are recognised. All other control characters, are

ignored by the interpreter.

CTRL-C

Will abort any program, terminates with the message "STOP IN LINE

XXXX" where XXXX is the number of the line containing the statement

which would have been executed next. Will also abort program, if

entered when responding to an INPUT statement, but will not print the

"STOP" message. Will also abort output from the LIST command.

The D6809 simulator supports the following display control codes

which can be output with ' PRINT @[value]$' :

^D 4 Forward scroll screen

^E 5 Reverse scroll screen

^G 7 Sound beep

^H 8 Move cursor left

^I 9 Advance to next TAB stop

^J 10 Line-feed : Move cursor down

^M 13 Carraige-return: Move cursor to left margin

^N 14 Reverse video ON

^O 15 Reverse video OFF

^P 16 Position cursor (follow by X & Y, offset by 32)

^U 21 Move cursor forward

^V 22 Clear from cursor to end of line

^W 23 Clear for cursor to end of screen

^X 24 Clear entire screen

^Y 25 Home cursor

^Z 26 Move cursor up

MICRO BASIC USERS GUIDE Page: 13

6. ERROR MESSAGES

Below is a list of error messages produced by the interpreter.

Errors occuring in a program, will be followed by " IN LINE XXXX"

where XXXX is the line on which the error was discovered. All errors

except '?BAD DATA - RETRY' are fatal, and will stop an executing

program.

?SYNTAX ERROR

Results from a statement that is not decodeable. (Does not follow

syntax) Also results if you attempt to use a command in the wrong

context. Ie. You use a command from the keyboard which is only

allowed from within a program.

?NO PROGRAM ERROR

Results from an attempt to RUN or to SAVE an zero line program.

?DIMENSION ERROR

Results from an attempt to index a non-array variable, or from

indexing a character variable with a value greater than 34.

?DIVIDE BY ZERO ERROR

Results from attempt to divide any value by zero.

?LINE NUMBER ERROR

Results from reference to a program line number that does not

exist.

?DATA ERROR

Results from attempt to ORDER to a line which does not start with

a DATA statement, attempt to READ before you have executed an ORDER,

reading beyond the end of a DATA BLOCK, or from reading the wrong

data type (character or numeric) for the operand variable.

?NESTING ERROR

Results from improper nesting of GOSUB/RETURN or FOR/NEXT loops.

?BAD DATA - RETRY

Results from any error in a numeric expression typed as the

response to an INPUT to a numeric variable. Does not stop, but

prompts again.

MICRO BASIC USERS GUIDE Page: 14

7. SOURCE FORMAT

Micro Basic programs are stored in memory, as variable length

records, separated by carriage return character (0D hex). The end of

the program is marked by a line starting with a hexidecimal FF.

Program lines are in the following format:

-------------------------------------------------

|2 bytes|1 byte| variable length section |1 byte|

-------------------------------------------------

\\_\_\_/ \\_\_/ \\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \\_\_\_\_/

^ ^ ^ ^\_ Carriage Return.

^ ^ ^

^ ^ ^\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Program text.

^ ^

^ ^\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Length of remainder

^ of line. (+11 hex)

^

^\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Packed decimal line

number (0000-9999).

(FFxx=end of prog.)

MICRO BASIC USERS GUIDE Page: 15

8. EXAMPLE PROGRAMS

The following are some simple Micro Basic programs, which

demonstrate many of the features of the language. A good excercise to

gain experience with the interpreter, is to enter and run them, and

observe the results.

0010 REM THIS PROGRAM PLAYS THE HIGH/LOW GAME.

0020 PRINT "I WILL PICK A NUMBER BETWEEN 1 AND 100"

0030 PRINT "THEN I WANT YOU TO TRY AND GUESS IT."

0040 PRINT "I WILL TELL YOU IF YOU ARE TOO HIGH, OR TOO LOW"

0050 C=0

0060 N=1+R;?%100

0070 INPUT "WHAT IS YOUR GUESS?",G

0080 C=C+1

0090 LIF G==N THEN PRINT "YOU GUESSED IT IN ",C," GUESSES!":END

0100 IF G>N THEN PRINT "YOU ARE TOO HIGH."

0110 IF G<N THEN PRINT "YOU ARE TOO LOW."

0120 GOTO 70

0010 REM THIS PROGRAM WILL COUNT FROM 1 TO 10, AND DISPLAY

0020 REM THE COUNT, BOTH AS A NUMBER, AND AS A WORD.

0030 ORDER 60 : FOR I=1 TO 10

0040 READ I$ : PRINT I," ",I$

0050 NEXT I

0060 DATA "ONE","TWO","THREE","FOUR","FIVE"

0070 DATA "SIX","SEVEN","EIGHT","NINE","TEN"

0010 REM THIS PROGRAM WILL INPUT N NUMBERS, AND PRINT THEM

0020 REM OUT IN REVERSE ORDER.

0030 INPUT"HOW MANY NUMBERS?",N:DIM A(N):FORI=1TON

0040 PRINT"NUMBER ",I,:INPUT X:A[I]=X:NEXT I

0050 PRINT"NOW HERE THEY ARE BACKWARDS."

0060 FOR I=0 TO N-1:PRINT" ",A[N-I],:NEXTI:PRINT""

0010 REM THIS PROGRAM WILL DISPLAY THE ASCII CHARACTER SET.

0020 FOR I=0 TO 127:PRINT @[I]$:NEXTI:PRINT""

0010 REM THIS PROGRAM WILL DISPLAY THE CONTENTS OF MEMORY

0020 REM FROM 0000 TO 07FF IN DECIMAL.

0030 FOR I=0 TO 7FF# : PRINT @[I],:NEXT I : PRINT ""

0010 REM THIS PROGRAM WILL DISPLAY THE CONTENTS OF MEMORY

0020 REM FROM 0000 TO 07FF IN ASCII.

0030 FOR I=0 TO 7FF# : PRINT @[@[I]]$," ", : NEXT I : PRINT ""

0010 REM THIS PROGRAM WILL INPUT A NUMBER, AND PRINT IT IN HEX.

0020 REM NOTE THE USE OF MOD. ARITHMETIC, AND CHAR. VARIABLE INDEX.

0030 R$="":H$="0123456789ABCDEF":INPUTN

0040 R$=H[R;N=N%16]$+R$:IFN>0THEN40

0050 PRINT R$