The Basic816 Language Manual

PJW

20 September 2019

1 About Basic816

Basic816 is an implementation of the BASIC programming language for the Western Design Center's 65816 microprocessor. It is focused on providing a simple BASIC interpreter for the C256 Foenix computer designed by Stefany Allaire. Basic816 has a few design goals and even a couple of anti-goals:

- It should provide a retro-computing feel.
- It should be simple to use and easy to learn.
- It should provide essential access to storage and the C256's abilities.
- It should be expandable, allowing advanced users to customize it or extend it.
- It should be a clean-room implementation, unencumbered by copy-right.
- It need not be the fastest programming language available.
- It need not provide all the advancements in programming languages developed since the 1980s.

As such, Basic816 is a fairly traditional, tokenized, line-number based implementation of BASIC, similar to those implementation of BASIC on the 8-bit computers of the 1970s and 1980s.

2 Data Types

Basic816 supports three data types:

Integer The integers in Basic816 are 32-bit signed integers and may have values between 2,147,483,648 and 2,147,483,647. Examples of integers are 0, 1, 42, -1, -128.

String Text data are stored in strings. A string may be entered into a BASIC816 program by enclosing it in double quote marks. Strings are represented internally as null-terminated ASCII strings, that is, there is no length data on the string, the end of which is marked by a 0 byte. Strings may be up to 65536 bytes in length.

Array Arrays allow multiple data items to be stored under a single name and accessed by an index. Two types of arrays are supported: arrays of integer and arrays of strings. Arrays may be just one-dimensional, using a single number as index, or they may be multi-dimensional, where each element has a unique combination of numbers serving as an index.

Note: when BASIC816 leaves the alpha stage of development, it will include a fourth type: 32-bit floating point numbers as well as arrays of 32-bit floating point numbers.

3 Variables

A BASIC816 program can assign data to variables using the LET statement or an implicit LET statement. Variables need not be declared before use, with the exception of array variables, which must be declared through the DIM statement.

Variable names may be up to eight characters long. The first character must be an alphabetic character from A through Z. Subsequent characters of the name may be A - Z, 0 - 9, or the underscore character (_). The type of the variable is indicated by a character at the end of the variable name. Integer variables are indicated by the percent sign (%), and string variables are indicated by the dollar sign ($). These type designations must be used in all references to the variable and may be considered as part of the name. This means that a program can have six different variables with the same "name" but different types (e.g. `A`, `A%`, `A$`, `A()`, `A%()`, and `A$()`).

4 Expressions

An expression is a sequence of values, operators, and function calls which will have some data value as a *result*. An expression may have a numeric (integer) result or a text (string) result.

Operators are common mathematical symbols which will perform some sort of computation on two values. These are the usual sort of thing: addition, subtraction, multiplication, division, and so on. Operators are typically evaluated left to right, but have an operator precedence which can alter the order of execution. For instance multiplication operators are evaluated before addition. This

Operators	Purpose
^	Exponent
*, /, MOD	Multiply, Divide, modulo
+, -	Addition, Subtraction
<, >, = , <=, >=, <>	Comparison operators
NOT	Bitwise Negation
AND	Bitwise AND
OR	Bitwise OR

Table 1: Operators in order of descending precedence.

precendence can be altered by using parentheses to enclose a sub-expression that should be evaluated as a unit.

5 Commands

Commands in BASIC816 are keywords which triggers some action but which may only be used at the interactive prompt. Commands may not be used within a BASIC816 program.

5.1 CONT

Continue execution of the current program from the point immediately after the STOP statement that was executed. It is an error to use this command if the current program was not interrupted by the STOP command or if there is no current program.

5.2 NEW

NEW clears all of BASIC816's memory. All program and variable data are erased, and a new program may be entered.

5.3 LIST [<start>] [- <end>]

LIST types a program listing to the console screen. It accepts two optional line-numbers to limit the listing. The first line number specifies the smallest line number to list; if it is not specified, there is no lower limit. The second line number specifies the largest line number to list; if it is not specified, there is no upper limit.

5.4 RUN

Starts execution of the program from the first line of the code.

6 Statements

Statements are keywords which trigger some action and may be used either at the interactive prompt or within a program. A BASIC816 program can be seen as a series of lines, each of which must have one or more statements on it. If a line has more than one statement, each statement is separated from the next by a colon (:).

6.1 CLR

Erases all variable definitions. Any variable that was defined prior to the use of `CLR` will be undefined. Also all data stored directly or indirectly through variables will be returned to free memory. This is similar to `NEW`, except that the program may be running and is left intact.

6.2 CLS

Clears the text screen and moves the cursor to the home position.

6.3 CALL <address>[, <a>[, <x>[, <y>]]]

Call a machine language subroutine at `<address>`. If the subroutine returns at all to the caller, it must return using the `RTL` instruction. Values may be provided for the `A`, `X`, and `Y` registers and may be 16-bit values.

6.4 DATA <value>[, ...]

Provide numeric or string data in the program which may be retrieved into a variable using the `READ` statement.

6.5 DIM <variable>(<size>[, ...])

Declare an array named `<variable>`. The array can have many dimensions, each with the size provided. An array can have up to 127 dimensions, and the size can be up to 256, but in practice the entire block of memory consumed by the array cannot exceed 65536 bytes (including the "book keeping" memory allocated to keep track of the array (which is at most 256 bytes).

6.6 END

Stops execution of the program. The only way to restart execution after END has been executed is to use the RUN command.

6.7 FOR <variable> = <initial> TO <target> [STEP <increment>]

The FOR statement marks the beginning of a loop that will repeat a specified number of times. It starts by assigning an <initial> value to a <variable> and executing the following statement until the matching NEXT statement is encountered. It will then either add 1 to <variable> or <increment>, if provided. So long as <variable> is not <target>, the statements between the FOR and NEXT will be executed.

6.8 GOTO <line number>

Continue execution with the first statement on the line with the given <line number>. It is an error to GOTO a <line number> that does not exist in the program.

6.9 GOSUB <line number>

Call a subroutine starting at the line with the given <line number>. A subsequent RETURN statement will return the program to the statement following the GOSUB. It is an error to GOSUB a <line number> that does not exist in the program.

6.10 IF <expr> THEN <line>

Evaluates <expr> and examines the result. If the result is not zero, execution continues with the first statement on the line with number <line>. Otherwise, execution continues with the next statement after the IF.

Note: This statement will be receiving considerable improvements in subsequent versions of BASIC816.

6.11 NEXT

Close the matching FOR loop.

6.12 POKE <address>, <value>

Write the 8-bit <value> to the memory location at <address>. It is an error to try to write a value that requires more than 8-bits.

6.13 POKEW <address>, <value>

Write the 16-bit <value> to the memory location at <address>. The low byte of <value> will be written to <address>, and the high byte of <value> will be written to the following byte in memory. It is an error to try to write a value that requires more than 16-bits.

6.14 POKEL <address>, <value>

Write the 24-bit <value> to the memory location at <address>. The low byte of <value> will be written to <address>, and the middle byte of <value> will be written to the following byte in memory, and the high byte of <value> will be written to the next byte in memory. It is an error to try to write a value that requires more than 24-bits.

6.15 PRINT [<value> [,/;]] ...

Write the textual representation of <value> to the screen. If more than one <value> is provided, they must be separated by either a comma (,) or a semicolon (;). If a comma is used, the two items will be separated by a TAB. If a semicolon is used, the two items will be printed one after the other. A PRINT statement will print a carriage return as the last thing, unless the statement is ended with a semicolon.

6.16 REM <comment>

Inserts a comment into the BASIC816 program. All characters after the REM until end of the line will be ignored.

6.17 READ <variable>[, ...]

Read one of more values out of the DATA statements in the program into <variable>. The data read must have a compatible type to the variable. Each variable read will advance a data cursor forward one data element. If a READ is executed when the cursor has reached the end of the data elements, it is an error.

6.18 RESTORE

Resets the data cursor to the first data element of the first DATA statement.

6.19 STOP

Stops execution of the program in such a way that the user can restart it with the CONT command.

7 Functions

7.1 ABS(<value>)

Returns the absolute value of <value>. If the parameter is negative, it is converted to its positive equivalent (for instance, ABS(-5) will evaluate to 5).

7.2 ASC(<text>)

Returns the ASCII code for the first character of <text>. Example: ASC("A") returns 65.

7.3 CHR$(<value>)

Returns the character corresponding to the ASCII code in <value>. Example: CHR$(65) returns "A".

7.4 DEC(<hex>)

Returns an integer that is conversion of the hexadecimal number in the string <hex>. Example: DEC("A0") returns 160.

7.5 HEX$(<value>)

Returns a string that is the hexadecimal representation of the integer value passed. Example: HEX$(160) returns "A0".

7.6 LEFT$(<text>, <count>)

Returns the left-most <count> characters of the string <text>.
Example: LEFT$("Hello", 3) returns "Hel".

7.7 MID$(<text>, <first>, <count>)

Returns a substring of the string <text>. The parameter <first> specifies the number of the first character to use, where a 0 is the number of the first character in the source string. The parameter <count> indicates how many characters should be returned.

Example: MID$("Hello", 2, 3) returns "llo".

7.8 PEEK(<address>)

Returns the byte stored in memory at location <address>.

7.9 PEEKW(<address>)

Returns the 16-bit word stored in memory at location <address>. The low byte of the returned value is at <address>, and the high byte is at address + 1.

7.10 PEEKL(<address>)

Returns the 24-bit word stored in memory at location <address>. The low byte of the returned value is at <address>, the middle byte is at address + 1, and the high byte is at address + 2.

7.11 RIGHT$(<text>, <count>)

Returns the right-most <count> characters of the string <text>.
Example: LEFT$("Hello", 3) returns "llo".

7.12 SGN(<value>)

Returns the sign of the number <value>. If the number is negative, the result is -1, if the number is positive, the result is 1, and if the number is zero, the result is 0.
Example: SGN(-25) returns -1.

7.13 SPC(<value>)

Returns a string containing <value> spaces.
Example: SPC(5) returns a string of five spaces.

7.14 STR$(<value>)

Returns a string containing the decimal representation of the number <value>.
Example: STR$(25) returns "25".

7.15 TAB(<value>)

Returns a string containing <value> TAB characters.
Example: TAB(2) returns a string of two TABs.

7.16 VAL(<text>)

Returns the numeric value represented by the string of decimal digits in <text>.
Example: VAL("42") returns 42.

8 C256 Specific Statements

BASIC816 includes a number of statements to support features of the C256 Foenix.

8.1 MONITOR

Enter the machine language monitor.

8.2 SETBRDCOLOR <red>, <green>, <blue>

Sets the color of the border of the C256's screen, given <red>, <green>, and <blue> intensity values, which must be between 0 – 255.

8.3 SETBGCOLOR <index>

Sets the color for the background of text to be printed on the screen given an <index> into the color lookup table. The index must be between 0 – 255.

8.4 SETDATE <day>, <month>, <year>

Sets the date on the C256's real time clock, given the date as three numbers: <day>, <month>, and <year>. The day number must be from 1 – 31. The month number must be from 1 – 12. The year number must be from 0 – 99.

8.5 SETFGCOLOR <index>

Sets the color for the foreground of text to be printed on the screen given an <index> into the color lookup table. The index must be between 0 – 255.

8.6 SETTIME <hour>, <minute>, <second>

Sets the time on the C256's real time clock, given the time as three numbers: <hour>, <minute>, and <second>.

9 C256 Specific Functions

9.1 GETDATE(0)

Returns the current date from the C256's real time clock as a string in "DD:MM:YY" format.

9.2 GETTIME(0)

Returns the current time from the C256's real time clock as a string in "HH:MM:SS" format.