

# BasTo6809 User Manual

## Introduction

**BasTo6809** is a compiler that converts a BASIC program into 6809 Assembly Language, designed to run on the TRS-80 Color Computer. The assembly code generated by BasTo6809 is ready for use with **LWASM**, allowing you to assemble and execute machine language programs on your CoCo.

This tool is ideal for anyone looking to take their BASIC programs and convert them to a lower-level language for faster execution or to speed up development of assembly language code.

## Version Information

**BasTo6809 Version:** 2.12

**Author:** Glen Hewlett

**GitHub:** [BASIC-To-6809](https://github.com/glenhewlett/BASIC-To-6809)

## Usage

**BasTo6809** [options] **program.bas**

Where program.bas is the BASIC program you wish to convert to 6809 assembly language.

By default, the compiler will output a fully commented assembly language file (program.asm) that can be processed by LWASM to produce a machine code program for the Color Computer.

## Command Line Options

BasTo6809 provides several command-line options to customize the behaviour of the compiler:

### **-coco**

Use this option if your input is a tokenized Color Computer BASIC program.

### **-ascii**

Use this option for a plain text BASIC program written in ASCII format, such as a program created with a text editor or QB64.

## **-SXXX**

This option sets the maximum length to reserve for strings in an array. The default (and maximum) is 255 bytes. If your program uses smaller strings, setting this value can reduce the amount of RAM your program uses.

Example: -s128 reserves 128 bytes for each string.

## **-OX**

Controls the optimization level during the compilation process:

-o0 disables optimizations (not recommended).

-o1 enables basic optimization.

-o2 (default) enables full optimization for the fastest and smallest possible code.

## **-bx**

Optimizes the branch lengths, affecting how efficiently LWASM assembles the program.

-b0 (default): Some branches may be longer than necessary,

resulting in larger/slower programs.

-b1 Ensures all branches are as short as possible, producing smaller  
& faster programs, but will slow down the assembly process.

## **-pi**

Specifies the starting memory location for the program in hexadecimal. Useful if you need some extra space reserved for your own program. The default starting location for the compiled program is \$2600 which is the memory space after the first PMODE graphics screen.

Example: -p4000 sets the starting address at \$4000.

## **-v**

Displays the version number of BasTo6809.

## **-Vx**

Sets the verbosity level of the compiler output.

-v0 (default) produces no output during compilation.

-v1 shows basic information while compiling.

## **-k**

Keeps miscellaneous files generated during the compilation process.

By default, these files are deleted, leaving only the .asm file.

**-fxxxx**

Where xxxx is the font name used for printing to the PMODE 4

screen (default is Arcade). Look in folder

Basic\_Includes/Graphic\_Screen\_Fonts to see font names available

**-h**

Displays a help message with information on how to use BasTo6809.

## Cool things the Compiler can do

- You can write the program on a CoCo or on a modern computer using any text Editor
- Use of line numbers is optional
- You can use Labels for sections of code to jump to (case sensitive)
- Variable names can be 30 characters long (case sensitive)
- Doesn't use any ROM calls, possible to use all of the 64k of RAM
- The assembly code generated is fully commented showing each BASIC line and how it is compiled. This assembly file can be used to help someone learn how to program in assembly language. Or allow an experienced assembly programmer to optimize the program by hand.

## New commands or features added to BASIC

- Use IF/THEN/ELSE/ELSEIF/ENDIF
- Use SELECT/CASE
- Use WHILE/WEND
- Use DO/WHILE/LOOP
- Use DO/LOOP/UNTIL
- PRINT #-3, Command allows you to print text on the PMODE 4 screen  
Use LOCATE x,y command to locate where on the PMODE screen to print the text.
- PUT Command can use all the usual options like PSET (default if no option is used),PRESET, AND, OR, NOT now can use XOR which XORs the bits on screen with the bits of the GET buffer as:  
99 PUT(c,y)-(c+4,y+9),Sprite1,XOR



## How to use

The compiler is called BasTo6809 and is written in BASIC specifically [QB64pe](#) (Phenix Edition). [QB64pe](#) is multi-platform so BasTo6809 can be used on a Mac, Linux or Windows machine. You'll need to compile it using [QB64pe](#).

Once you have BasTo6809 compiled using [QB64pe](#) you'll need to have it along with a specific folder called Basic\_Includes of .asm libraries that the compiler will add (if needed) to the output.asm file it generates from the source .BAS file.

The last thing you will need to do is install [lwasm](#) on your computer as this is the assembler that is needed to turn the assembly output from my compiler into the final machine language program.

Once you're compiling folder is setup it's fast and easy to compile your BASIC program to machine language using the following commands:

Using MacOS or Linux:

```
./BasTo6809 -ascii BASIC.bas  
lwasm -9bl -p cd -o./ML.bin BASIC.asm > ./Assembly_Listing.txt
```

Using Windows:

```
.\BasTo6809 -ascii BASIC.bas  
lwasm -9bl -p cd -o./ML.bin BASIC.asm > ./Assembly_Listing.txt
```

At this point you'll have an EXECutable program called ML.bin in the folder that you can use on a real CoCo or an emulator.

## Get your system ready to compile BASIC programs for the CoCo

- 1) – Install and test [QB64pe](#) so you're familiar with how it compiles BASIC program
- 2) – Install [LWASM](#) on your system

Once those two programs are installed on your system and you've used QB64 to compile BasTo6809.bas, BasTo6809.1.Tokenizer.bas and BasTo6809.2.Compile.bas along with cc1sl.bas, you're ready to go.

Make sure your working directory now has the BasTo6809 executable, BasTo6809.1.Tokenizer executable, BasTo6809.2.Compile executable and the cc1sl executable program and the sub folder called "Basic\_Includes" which has .asm files and more in it.

With that all setup you're now good to go, here's an example of how to compile a basic program called HELLO.BAS from the command line:

From MacOS or Linux:

```
./BasTo6809 HELLO.BAS
```

From Windows:

```
.\BasTo6809.exe HELLO.BAS
```

If the compiler doesn't report any errors you should now have a file saved in your directory as HELLO.asm you can look through the .asm file with any text editor to see the assembly code the compiler created. It generates an assembly language file with a lot of comments.

Next step is to use LWASM to assemble the program into a CoCo executable program, something like this:

```
lwasm -3b1 -p cd -o./HELLO.BIN HELLO.asm > ./NEW_Assembly_Listing.txt
```

This will create an output file called HELLO.BIN that you can now take and use on a real CoCo or an emulator and execute.

## Optimizing

To generate the fastest and smallest version of your program use this command (LWASM will take awhile to assemble so be patient, could be a minute or so and **it may seem like nothing is happening**):

For MacOS and Linux:

```
./BasTo6809 -b1 HELLO.BAS
```

For Windows:

```
.\BasTo6809 -b1 HELLO.BAS
```

The only other thing you might need to do if you have a program that is very big is use the cc1sl program. The steps for compiling a big program are:

For MacOS and Linux:

```
./BasTo6809 -b1 HELLO.BAS
```

```
lwasm -9b1 -p cd -o./HELLO.BIN HELLO.asm > ./NEW_Assembly_Listing.txt  
./cc1sl -l HELLO.BIN -oBIGFILE.BIN
```

For Windows:

```
.\BasTo6809 -b1 HELLO.BAS
```

```
lwasm -9b1 -p cd -o./HELLO.BIN HELLO.asm > ./NEW_Assembly_Listing.txt  
.\cc1sl -l HELLO.BIN -oBIGFILE.BIN
```

In this case your final program to execute on the CoCo is called BIGFILE.BIN, you can of course call it whatever you want. Remember to only use cc1sl if your file is fairly big. I remember testing it with small programs and it seemed to not work. I never did look into why at least as of yet. But it works perfect if you do have a large program.

The latest version of the compiler can be found on my [GitHub site](#). For support, ask for help on the [CoCo Nation basic compilers Discord channel](#)

## 64k programs

If your program requires more than 32k you must use the cc1sl program (CoCo 1 Super Loader). This program enables the loading of an ML program no matter where it will be loaded into RAM including where the BASIC ROMs are located.

```
cc1sl - CoCo 1 Super Loader v1.03 by Glen Hewlett
Usage: cc1sl [-l] [-vx] FILENAME.BIN -oOUTNAME.BIN
[.scn] or [.csv]...
```

Turns a CoCo 1 Machine Language program into a loadable program no matter if it over writes BASIC ROM locations and more

Where:

-l Will add the word LOADING at the bottom of the screen while the program loads

-vx Amount of info to display while generating the new file x can be 0, 1 or 2. Default x=0 where no info is shown

FILENAME.BIN is the name of your big CoCo 1 program, it must end with .BIN

OUTNAME.BIN is the name of the output file to be created otherwise it defaults to GO.BIN

\*.scn A binary file that must end with .scn will be shown on the CoCo text screen while loading

\*.csv A csv text file that must end with .csv will be shown on the CoCo text screen while loading

For more info see the cc1sl\_help.txt file

cc1sl.bas is also a QB64pe program cc1sl.bas that you must compile with QB64pe before using.

## Supported BASIC commands

AUDIO ON/OFF

BUTTON

CIRCLE

CLEAR - Only clears all the variables to zero

CLS

CASE

DATA

DEF FN

DIM

DO (WHILE/UNTIL)

DRAW

ELSE

ELSEIF

END

END IF

END SELECT

EVERYCASE

EXEC

EXIT (DO, FOR, WHILE)

FOR/NEXT

GET

GETJOYD (new command which returns values of 0,31 or 63 for both joysticks in both Vert & Horizontal movement) Quickly get the joystick values of 0,31,63 of both joysticks both horizontally and vertically

Results are stored same place BASIC normally has the Joystick readings:

**LEFT LEFT RIGHT RIGHT**

**VERT HORIZ VERT HORIZ**

**\$15A \$15B \$15C \$15D**

GOSUB

GOTO

IF

INPUT

LET

LINE

LOADM

LOCATE - Sets the cursor for text on the PMODE 4 screen LOCATE x,y  
Where x is 0 to 31 and y is 0 to 184

LOOP (WHILE/UNTIL)

MOTOR ON/OFF

NEXT  
ON GOSUB  
ON GOTO  
PAINT  
PCLS  
PLAY  
PMODE – Only PMODE 4  
POINT  
POKE  
PRINT – Can't do PRINT USING, Use PRINT #-3,"Hello" to print on  
PMODE 4 screen  
PRESET  
PSET  
PUT  
READ  
RESET  
RESTORE  
RETURN  
SCREEN  
SELECT  
SET  
STEP  
STOP  
SOUND  
TAB()  
TIMER  
UNTIL  
WHILE/WEND  
WPOKE

### **Numeric Commands it can handle**

ABS()  
ASC()  
BUTTON()  
FN()  
INSTR([start],Basestring,SearchString)  
INT()  
JOYSTK()  
LEN()

PEEK()  
POINT()  
PPOINT()  
RND() - Fast random number generator value of 2 to 255  
RNDL() - Random number value of 2 to 32767  
RNDZ() - Better randomness value of 2 to 255  
SGN()  
SQR()  
VAL()  
WPEEK()

### **String Commands it can handle**

CHR\$()  
HEX\$()  
INKEY\$  
LEFT\$()  
MID\$()  
RIGHT\$()  
STR\$()  
STRING\$()

### **Logical operators it can handle**

AND  
OR  
XOR  
NOT

### **Math operators**

- +, -, \*, /, ^, MOD = remainder, DIVR same as / except the result is rounded to the nearest value. For compatibility it accepts \ as integer division (which is the same as /)

## GET dimension size calculation

To calculate the size of the array space for your GET/PUT buffer use the following formula:

First dimension in the array is calculated with this formula:  
 $(\text{INT}(\text{Width in pixels}/8)+3)*8$

Second dimension in the array is simply the number of rows in your sprite

For example, if you have a sprite that is 15 pixels wide and 9 rows high such as:

GET(0,0)-(14,8),Sprite1,G

The calculation for the needed space is:

$$(\text{INT}(15/8)+3)*8 = 32$$

The DIM command for this array would be:

DIM Sprite1(32,9)

\*\*\* If the calculated value for the first dimension of your GET buffer array is larger than 254 you will need to use these values for your array

$$(\text{INT}(\text{Width in pixels}/8)+3) * 4 , \text{Height in Pixels} * 2$$

GET(0,0)-(255,3),Sprite1,G

The calculation for the needed space is:

$$(\text{INT}(256/8)+3)*4 = 140 , 4 * 2 = 8$$

The DIM command for this array would be:

DIM Sprite1(140,8)

The reason so much space is needed for sprites is because the GET command preprocesses the sprite data and saves bit shifted versions in the array space that are ready to be PUT on the screen as fast as possible. This means the sprites will be just as fast on a byte boundary as it is on any other pixel.



## Limitations of the Compiler

- Numeric variables are 16 bit signed values (No floating point numbers, Speed is the priority)
- Other than support for LOADM it can't handle Disk access (Maybe in the future, maybe I'll also add CoCoSDC file I/O)
- Arrays can only have up to 255 elements per dimension, such as DIM ArrayName(255), but they can also be multidimensional, like ArrayName(10,10,2,3)
- No PLAY command support
- Only supports PMODE 4 graphics and graphic commands
- CIRCLE command can only draw complete circles, you can't squeeze them or draw an arc

## Error Handling

If your program isn't compiling, a lot of the times it's because the compiler is having a hard time parsing the program. Usually making sure you have spaces between commands and variables and operators and variables.

You can sometimes figure out what is causing the problem from the error message and line number given where the compiler found the problem. If you don't use line numbers in your program then the error message won't be able to give you the correct line the error occurred.

Also looking at the end of the actual .asm file it created might help to see what the compiler is trying to parse and failed.

## Thanks

I'd like to thank Scott Cooper (Tazman) for initial testing of the compiler. I also like to thank others on Discord who inspired me to keep adding new features, including Bruce D. Moore, Erico Monteiro.