BasTo6809 User Manual

BasTo6809 is a floating point & Integer compiler that converts a BASIC program into 6809 Assembly Language, designed to run on the TRS-80 Color Computer and compatible computers.

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.

BasTo6809 Version: 5.1

Author: Glen Hewlett

Latest version can be found on my GitHub page
GitHub: BASIC-To-6809

Support
Discord: CoCo Nation basic-to-6809 Discord channel

Table of contents

Quick Start

1) Unzip the BASIC_To_6809 compressed files on your computer

2) From the terminal go into the new BASIC_To_6809 folder

3) If you are not running MacOS skip to step 4, otherwise you will probably need to bypass the gatekeeper security features so the executable programs will run on your Mac.

   Run the following command line:
xattr -d com.apple.quarantine * 2>/dev/null

4) Start the Super Duper Extended Color Basic IDE with the command:

./SDECB

Windows users might need to use:
SDECB.exe

5) This starts up the IDE where you can type in or copy and paste BASIC programs. The IDE is very useful for finding typos with your program before you even try to compile it.

5) Load the TEST.BAS program from the File menu

6) Compile the program from the Compile Menu and click on one of the "Compile using lines". These lines are associated with the options in the makefile (compile.bat on Windows) that is in the BASIC_To_6809 folder.

The makefile (compile.bat on Windows) is split up into different sections and is totally customizable by the user. The first section is **defaultF11**:

These commands will be executed when you press F11 or select from the SDECB IDE compile menu. The makefile comes setup to compile the program which creates a 6809 assembly file, the next step is to assemble the 6809 .asm code to a CoCo binary using William Astle's excellent lwasm. Next step is to copy the BLANK.DSK disk image to a new disk

image called DISK1.DSK and lastly it uses Boisy Pitre's decb command to copy the user's program to the DISK1.DSK.

From this point you can copy the DISK1.DSK to an SD card and use it on a real CoCo using the CoCoSDC. Or add some lines to the makefile/ compile.bat to startup an emulator and load the DISK1.DSK file.

Your compiled BASIC program can be executed by using the the usual commands on the CoCo:
LOADM"TEST":EXEC

Once you are able to make the TEST.BIN file you're ready to make your own BASIC files in the SDECB IDE and compile them.

Have Fun!

Command Line Options

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

All these options can be set and will take precedence if the first line of your .bas file has a line starting with (example):
`'CompileOptions: -fArcade_B0_F1 -o -s32`

-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.

-dragon
Compiled output will work on a Dragon computer
See these details for more information

-a
Makes the program autostart after it is loaded

-v
Displays the version number of BasTo6809.

-ox
Controls the optimization level during the compilation process:
-o0 disables optimizations (not recommended).
-o1 (default) enables full optimization for the fastest and smallest possible code.

-pxxxx
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 $0E00.
Example: -p4000 sets the starting address at $4000.

-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 will dramatically reduce the amount of RAM your program uses.
Example: -s32 reserves 32 bytes for each string.

-fxxxx

Where xxxx is the font name used for printing to the graphics screen (default is Arcade_B0_F1). Look in folder Basic_Includes/GraphicsMode/Graphic_Screen_Fonts to see font names available

-Vx

Sets the verbosity level of the compiler output.
-v0 (default) produces no output during compilation.
-v1 shows basic information while compiling.
-vx x=2,3 or 4 more info is displayed while compiling

-k

Keeps miscellaneous files generated during the compilation process. By default, these files are deleted, leaving only the .asm file.

-h

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

Enhancements to Colour BASIC

– Write BASIC programs using the helpful included IDE.

– New **GMODE** command allows you to choose every graphic mode the CoCo can produce, including semi graphics and if using a CoCo 3 all of the CoCo 3 graphics modes and Colour modes. Using these new screens you can use LINE (with B & BF), CIRCLE and PAINT commands.

– Can print directly to any graphic screen using PRINT #-3,

– Use of line numbers is optional

– You can use Labels for sections of code to jump to (case sensitive)

– Variable names can be 25 characters long (case sensitive)

– Doesn't use any ROM calls (Extended BASIC ROM not required), possible to use all of the 64k of RAM on Coco 1 & 2

– Added new sprite commands

– Added commands for the CoCo 3 to use scrollable playfields (backgrounds)

– Added new audio sample commands

– Many new SDC related commands allow you to read and write directly to the SDC filesystem from your BASIC program

– A new SDC audio streaming command to play RAW music files/audio samples directly from the SD card in the CoCoSDC

– Easily insert your own assembly code anywhere you want in your program and easily share values of variables between BASIC and your assembly code.

– Added keyword Const which let's you label a value and use the value in your program, but doesn't take any RAM in your final compiled program.

– The assembly language code generated is fully commented showing each BASIC line and how it's compiled. The assembly file generated 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.

Changes to BASIC's Graphic features

PMODE has been replaced by the **GMODE** command
PCLS has been replaced by the **GCLS** command
PCOPY has been replace with the **GCOPY** command
LINE command format has been changed to include a colour value and no longer uses **PSET** and **PRESET**.

The commands **PSET, HSET, PRESET, HRESET, PPOINT** or **HPOINT** are not supported. Instead they are replaced with **SET** and **POINT** commands. The compiler will use whichever graphic mode is set using the GMODE command and will SET pixels to the requested colour the user wants that matches the GMODE requested.

You can now use SET,POINT,LINE,CIRCLE & PAINT commands on every screen, even the regular text screen, using GMODE 0,1

GMODE ModeNumber,GraphicsPage

Selects the graphics screen and the graphics page.
ModeNumber is the graphics mode you want to use
GraphicsPage is the Page you want to show/use for your graphics commands
To see a list of ModeNumbers and the resolutions **go here**

Special note the **ModeNumber** must be an actual number and cannot be a variable as the compiler needs to know exactly which graphic mode commands to be included at compile time.
GraphicsPage can be a variable.
If you are going to use Graphic pages, the compiler needs to know how many pages to reserve in RAM (for CoCo 1 & 2 graphics). So you must have a GMODE #,MaxPages entry at the beginning of your BASIC program. Where the value of MaxPages will be an actual number and not a variable.

GCLS #

Colour the graphics screen
is the colour value you want the screen to be coloured

GCOPY SourcePage,DestinationPage

Makes a copy the Source graphics page to the Destination graphics page.

SourcePage - Source graphics page

DestinationPage - Destination graphics page

SET(x,y,Colour)

Sets a pixel on the screen

x,y - Screen location of the pixel to be drawn

Colour - Colour Number of the pixel to be drawn

POINT(x,y)

Returns the colour value of the pixel selected

x,y - Screen location of the pixel value requested

LINE(x0,y0)-(x1,y1),Colour[,B][F]

x0,y0 - Starting location

x1,y1 - Ending location

Colour - Colour of the Line or Box to draw

B - Draw a Box

F - Fill the Box

PAINT(x,y),OldColour,FillColour

Fills the old colour value with the fill colour value which must also be the border colour of the section you are painting

x,y - Starting location

OldColour - Colour Number

FillColour - Colour Number

CIRCLE(x,y),Radius,Colour

Draws a circle on the screen

x,y - Origin of the circle

Radius - Size of the circle, to keep the aspect ratio close to round

Some of the graphics modes use scaling so the Radius isn't always a count of actual pixel values.

Colour - Colour Number of the circle to draw

PALETTE v,Colour

Sets the CoCo 3 Palette value

v -palette slot of 0 to 15

Colour - Colour value of 0 to 63

GET & PUT commands are not available

New commands and features added to Color BASIC

– IF/THEN/ELSE/ELSEIF/ENDIF
– SELECT/CASE
– WHILE/WEND
– DO/WHILE/LOOP
– DO/LOOP/UNTIL

– COCOHARDWARE a Command that let's you identify which CoCo your program is running on

– COCOMP3 Commands See here for more info

– SDC_PLAY Command that plays an audio sample or song directly off the SD card in the SDC Controller. See here for more info

– SDC_PLAYORCL, SDC_PLAYORCR, SDC_PLAYORCS these commands are similar to SDCPLAY except the audio is sent to the Orchestra 90 or COCOFLASH cartridge. See here for more info

– SDC file access commands that allow you to Read & Write files directly on the SD card's own filesystem. See here for more info

– GETJOYD - Quickly get the joystick values of 0,31,63 of both joysticks both horizontally and vertically

- PRINT #-3, - Print to the graphics screen created with the GMODE command. Can also be used as ?#-3,"Hello World!". The default font is ArcadeArcade_B0_F1, but you can select others using the compiler command -fxxxx Where xxxx is the font name used for printing to the graphics screen (default is Arcade_B0_F1). Look in folder Basic_Includes/GraphicsMode/Graphic_Screen_Fonts to see font names available

- SPRITE Handling commands. See here for more info

TRIM$(string)

Removes spaces from both ends of a string

LTRIM$(string)

Removes spaces from the beginning of a string

RTRIM$(string)

Removes spaces from the end of a string

Adding Assembly code to your program

You can insert your own assembly code directly in your BASIC program anywhere you like. To insert your own assembly code into the program you can do so by using the special words "**ADDASSEM:**" and "**ENDASSEM:**" after a REM (') symbol and a space as:

```
' ADDASSEM:
    LDA    #$FF
    RORB
' ENDASSEM:
```

Another example of putting your own assembly code in the BASIC program:

```
Print "HEY"
GoSub Mycode
Print "HELLO"
…


Mycode:
' ADDASSEM:
    LDA    #$FF
    STA    $400
' ENDASSEM:
Return
```

Accessing Numeric variables in your assembly code

You access numeric variables by adding the prefix "_Var_" to the name. For example if you have an integer variable in your program called Counter and you want to read the value of the variable you could use code like this:

```
DIM Counter As Integer
PRINT "COUNTER=";Counter
' ADDASSEM:
      LDD   _Var_Counter      ; Get the variable in D
      ANDB  #%11111110        ; Make the number even
      STD   _Var_Counter      ; Save the updated value
' ENDASSEM:
PRINT "COUNTER=";Counter
```

Accessing String variables in your assembly code

You access string variables by adding the prefix "_StrVar_" to the name. Strings are stored as data where the first byte is the length of the string the rest of the bytes are the actual string data.

For example if you want to interact with a string variable called A$ in assembly you could use code like this:

```
' ADDASSEM:
    LDX     #_StrVar_A  ; X points at the first byte in the A$ variable
    LDB   ,X+           ; B = value of the first byte of the string, X=X+1
                        ; First byte is the length of the string 0 to 255
    BEQ   @Finished     ; If the string length is zero then skip past
    LDU   #$400         ; Text screen start
!   LDA   ,X+           ; Get a byte from the string
    STA   ,U+           ; Show the byte on screen
    DECB                ; Decrement the counter
    BNE   <             ; Keep looping until B=0
@Finished:              ; @ in the label makes it a local label
                        ; always leave a blank line after a section of code
                        ; that uses the @ sign as a local label

' ENDASSEM:
```

Compiling from the command line

It is highly recommended that you use the SDECB IDE as it will show you syntax errors and other errors that will help to compile your program without errors. You can also compile your BASIC programs to machine language from the without the IDE from the command line using the following:

Using **MacOS** or **Linux**:

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

Using **Windows**:

```
.\BasTo6809.exe -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.

Optimizing

The 6809 CPU has the hardware MUL instruction and it is used for many of the multiplication routines even the large 16, 32, 64 bit integer and the floating point math routines use it. Division is always going to be slower as it is all software based. Therefore if you want to maximize the speed of your program, try to use multiplication instead of division if possible.

For example if you have something like this:
A = B / 2

Use:
A = B * .5

If variables are not assigned a type using the DIM command they will default to Single (Fast Floating Point). To maximize the speed of the 6809 CPU try to use 8 bit (fastest) or 16 bit types (fast). Other variable types will take more space and will run slower. See the next page for all the variable types.

Variable Types

If the compiler can't detect the numeric type it will default to being Single. It's best if you assign variables with the DIM statement or use the symbol values as a suffix after your variable name or literal number, to use the minimum size needed.

Type	Symbol		
1	`	_Bit	Min -1, Max 0
2	~`	_Unsigned _Bit	Min 0 , Max 1
3	%%	_Byte	Min -128, Max 127
4	~%%	_Unsigned _Byte	Min 0, Max 255
5	%	Integer	Min -32,768, Max 32,767
6	~%	Unsigned Integer	Min 0, Max 65,535
7	&	Long	Min -2 Gig, +2 Gig
8	~&	_Unsigned Long	Min 0, Max 4 Gig
9	&&	_Integer64	Min -(8 byte value), Max +(8 byte value)
10	~&&	_Unsigned _Integer64	Min 0, Max (8 byte value)
11	!(None)	Single (Default size)	Min E-20, Max E+19 (Fast Float)
12	#	Double	Min E-308, Max E+308

*** Types 7 to 12 will slow your program down considerably.

Examples of assigning variable types:

```
DIM MyVar1 AS _Byte
DIM MyVar2 AS _UNSIGNED LONG
DIM MyVar3 AS DOUBLE
```

Generally you don't need to force a variable to a different type then assigned, the compiler will scale the types between each other. But this is an example of forcing the type:

```
A = V~%%            ' Force V to be an Unsigned Byte
```

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 ROM addresses are.

If you use the **-dragon** option and are compiling for a Dragon computer, you don't need to worry about this step. The DRAGLOAD.BIN handles 64k files fine as they are. This is only for CoCo programs.

```
cc1sl – CoCo 1 Super Loader
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

The steps for compiling a big program are:

For **MacOS** and **Linux**:

```
./BasTo6809 -b1 HELLO.BAS
lwasm -9bl -p cd -o./HELLO.BIN HELLO.asm > ./
NEW_Assembly_Listing.txt
./cc1sl -l HELLO.BIN -oBIGFILE.BIN
```

For **Windows**:

```
.\BasTo6809 -b1 HELLO.BAS
lwasm -9bl -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.

Compiling for a Dragon computer

There are a few things you need to do in order to compile and run a program on a Dragon computer. Since the Dragon computer is very similar to a CoCo 1 code from the compiler that works on a CoCo 1 & 2 will pretty much execute as is on a Dragon computer. But the keyboard layout is different between the two computers. When compiling your program to target a Dragon computer you must use the **-dragon** option. This tells the compiler to include the proper keyboard routines that will work for the Dragon computer. Also since the Dragon uses a different file format for loading binary files you must copy the DRAGLOAD.BIN file to your floppy disk or emulator disk image. You will also need to rename your compiled program to COMPILED.BIN and copy COMPILED.BIN to the same floppy disk or emulator disk image as DRAGLOAD.BIN. You can rename DRAGLOAD.BIN

The steps you would take to compile a program called HELLO.BAS

1) Compile your program into .asm file
./BasTo6809 -ascii -dragon HELLO.BAS

2) Assemble the program into a DECB machine language binary file
lwasm -9b -p cd -o./HELLO.bin HELLO.asm

3) Copy the DRAGLOAD.BIN file to a floppy disk or disk image and rename the file as HELLO.BIN
./decb copy -2 -b -r ./DRAGLOAD.BIN DISK1.DSK, HELLO.BIN

4) Copy your actual program HELLO.bin to a floppy disk or disk image, and rename the file as COMPILED.BIN
./decb copy -2 -b -r ./HELLO.bin DISK1.DSK,COMPILED.BIN

Your disk should now have two files on it called HELLO.BIN (which is really the DRAGLOAD.BIN program) and COMPILED.BIN (which is your actual compiled program). The loader expects the file to be loaded from drive 1 and you start your program on the Dragon computer by typing: RUN”HELLO.BIN”<ENTER>

The DRAGLOAD.BIN program will handle programs that are 64k.

New commands and features added to BASIC

– **IF/THEN/ELSE/ELSEIF/ENDIF**
– **SELECT/CASE**
– **WHILE/WEND**
– **DO/WHILE/LOOP**
– **DO/LOOP/UNTIL**

- COCOHARDWARE a Command that let's you identify which CoCo your program is running on
V=COCOHARDWARE(0)

– COCOMP3 Commands See here for more info

– SDC_PLAY Command that plays an audio sample or song directly off the SD card in the SDC Controller. See here for more info

– **SDC_PLAYORCL**, **SDC_PLAYORCR**, **SDC_PLAYORCS** these commands are similar to SDCPLAY except the audio is sent to the Orchestra 90 or COCOFLASH cartridge. See here for more info

– SDC file access commands that allow you to Read & Write files directly on the SD card's own filesystem. See here for more info

– GETJOYD - Quickly get the joystick values of 0,31,63 of both joysticks both horizontally and vertically

- PRINT #-3, - Print to the graphics screen created with the GMODE command. Can also be used as ?#-3,"Hello World!". The default font is ArcadeArcade_B0_F1, but you can select others using the compiler command -fxxxx Where xxxx is the font name used for printing to the graphics screen (default is Arcade_B0_F1). Look in folder Basic_Includes/GraphicsMode/Graphic_Screen_Fonts to see font names available

- PLAYFIELD # - Used to set the Scrollable playfield mode. The # given must be an actual number and not a variable.

#	Max Resolution	Min Resolution	Size Multiple
1	256 x 7872	256 x 192	not applicable x 64
2	512 x 3840	512 x 192	not applicable x 192
3	1024 x 512	1024 x 256	not applicable x 256
4	2048 x 256	512 x 256	256 x not applicable
5	2560 x 192	512 x 192	256 x not applicable

- VIEW x,y - Command to select where in the scrollable playfield you want to see. The x & y co-ordinates are the top left corner of a viewable window of the playfield.

SPRITE_LOAD "SpriteName.asm", #[,f] — Load a Sprite

The SPRITE_LOAD command loads a sprite into your program.

- "SpriteName.asm" is the name of your sprite file (must be in assembly format). Generated by the command line tool called **PNGtoCCSprite**
- **#** is the sprite number — a numeric ID used to reference this sprite in your program when using actual SPRITE commands.
- **f** is the number of frames this sprite has can be left blank if the sprite has only one frame.

This command associates the loaded sprite with the given number, so you can easily manage multiple sprites in your game.

SPRITE Command — Control Sprites on the Screen

The SPRITE command, along with various options, allows you to control the appearance and behaviour of sprites in your program.

Command Reference:

SPRITE OFF [#]

Turns off sprite number **#**.

If no number is provided, all sprites are turned off.

SPRITE LOCATE #, x, y

Moves sprite **#** to a new position on the screen.

x and **y** are the playfield coordinates.

SPRITE SHOW #[, f]

Displays sprite # using frame f.

If the sprite is not animated (i.e. it has only one frame) use of ,f is optional.

SPRITE BACKUP #
Saves the background behind sprite #.
This allows the sprite to be cleanly erased later.

SPRITE ERASE #
Erases sprite **#** and restores the background previously saved with SPRITE BACK.

SPRITE #,x,y[,f]
Draws sprite number **#** at **x**,**y** co-ordinates using frame **f**.
f is optional, a value of 0 (single frame sprite) is used if not given.

WAIT VBL - Wait for Vertical Blank then update the sprites on screen

TRIM$(string)
Removes spaces from both ends of a string
string - String variable

LTRIM$(string)
Removes spaces from the beginning of a string
string - String variable

RTRIM$(string)
Removes spaces from the end of a string
string - String variable

BASIC commands

ATTR c1,c2,B,U

Sets display attributes of a high-resolution text screen.
c1 Foreground color
c2 Background color
B Character blink on
U Underline on
Example: ATTR 3,2,U

AUDIO switch

Connects or disconnects cassette audio output to the display speaker.
switch ON Routes sound from cassette player to the display speaker
switch OFF Disconnects cassette player sound from the display speaker
Example: AUDIO OFF

CIRCLE(x,y),Radius,Colour

Draws a circle on the screen
x,y - Origin of the circle
Radius - Size of the circle
To keep the aspect ratio close to round some of the graphics modes use scaling so the Radius isn't always a count of actual pixel values.
Colour - Colour Number of the circle to draw
Example: CIRCLE (X,Y),R,C

CLEAR

Erases all variables and arrays
Example: CLEAR

CLS [c]

Clears the text screen to a specified colour.
If c is not specified, uses the current background colour.
c Colour code
Example: CLS 2

CMP

Resets the palette registers to the standard colours for a composite monitor/TV.
Example: CMP

COLOR c1,c2

Sets the foreground and background colours of the current low-resolution graphics screen.
c1 Foreground colour code (0–8)
c2 Background colour code (0–8)
Example: COLOR 2,3

CONST constantName = value[, ...]

The CONST statement globally defines one or more named numeric or string values which will not change while the program is running.
Example: CONST PI = 3.141592654

COPYBLOCKS source,destination,#

This is a CoCo 3 specific command which copies 8k blocks to other 8k blocks very fast. Useful for double buffering CoCo 3 background screens
source - First source block to be copied from
destination - First Destination block to be copied to
- Number of blocks to copy
Example: COPYBLOCKS &H10,&H20,3

CPUSPEED #

The compiler will detect and run the CPU at the maximum speed it can on the hardware it is running. If you want to change the speed use this command to set the CPU speed to value x

= 1 Set the CPU in Emulation mode and set the speed at .895 Mhz

= 2 Set the CPU in Emulation mode and set the speed at 1.79 Mhz

= 3 Set the CPU in Emulation mode and set the speed at 2.864 MHz

If # is anything else then the CPU will be set in Native mode and run at it's max speed

Example: CPUSPEED 1

DATA constant,constant,…

Stores numeric and string constants for use with the READ statement.

constant String or numeric constant(s), such as 127, 2985, or BEAGLE

Example: DATA 45,“CAT”,98,“DOG”,24.3,1000

DIM x As _Byte

Used to declare a variable or an array as a specific data type

Examples:

To setup an array A$() as string and R4() as Single (Fast floating Point format) which is the default:

DIM A$(3,10), R4(22)

To setup an array R4() as _Byte and variable x as Long

DIM R4(10) As _Byte, x As Long

DO … LOOP

Defines a loop that repeats a block of statements. You can control it with DO WHILE/UNTIL condition and/or LOOP WHILE/UNTIL condition.

Examples:

```
Do While A < 10
        A = A + 1
Loop

Do Until A > 9
        A = A + 1
Loop

Do
        A = A + 1
Loop Until A = 10

Do
        A = A + 1
Loop While A < 10
```

DRAW string

Draws a line on the current low-resolution graphics screen as specified by string.
String commands include:
A Angle
BM Blank move
C Color
D Down
E 45-degree angle
F 135-degree angle
G 225-degree angle
H 315-degree angle
L Left
M Move draw position
N No update
R Right
S Scale
U Up
Example:
DRAW “BM128,96;U25;R25;D25;L25”

END

Marks the end of a BASIC program.
Example: END

EXEC (address)

Transfers control to a machine-language program at address.
Example:
EXEC &H7022

EXIT DO / EXIT FOR / EXIT WHILE

Exits the current DO…LOOP, FOR…NEXT, or WHILE…WEND loop immediately; execution continues with the first statement after the loop's ending keyword (LOOP, NEXT, or WEND).

Example:

```
WHILE A < 100
        IF A = 10 THEN
                EXIT WHILE
        ELSE
                A = A + 1
        END IF
WEND
```

FOR variable=n1 TO n2 STEP n3

Defines the beginning of a loop. The end is specified by NEXT.

variable Loop counter variable
n1 Starting value of counter
n2 Ending value of counter
n3 Increment or decrement value (optional; default is 1)

Example:

```
FOR Z=35 TO 125 STEP 5
```

GCLS [c]

Colour the graphics screen
c is the colour value you want the screen to be coloured
Example: GCLS 2

GCOPY SourcePage,DestinationPage

Makes a copy the Source graphics page to the Destination graphics page.
SourcePage - Source graphics page
DestinationPage - Destination graphics page
Example: GCOPY 0,3

GETJOYD

Quick Read of both joysticks (horizontal and vertical) as digital values (0, 31, or 63) and stores the results in the standard BASIC joystick locations:
Left V/H at $015A/$015B
Right V/H at $015C/$015D
Example: GETJOYD

GMODE ModeNumber,GraphicsPage

Selects the graphics screen and the graphics page.
ModeNumber is the graphics mode you want to use
GraphicsPage is the Page you want to show/use for your graphics commands

To see a list of ModeNumbers and the resolutions go here
Special note the **ModeNumber must be an actual number and cannot be a variable as the compiler needs to know exactly which graphic mode commands need to be includedat compile time.
GraphicsPage can be a variable.
If you are going to use Graphic pages, the compiler needs to know how many pages to reserve in RAM (for CoCo 1 & 2 graphics). So you must have a GMODE #,MaxPages entry at the beginning of your BASIC program. Where the value of MaxPages will be an actual number and not a variable.
Example: GMODE 16,0

** Special note if you are going to use GMODE 160 to 165 (the special NTSC composite 256 colour modes). When the compiler comes across the first GMODE command with the values of 160 to 165 it will automatically set the palette to the following values:
Palette 0,0 ' xx000000
Palette 1,16 ' xx010000
Palette 2,32 ' xx100000
Palette 3,48 ' xx110000

GOSUB line

Calls a subroutine beginning at the specified line number.
Example: GOSUB 330

GOTO line

Jumps to the specified line number.
Example: GOTO 125

IF test THEN … ELSEIF … ELSE … END IF

Multi-line decision block. BASIC runs the first true section; if none are true, it runs the ELSE section (if present).
Example:

```
IF X=0 THEN
        PRINT “ZERO”
ELSEIF X<0 THEN
        PRINT “NEG”
ELSE
        PRINT “POS”
END IF
```

INPUT var1,var2,…

Reads data from the keyboard and stores it in one or more variables.
Example: INPUT K3

LET

Assigns a value to a variable (optional keyword).
Example: LET A3=27

LINE(x0,y0)-(x1,y1),Colour[,B][F]

x0,y0 - Starting location
x1,y1 - Ending location
Colour - Colour of the Line or Box to draw
B - Draw a Box
F - Fill the Box
Example: LINE (22,33)-(27,39),1,BF

LOADM "filename", offset address

Loads the specified machine language program file from disk. You can specify an offset address to add to the program's loading address.
Example: LOADM “PROG.BIN", 3522

LOOP

LOOP is used in a DO/LOOP, see DO for more information.

LOCATE x,y

Moves the high-resolution text screen cursor to position x,y.
Example: LOCATE 20,12

LPOKE location,value

Stores a value (0–255) in a virtual memory location (0-2097151 decimal or 0-&H1FFFFF hex)
Example: LPOKE 480126,241

MOTOR

Turns the cassette motor ON or OFF.
Example: MOTOR ON

NEXT v1,v2,…

Defines the end of a FOR loop.
v1,v2,… Optional variable names for nested loops (if used, list in reverse order of the FOR variables).
If omitted, ends the most recent FOR.
Example: NEXT X,Y,Z

NTSC_FONTCOLOURS b,f

Sets the background (b) and foreground (f) colours used by the NTSC composite-output fonts in GMODE 160 to 165.
Example: NTSC_FONTCOLOURS 0,3

ON expr GOSUB line1,line2,…

Multiway call to one of several subroutines, based on expr.
Example: ON A GOSUB 100,230,500,1125

ON expr GOTO line1,line2,…

Multiway branch to one of several line numbers, based on expr.
Example: ON A GOTO 100,230,500,1125

PAINT(x,y),OldColour,FillColour

Fills the old colour value with the fill colour value which must also be the border colour of the section you are painting
x,y - Starting location
OldColour - Colour Number
FillColour - Colour Number
Example: PAINT (44,55),2,3

PALETTE pr,c

Stores colour code c (0–63) into palette register pr (0–15).
Example: PALETTE 1,13

PLAY string

Plays music as specified by string. Common commands include:

A–G Notes
L Length
O Octave
P Pause
T Tempo
or + Sharp
- Flat

Example: PLAY “L1;A;A#;A-”

PLAYFIELD #

Sets the scrollable playfield mode in your BASIC program. The # must be a literal number, not a variable.

To create playfield graphics, use the command-line tool PNGtoCC3Playfield to convert a PNG into a file (or files). These can
then be loaded into memory using one of the following commands in BASIC:

- LOADM
- SDC_LOADM
- SDC_BIGLOADM

Once loaded, the background playfield becomes scrollable according to the selected mode.

# Max Resolution	Min Resolution	Size Multiple
1 - 256 x 7872	256 x 192	not applicable x 64
2 - 512 x 3840	512 x 192	not applicable x 192
3 - 1024 x 512	1024 x 256	not applicable x 256
4 - 2048 x 256	512 x 256	256 x not applicable
5 - 2560 x 192	512 x 192	256 x not applicable

POKE location,value

Stores a value (0–255) in a memory location (0–65535 decimal or 0–$FFFF hexadecimal).
Example: POKE 28000,241

PRINT message

Prints on the text screen.
Example: PRINT “HELLO!”

PRINT TAB(n)

Moves the cursor to column n on the low- and high-resolution text screens.
Example: PRINT TAB(22);“HELLO!”

PRINT@ n,message

Prints message on the low-resolution text screen at position n.
Example: PRINT@11,“HELLO!”

READ var1,var2,…

Reads the next item(s) from DATA statements and stores them into variables.
Example: READ A1,B,C7

REM comment

Inserts a remark (comment). BASIC ignores everything after REM on that line.
Example: REM THIS IS A COMMENT LINE

RESTORE

Resets the READ pointer back to the first item on the first DATA line.
Example: RESTORE

RETURN

Returns from a subroutine to the statement following the last GOSUB.
Example: RETURN

RGB

Resets the palette registers to the standard colours for an RGB monitor.
Example: RGB

RUN

Executes a program.
Example: RUN

SAMPLE option

Control audio sample playback. The following options are available:
SINGLE #- Plays sample # once
LOOP # - Plays sample # continuously
OFF - Turns off sample playback
Example: SAMPLE SINGLE 2

SAMPLE_LOAD “filename", sample number

Loads a raw audio sample (samples must be raw 8 bit unsigned at 6003 hz) into the CoCo memory and assigns a number to the sample to be referenced for play back with the SAMPLE command.
Example: SAMPLE_LOAD “JUMP.raw”, 0

SCREEN type,colors

Selects low-resolution screen modes and color sets.
type
0 Text
1 Graphics
colors
0 Color set 0
1 Color set 1
Example: SCREEN 0,1

SELECT CASE expr … CASE … CASE ELSE … END SELECT

Multi-way branch based on expr. BASIC executes the first matching CASE block; if none match, it executes CASE ELSE (if present).
Example:

```
Select Case A
        Case 1
                Print "ONE"
        Case 2
                Print "TWO"
        Case Else
                Print "OTHER"
End Select
```

SET(x,y,Colour)

Sets a pixel on the screen
x,y - Screen location of the pixel to be drawn
Colour - Colour Number of the pixel to be drawn
Example: SET (11,11,3)

SOUND tone,duration

Sounds a tone for a specified duration.
tone 1–255 sets pitch
duration 1–255 sets duration
Example: SOUND 33,22

SPRITE (options)

Controls sprites on the screen (turn on/off, position, draw/ erase, and background save/restore). Sprites are referenced by sprite number #, with optional frame f for animated sprites.
Example: SPRITE LOCATE 1, 80, 60 : SPRITE SHOW 1, 2

Sprite Option details:
SPRITE OFF [#]
Turns off sprite number #.
If no number is provided, all sprites are turned off.

SPRITE LOCATE #, x, y
Moves sprite # to a new position on the screen.
x and y are the playfield coordinates.

SPRITE SHOW #[, f]
Displays sprite # using frame f.
If the sprite is not animated (i.e. it has only one frame) use of ,f is optional.
SPRITE BACKUP #
Saves the background behind sprite #.
This allows the sprite to be cleanly erased later.
SPRITE ERASE #
Erases sprite # and restores the background previously saved with SPRITE BACK.

SPRITE #,x,y[,f]
Draws sprite number # at x,y co-ordinates using frame f.
f is optional, a value of 0 (single frame sprite) is used if not given.

SPRITE_LOAD “SpriteName.asm”, # [,f]

Loads a sprite definition from an assembly-format sprite file (Generated by the command line tool called PNGtoCCSprite) and assigns it to sprite number # for later use by SPRITE commands. Optional f specifies the number of frames (omit if the sprite has only one frame).
Example: SPRITE_LOAD “HERO.asm”, 1, 4

STOP

Stops execution of a program.
Example: STOP

TIMER=n

Sets TIMER to n.
Example: TIMER=120

UNTIL

UNTIL is also used in a DO/LOOP, see DO for more information.

VIEW x,y

Selects which part of the scrollable playfield is shown on-screen. The x and y values specify the top-left corner of the visible window into the playfield.
Example: VIEW 0,0

WAIT VBL

Wait for Vertical Blank then update the sprites on screen
Example: Wait VBL

WHILE test … WEND

Repeats a block of statements while the test is true. When the test becomes false, execution continues with the line after WEND. WHILE is also used in a DO/LOOP, see DO for more information.

Example:

```
While A < 10
        A = A + 1
Wend
```

WIDTH n

Sets the text screen resolution:

32 32x16 (low-resolution text)
40 40x28 (high-resolution text)
64 64x28 (high-resolution text)
80 80x28 (high-resolution text)

Example: WIDTH 80

Functions Commands

ABS (n)

Returns the absolute value of n.
Example: A = ABS(B)

ASC (string)

Returns the code (ASCII value) of the first character in string.
Example: A = ASC(B$)

ATN (n)

Returns the arctangent of n, in radians.
Example: A = ATN(B/3)

BUTTON (n)

Returns 1 if joystick button n is being pressed; returns 0 if it is not.
n can be:
0 - Right joystick, Button 1 (old joystick)
1 - Right joystick, Button 2
2 - Left joystick, Button 1 (old joystick)
3 - Left joystick, Button 2
Example: A = BUTTON(D)

CHR$ (n)

Returns the character corresponding to character code n.
Example: A$ = CHR$(65)

COCOHARDWARE(0)

Returns with a numeric value representing which CoCo you are using.
Example: V = COCOHARDWARE(0)
Where the bits of variable V will signify the CoCo Hardware as:
Bit 0 is the Computer Type,0 = CoCo 1/2, 1 = CoCo 3
Bit 7 is the CPU type, 0 = 6809, 1 = 6309

COS (angle)

Returns the cosine of angle (angle is in radians).
Example: A = COS(B)

EXP (n)

Returns the natural exponential of n (e^n).
Example: A = EXP(B*1.15)

FIX (n)

Returns the truncated integer of n. Unlike INT, FIX does not return the next lower number for negative values; it truncates toward zero.
Example: A = FIX(B - .2)

HEX$ (n)

Returns a string containing the hexadecimal value of n.
Example: PRINT HEX$(A); “=”; A

INKEY$

Checks the keyboard and returns the key being pressed or, if no key is being pressed, returns a null string (””).
Example: A$ = INKEY$

INSTR (p, s, t)

Searches a string. Returns the position of target string t, within search string s, starting at position p.
p - Start position of search
s - String being searched
t - Target string
Example: A = INSTR(1, M5$, “BEETS”)

INT (n)

Converts n to the largest integer that is less than or equal to n.
Example: A = INT(B + .5)

JOYSTK (i)

Returns the horizontal or vertical coordinate (i) of the left or right joystick.
0 - Horizontal, right joystick
1 - Vertical, right joystick
2 - Horizontal, left joystick
3 - Vertical, left joystick
Example: A = JOYSTK(B)

LEFT$ (string, length)

Returns the left portion of a string.
length specifies the number of characters returned.
Example: A$ = LEFT$(B$, 3)

LEN (string)

Returns the length of string.
Example: A = LEN(B$)

LOG (n)

Returns the natural logarithm of n.
Example: A = LOG(B / 2)

LPEEK (memory location)

Returns the contents of a virtual memory location (0-2097151 decimal or 0-&H1FFFFF hex)
Example: A = LPEEK(&H60000)

LTRIM$ (string)

Removes both leading spaces from the string.
Example: A$ = LTRIM$(B$)

MID$ (s, p, l)

Returns a substring of string s.
s - Source string
p - Starting position of substring
l - Length of substring
Example: A$ = MID$(B$, 2, 2)

PEEK (memory location)

Returns the contents of a memory location (0–65535 decimal or 0–&HFFFF hexadecimal).
Example: A = PEEK(3020)

POINT (x,y)

Returns the colour value of the pixel selected from the current GMODE screen.
x,y - Screen location of the pixel value requested
Example: A = POINT(10,12)

POS (dev)

Returns the current print position.
dev (print device number):
0 - Screen
-2 - Printer
Example: A = POS(0)

RIGHT$ (string, length)

Returns the right portion of a string.
length specifies the number of characters returned.
Example: A$ = RIGHT$(B$, 4)

RND (n)

Generates a “random” number between 1 and n. If n = 0 then it generates a random number from 0 to <1
Example: A = RND(0)

RTRIM$ (string)

Removes trailing spaces from the string.
Example: A$ = RRIM$(B$)

SGN (n)

Returns the sign of n:
-1 - Negative
0 - Zero
1 - Positive
Example: A = SGN(A + .1)

SIN (angle)

Returns the sine of angle (angle is in radians).
Example: A = SIN(B/3.14159)

STRING$ (l, c)

Returns a string of a repeated character.
l - Length of string
c - Character used (can be a code or a string)
Example: A$ = STRING$(22, “A”)

STR$ (n)

Converts n to a string.
Example: A$ = STR$(1.234)

SQR (n)

Returns the square root of n.
Example: A = SQR(B/2)

TAN (angle)

Returns the tangent of angle (angle is in radians).
Example: A = TAN(B)

TIMER

Returns the contents of the timer (0–65535).
Example: A = TIMER / 18

TRIM$ (string)

Removes both leading and trailing spaces from the string.
Example: A$ = TRIM$(B$)

VAL (string)

Converts a string to a number.
Example: A = VAL(“1.23”)

VARPTR (variable)

Returns a pointer to where a variable is located in memory.
Example: A = VARPTR(B)

Constants

You can use Constants in your program that will act like place holders for actual values that will be substituted at compile time. They are different than variables as they don’t actually take any RAM in your program.

Examples:

```
Const PI = 3.141593
Const PI2 = PI * 2
```

Constants can also be quoted strings:

```
Const circumferenceText = "THE CIRCUMFERENCE OF THE CIRCLE IS"
Const areaText = "THE AREA OF THE CIRCLE IS"
```

Math / Logical Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division (Integers will be rounded)
^	Exponent
MOD	Remainder of division
\	Integer division (no rounding)

* Doing division with numbers that have different sign/unsigned type might give incorrect results. There could be checks added in the code, but this would slow down doing division, which is already a slow process.

AND	Bitwise AND
OR	Bitwise OR
XOR	Bitwise Exclusive OR
NOT	Bitwise Compliment (invert each bit)

CoCoMP3 Commands

You can easily and directly control the CoCoMP3 hardware from your BASIC program using the following commands:

COCOMP3_AUDIO_MODE(x)

Sets where CoCoMP3 audio is routed.
x=0 leave unchanged
x=1 (default) route audio to the CoCo cassette interface so it plays through the TV speaker.
Example: z=COCOMP3_AUDIO_MODE(1)

COCOMP3_COMBINATION_PLAY_SETTING(x,y)

Sets a “combination play” range of tracks from x to y.
Example: z=COCOMP3_COMBINATION_PLAY_SETTING(10,25)

COCOMP3_CYCLE_MODE_SETTING(x)

Sets playback mode.
x=0 full cycle
1 repeat track
2 stop after track
3 random all
4 repeat folder
5 random in folder
6 order in folder then stop
7 order all then stop.
Example: z=COCOMP3_CYCLE_MODE_SETTING(4)

COCOMP3_END_COMBINATION_PLAY(0)

When a combination play range finishes, stop playback.
Example: z=COCOMP3_END_COMBINATION_PLAY(0)

COCOMP3_END_PLAYING(0)

Ends the current track and skips to the next track (like Next).
Example: z=COCOMP3_END_PLAYING(0)

COCOMP3_GET_CURRENT_TRACK(0)

Returns the current track number assigned by the CoCoMP3 (usable with PLAY_TRACK_NUMBER or SELECT_BUT_NO_PLAY).
Example: z=COCOMP3_GET_CURRENT_TRACK(0)

COCOMP3_GET_DRIVE_STATUS(0)

Returns microSD status:
0 = not installed
2 = installed
Example: z=COCOMP3_GET_DRIVE_STATUS(0)

COCOMP3_GET_FOLDER_DIR_TRACK(0)

Returns the first track number in the current folder.
Example: z=COCOMP3_GET_FOLDER_DIR_TRACK(0)

COCOMP3_GET_NUMBER_OF_TRACKS(0)

Returns the total number of tracks on the microSD.
Example: z=COCOMP3_GET_NUMBER_OF_TRACKS(0)

COCOMP3_GET_PLAY_STATUS(0)

Returns play status:
0 = stopped
1 = playing
Example: z=COCOMP3_GET_PLAY_STATUS(0)

COCOMP3_GET_TRACKS_IN_FOLDER(0)

Returns how many tracks are in the current folder.
Example: z=COCOMP3_GET_TRACKS_IN_FOLDER(0)

COCOMP3_NEXT(0)

Skips to (and plays) the next track.
Example: z=COCOMP3_NEXT(0)

COCOMP3_PAUSE(0)

Pauses playback of the current track.
Example: z=COCOMP3_PAUSE(0)

COCOMP3_PLAY(0)

Plays the current track (or the first track after power-on).
Example: z=COCOMP3_PLAY(0)

COCOMP3_PLAY_NEXT_FOLDER(0)

Advances to the next folder and starts playing there (behavior depends on Cycle Mode).
Example: z=COCOMP3_PLAY_NEXT_FOLDER(0)

COCOMP3_PLAY_PREVIOUS_FOLDER(0)

Moves to the previous folder and starts playing there (behavior depends on Cycle Mode).
Example: z=COCOMP3_PLAY_PREVIOUS_FOLDER(0)

COCOMP3_PLAY_TRACK(T$)

Selects and plays a specific file path. The command converts ".MP3/.WAV" to "*MP3/*WAV" as required by the hardware.
Example: z=COCOMP3_PLAY_TRACK("/FOLDER02/00001.MP3")

COCOMP3_PLAY_TRACK_NUMBER(x)

Plays track number x (track numbers follow copy order to the microSD; use GET_CURRENT_TRACK to discover a track's number).
Example: z=COCOMP3_PLAY_TRACK_NUMBER(12)

COCOMP3_PREVIOUS(0)

Skips to (and plays) the previous track.
Example: z=COCOMP3_PREVIOUS(0)

COCOMP3_RAW(R$)

Sends a raw command packet to the CoCoMP3 (for advanced control; see DY-SV5W_Datasheet.pdf for codes).
Example:z=COCOMP3_RAW(CHR$(&HAA)+CHR$(1)+CHR$(0)+CHR$(&HAB))

COCOMP3_SELECT_BUT_NO_PLAY(x)

Stops the current track, queues track x, but does not start playback until you call PLAY.
Example: z=COCOMP3_SELECT_BUT_NO_PLAY(12)

COCOMP3_SET_CYCLE_TIMES(x)

Sets how many times to repeat the current track/folder before stopping; manually selecting another track overrides this.
Example: z=COCOMP3_SET_CYCLE_TIMES(3)

COCOMP3_SET_EQ(x)

Sets the internal EQ preset:
0 Normal
1 Pop
2 Rock
3 Jazz
4 Classical
Example: z=COCOMP3_SET_EQ(2)

COCOMP3_SET_PATH_INTERLUDE(P$)

Interrupts the current track, plays tracks from folder path P$, then resumes the interrupted track afterward (behaviour may depend on Cycle Mode).
Example: z=COCOMP3_SET_PATH_INTERLUDE(”/FOLDER02/*MP3”)

COCOMP3_SET_TRACK_INTERLUDE(x)

Interrupts the current track, plays track x, then resumes the interrupted track afterward.
Example: z=COCOMP3_SET_TRACK_INTERLUDE(20)

COCOMP3_SET_VOL(x)

Sets volume level 0–30 (default on power-up is 30).
Example: z=COCOMP3_SET_VOL(18)

COCOMP3_STOP(0)

Stops playback of the current track.
Example: z=COCOMP3_STOP(0)

COCOMP3_TEST(0)

Checks if CoCoMP3 is ready. Returns:
0 OK
-1 not powered/connected
-2 no microSD
-3 no tracks on microSD.
Example: z=COCOMP3_TEST(0)

COCOMP3_VOL_DOWN(0)

Decreases volume by 1 step.
Example: z=COCOMP3_VOL_DOWN(0)

COCOMP3_VOL_FADE(x)

Fades volume to 0 over x milliseconds, then stops playback (software-driven; the CoCo is halted during the fade).
Example: z=COCOMP3_VOL_FADE(1500)

COCOMP3_VOL_MAX(0)

Sets volume to maximum (30).
Example: z=COCOMP3_VOL_MAX(0)

COCOMP3_VOL_UP(0)

Increases volume by 1 step.
Example: z=COCOMP3_VOL_UP(0)

CoCo SDC Specific Commands

SDC_LOADM “FILENAME.BIN”,#[,Offset]

Loads a machine-language binary file from the SDC directly into memory.# is the file number (0 or 1). Offset is optional and is added to the file’s original LOADM address.
Example: SDC_LOADM “GAME.BIN”,0,256

SDC_SAVEM “FILENAME.BIN”,#,Start,End,Exec

Saves a section of memory directly to the SDC as a machine-language file.
is the file number (0 or 1).
Start/End define the memory range to save; Exec is the run address.
Example: SDC_SAVEM “LEVEL1.BIN”,1,28000,29000,28000

SDC_BIGLOADM “FILENAME.BIN”,#

Loads CoCo 3 memory blocks very quickly from the SDC (useful for large assets like screens).
is the file number (0 or 1). File must match the BIGLOADM format spec.
Example: SDC_BIGLOADM “TITLESCR.BIN”,0

SDC_OPEN “FILENAME.EXT”,“X”,#

Opens a file on the SD card for direct read/write access and forces CPU to slow speed while open.
“X” is:
“R” (read)
“W” (write)
is the file number (0 or 1).
Example: SDC_OPEN “DATA.DAT”,“R”,0

SDC_CLOSE(#)

Closes the open file (0 or 1) and restores CPU speed to the previous setting (default Max).
Example: SDC_CLOSE(0)

SDC_PUTBYTE0 x

Writes one byte (value/variable x) to open file #0.
Example: SDC_PUTBYTE0 A

SDC_PUTBYTE1 x

Writes one byte (value/variable x) to open file #1.
Example: SDC_PUTBYTE1 255

x = SDC_GETBYTE(#)

Reads one byte from open file # (0 or 1) into x; file position auto-increments after each read.
Example: B = SDC_GETBYTE(0)

SDC_SETPOS0(x)

Sets the read/write position for open file #0. x is a 32-bit (unsigned) byte offset, zero-based.
Example: SDC_SETPOS0(299)

SDC_SETPOS1(x)

Sets the read/write position for open file #1. x is a 32-bit (unsigned) byte offset, zero-based.
Example: SDC_SETPOS1(0)

A$ = SDC_FILEINFO$(#)

Returns a 32-byte info record for the file associated with file # (0 or 1), useful for file size.
Bytes:
1-8 name
9-11 ext
12 attributes ($10 dir, $04 SDF, $02 hidden, $01 locked)
29-32 size (LSB first).
Example: A$ = SDC_FILEINFO$(0)

x = SDC_DELETE(A$)

Deletes a file or an empty directory on the SDC. A$ is the full path.
Returns:
0 OK
1 busy too long
3 invalid path
4 HW error
5 not found
6 dir not empty.
Example: R = SDC_DELETE(”/GAMES/OLD.BIN”)

x = SDC_MKDIR(A$)

Creates a directory on the SDC. A$ is the full path of the directory to create.
Returns:
0 OK
1 busy too long
3 invalid path
4 HW error
5 parent not found
6 name in use.
Example: R = SDC_MKDIR(”/GAMES/NEW”)

x = SDC_SETDIR(A$)

Changes the current directory on the SDC. A$ is the full path to change to.
Returns:
0 OK
1 busy too long
3 invalid path
4 HW error
5 target dir not found.
Example: R = SDC_SETDIR(”/GAMES”)

A$ = SDC_GETCURDIR$(#)

Gets current-directory info for the SD card (per file # 0 or 1) into A$.
Bytes:
1-8 filename
9-11 extension
12-31 private/reserved.
Example: A$ = SDC_GETCURDIR$(0)

x = SDC_INITDIR(A$)

Initializes a directory listing with a path + wildcard filter (must be called before DIRPAGE/DIRLIST$).
Example patterns: “MYDIR/.” or “MYDIR/.TXT”
Returns:
0 OK
1 busy too long
3 invalid path
4 HW error
5 target dir not found.
Example: R = SDC_INITDIR(“MYDIR/.TXT”)

x = SDC_DIRPAGE(0)

Retrieves the next 256-byte directory page (16 records × 16 bytes). Keep calling until a page contains unused (zero) records.
Returns:
0 OK
1 busy too long
4 not initiated or end of listing.
Example: R = SDC_DIRPAGE(0)

A$ = SDC_DIRLIST$(y)

Returns one 16-byte directory entry from the most recent DIRPAGE.
y selects entry 0–15.
Entry bytes:
1-8 name
9-11 ext
12 attributes ($10 dir, $02 hidden, $01 locked)
13-16 size (MSB first).
Example: A$ = SDC_DIRLIST$(3)

CoCo SDC Audio Playback Commands

These commands allow you to playback audio samples/music files directly off the CoCo SDC at a very high bitrate. For mono at 44.75 kHz and stereo at 22.375 kHz

SDC_PLAY “filename”,#

Plays an audio file stored on the SDC, outputting sound directly through the CoCo.
is the file number (0 or 1).
Example: SDC_PLAY “MYAUDIO.RAW”,0

SDC_PLAYORCL “filename”,#

Plays an audio file stored on the SDC, outputting sound through the left channel of the Orchestra 90/CoCo Flash
is the file number (0 or 1).
Example: SDC_PLAYORCL “MYAUDIO.RAW”,0

SDC_PLAYORCR “filename”,#

Plays an audio file stored on the SDC, outputting sound directly through the CoCo.
is the file number (0 or 1).
Example: SDC_PLAYORCR “MYAUDIO.RAW”,0

SDC_PLAYORCS “filename”,#

Plays an audio file stored on the SDC, outputting sound directly through the CoCo.
is the file number (0 or 1).
Example: SDC_PLAYORCS “MYAUDIO.RAW”,0

To stop a playing music file or sample you can press the BREAK key.

In order for you to get your audio sample in the correct format to be played back you’ll need to prepare your audio samples and

put them on the SD card. The format for the raw audio file that will be played is mono 8 bits unsigned. To convert any sound file or even the audio from a video file to the correct format used with the SDCPLAY command use FFMPEG and the following command:

ffmpeg -i source_audio.mp3 -acodec pcm_u8 -f u8 -ac 1 -ar 44750 -af aresample=44750:filter_size=256:cutoff=1.0 MYAUDIO.RAW

You can also use an audio tool like Audacity to Export an audio sample to an uncompressed Mono 44750 Hz, RAW (header-less) unsigned 8-bit PCM sample.
If you want to stream 8 bit stereo sound from your CoCo to the COCOFLASH/Orchestra90 use the command:
SDC_PLAYORCS where the sample MYSAMPLE.RAW is stored on the SD card in your SDC Controller. It can be created with the FFMPEG command below:

ffmpeg -i source_audio.mp3 -acodec pcm_u8 -f u8 -ac 2 -ar 22375 -af aresample=22375:filter_size=256:cutoff=1.0 MYSAMPLE.RAW

You can also use an audio tool like Audacity to Export an audio sample to an uncompressed Stereo 22375 Hz, RAW (header-less) unsigned 8-bit PCM sample.

Audio Sample Handling - Overview

```
Use the SAMPLE_LOAD
SAMPLE_LOAD fileName$,Sample #
and SAMPLE PLAY #, SAMPLE LOOP # OR SAMPLE OFF command

' Handle SAMPLE command
' SAMPLE SINGLE 2 ' Play audio sample #2 one single time
' SAMPLE LOOP 4 ' Play audio sample #4 continously looping
' SAMPLE OFF ' turns off any sample playing


ffmpeg -i source_audio.mp3 -acodec pcm_u8 -f u8 -ac 1 -ar 6003
-af aresample=6003:filter_size=256:cutoff=1.0 MYSAMPLE.RAW
```

Sprite Handling - Overview

To use sprites in your BASIC programs you need to first prepare them with some external tools.
- A graphic editor like GIMP to make or convert/export your sprite image into a 32 bit RGBA (includes transparency) PNG file.
- The command line tool (included with this compiler) PNGtoCCSB to convert the PNG file into a Sprite to be used in your BASIC program.

Typically you'll use PNGtoCCSB to convert your PNG file into a sprite with a command similar to:

```
PNGtoCCSB -g15 -s0 MySprite.PNG
PNGtoCCSB -g15 -s0 MyOtherSprite.PNG
```

This command will convert the PNG files MySprite.PNG & MyOtherSprite.PNG into compiled sprites ready to be used -g15 tells the tool we are going to use this sprite with a GMODE 15 graphics screen in your BASIC program. When converting the PNG to a compiled sprite it will match the colours of the PNG to the available colours in the GMODE & Screen mode (-sx option) on the CoCo.

This will save the compiled sprites as MySprite.asm & MyOtherSprite.asm

In your BASIC program you will need to start your program with the following commands:

```
Sprite_Load "MySprite.asm", 0 ' Load sprite as #0
Sprite_Load "MyOtherSprite.asm", 1 ' Load sprite as #1

GMODE 15,1 'This will reserve graphics page 0 and graphics page 1
GMODE 15,0 'Set the current graphics page to use as 0
GCLS 0 'Colour the graphics screen colour 0
SCREEN 1,0 'Show the graphics screen (you could do this later)
Draw any background graphics for your program here…
```

Once your background graphics are drawn you need to copy screen 0 to screen 1 this is done with the GCOPY command. This is necessary as the sprite routines use double buffering where the screen is switched to show screen 1 the sprite updates are drawn on screen 0, once the sprites are updated, screen 0 is shown and screen 1 sprite updates are drawn.

```
GCOPY 0,1 ‘Copy graphics screen 0 to graphics screen 1
```

Now that your background is setup you are ready to draw the sprites on the screen using the following commands:

A normal Sprite usage routine would look like this:

```
SPRITE LOCATE 0, X1, Y1  ‘ Set the location of sprite 0
SPRITE BACKUP 0          ‘ Copy what’s behind sprite 0
SPRITE SHOW 0, 0         ‘ Draw sprite 0, frame 0
SPRITE LOCATE 1, X2, Y2  ‘ Set the location of sprite 1
SPRITE BACKUP 1          ‘ Copy what’s behind sprite 1
SPRITE SHOW 1, 0         ‘ Draw sprite 1, frame 0
```

Except for the SPRITE LOCATE command the above commands aren’t executed at this point, they are added to a sprite command queue that is executed when the WAIT VBL command is used as:

```
WAIT VBL                 ' This is when the actual sprite
updates occur
```

Typically you will next want the sprite erase commands after the Wait VBL command, which again are only added to the sprite command queue at this point. It is usually best to reverse the order of the sprites for erasing.

```
SPRITE ERASE 1      ‘ Erase Sprite 1 (reverse order is good)
SPRITE ERASE 0      ‘ Erase Sprite 0
```

Typically you would have a routine to update your sprites like this:

```
SpriteUpdate:
SPRITE LOCATE 0, X1, Y1  ‘ Set the location of sprite 0
SPRITE BACKUP 0          ‘ Copy what’s behind sprite 0
SPRITE SHOW 0, 0         ‘ Draw sprite 0, frame 0
SPRITE LOCATE 1, X2, Y2  ‘ Set the location of sprite 1
SPRITE BACKUP 1          ‘ Copy what’s behind sprite 1
SPRITE SHOW 1, 0         ‘ Draw sprite 1, frame 0
WAIT VBL                 ' This is when the actual sprite
updates occur
SPRITE ERASE 1      ‘ Erase Sprite 1 (reverse order is good)
SPRITE ERASE 0      ‘ Erase Sprite 0
Return                   ‘ All done updating sprites
```

In your program you would change X1,Y1 & X2,Y2 to the locations you want and then update the sprites on the screen with the command:

GOSUB SpriteUpdate

The last thing to know about the sprites is they can also have a frame option. If you create or download a sprite sheet of an animated sprite, they are usually created as many images in a row. For example a sprite sheet might look something like this:

If you wanted to use this sprite in your program you would use the -axx option with the PNGtoCCSB tool in this case we have 8 frames of animation so we would use the following command:
PNGtoCCSB -g15 -s0 -a8 WolfCub.PNG

The -a8 tells the tool to create a compiled sprite with 8 frames from the one PNG file. The PNG must only have the images in one horizontal row and be evenly spaced.

In your program you use the same SPRITE commands as above except you also now want to use the frame option of the SPRITE SHOW command where the 8 frames to show are from frame 0 to frame 7
Sprite_Load "WolfCub.asm", 2 ‘ Load sprite as #2

The sprite update code would now look like this:

```
SpriteUpdate:
SPRITE LOCATE 0, X1, Y1  ‘ Set the location of sprite 0
SPRITE BACKUP 0          ‘ Copy what’s behind sprite 0
SPRITE SHOW 0, 0         ‘ Draw sprite 0, frame 0
SPRITE LOCATE 1, X2, Y2  ‘ Set the location of sprite 1
SPRITE BACKUP 1          ‘ Copy what’s behind sprite 1
SPRITE SHOW 1, 0         ‘ Draw sprite 1, frame 0
SPRITE LOCATE 2, WolfCubX, WolfCubY     ‘ Set the location of
sprite 2
SPRITE BACKUP 2          ‘ Copy what’s behind sprite 2
SPRITE SHOW 2, F         ‘ Draw sprite 2, frame F
WAIT VBL                 ' This is when the actual sprite
updates occur
SPRITE ERASE 2      ‘ Erase Sprite 2 (reverse order is good)
SPRITE ERASE 1      ‘ Erase Sprite 1
SPRITE ERASE 0      ‘ Erase Sprite 0
Return                   ‘ All done updating sprites
```

* At this point PNGtoCCSB doesn’t support semigraphics screen modes for sprites, but all other screen modes are supported.

CoCo 1 & 2 Graphic Modes - GMODE

GMODE #	Resolution	Colours	Bytes Per Screen	Mode Name
0	32 x 16	9	512	Internal alphanumeric
1	32 x 16	2	512	External alphanumeric
2	64 x 32	9	512	Semi graphic-4
3	64 x 32	9	2048	Semi graphic-8*
4	64 x 48	9**	512	Semi graphic-6
5	64 x 48	9	3072	Semi graphic-12***
6	64 x 64	9	2048	Semi graphic-8
7	64 x 96	9	3072	Semi graphic-12
8	64 x 192	9	6144	Semi graphic-24
9	64 x 64	4	1024	Full graphic 1-C
10	128 x 64	2	1024	Full graphic 1-R
11	128 x 64	4	2048	Full graphic 2-C
12	128 x 96	2	1536	Full graphic 2-R
13	128 x 96	4	3072	Full graphic 3-C
14	128 x 192	2	3072	Full graphic 3-R
15	128 x 192	4	6144	Full graphic 6-C
16	256 x 192	2	6144	Full graphic 6-R
17			6144	DMA Mode (unusable)
18	256 x 192	2	6144	Compile for Artifacts

* Semi graphics 4 Hybrid using Semi Graphics 8 Mode
** 3 Colours for each colour set
*** Semi graphics-6 Hybrid using Semi Graphics 12 mode

CoCo 3 Graphic Modes - GMODE

GMODE #	Resolution	Colours	Bytes Per Screen
100	64 x 192	4	3200
101	64 x 200	4	3200
102	64 x 225	4	3600
103	64 x 192	16	6144
104	64 x 200	16	6400
105	64 x 225	16	7200
106	80 x 192	4	3840
107	80 x 200	4	4000
108	80 x 225	4	4500
109	80 x 192	16	7680
110	80 x 200	16	8000
111	80 x 225	16	9000
112	128 x 192	2	3072
113	128 x 200	2	3200
114	128 x 225	2	3600
115	128 x 192	4	6144
116	128 x 200	4	6400
117	128 x 225	4	7200
118	128 x 192	16	12288
119	128 x 200	16	12800
120	128 x 225	16	14400
121	160 x 192*	2	3840
*(viewable) really 128x192, * Special mode that repeats the left 4 bytes on the right side of the screen			
122	160 x 200*	2	4000
*(viewable) really 128x192, * Special mode that repeats the left 4 bytes on the right side of the screen			

123	160 x 225*	2	4500
*(viewable) really 128x192, * Special mode that repeats the left 4 bytes on the right side of the screen			
124	160 x 192	4	7680
125	160 x 200	4	8000
126	160 x 225	4	9000
127	160 x 192	16	15360
128	160 x 200	16	16000
129	160 x 225	16	18000
130	256 x 192	2	6144
131	256 x 200	2	6400
132	256 x 225	2	7200
133	256 x 192	4	12288
134	256 x 200	4	12800
135	256 x 225	4	14400
136	256 x 192	16	24576
137	256 x 200	16	25600
138	256 x 225	16	28800
139	320 x 192	2	7680
140	320 x 200	2	8000
141	320 x 225	2	9000
142	320 x 192	4	15360
143	320 x 200	4	16000
144	320 x 225	4	18000
145	320 x 192	16	30720
146	320 x 200	16	32000
147	320 x 225	16	36000
148	512 x 192	2	12288
149	512 x 200	2	12800
150	512 x 225	2	14400
151	512 x 192	4	24576

152	512 x 200	4	25600
153	512 x 225	4	28800
154	640 x 192	2	15360
155	640 x 200	2	16000
156	640 x 225	2	18000
157	640 x 192	4	30720
158	640 x 200	4	32000
159	640 x 225	4	36000
160*	128 x 192	256	24576
161*	128 x 200	256	25600
162*	128 x 225	256	28800
163*	160 x 192	256	30720
164*	160 x 200	256	32000
165*	160 x 225	256	36000

* These modes are NTSC composite CoCo 3 output only

Specifications

SDC_BIGLOADM"FILENAME.BIN",#

Loads CoCo 3 Memory Blocks very fast. Useful for loading in background screens for games or other large amounts of data.
is the file number 0 or 1

This command loads complete $2000 byte blocks into the CoCo 3's memory. The format of this file is written in 512 byte chunks, as the SDC streaming mode reads file data 512 bytes at a time.
Chunk 0 is the header/Memory Mapping Chunk:
Bytes

0 & 1	File version # Version 1 is the only supported version to date
2 & 3	Is the 16 bit number of the first bank to load
x & x+1	Is the 16 bit number of the next bank to load and so on until we get a $FFFF If we reach a $FFFF this signifies we've copied the last block and are done.
512 & 513	Chunk 1 - This is the actual first word of data for the first block This contains $2000 bytes of data, the next $2000 bytes will be the data for the 2nd bank number given in the memory mapping Chunk and will continue with $2000 bytes of data for all the rest of the banks given in the memory map Chunk.

Thanks

I'd like to thank Scott Cooper (Tazman) for initial testing of the compiler. Scott also wrote the assembly code for the SET and POINT routines used in the semi-graphics modes. Thanks to Curtis Boyle for teaching me how to do horizontal scrolling on the CoCo 3.

I'd also like to thank others on The CoCo Nation Discord who inspired me to keep adding new features, including Bruce D. Moore, Erico Monteiro & Pete Willard.

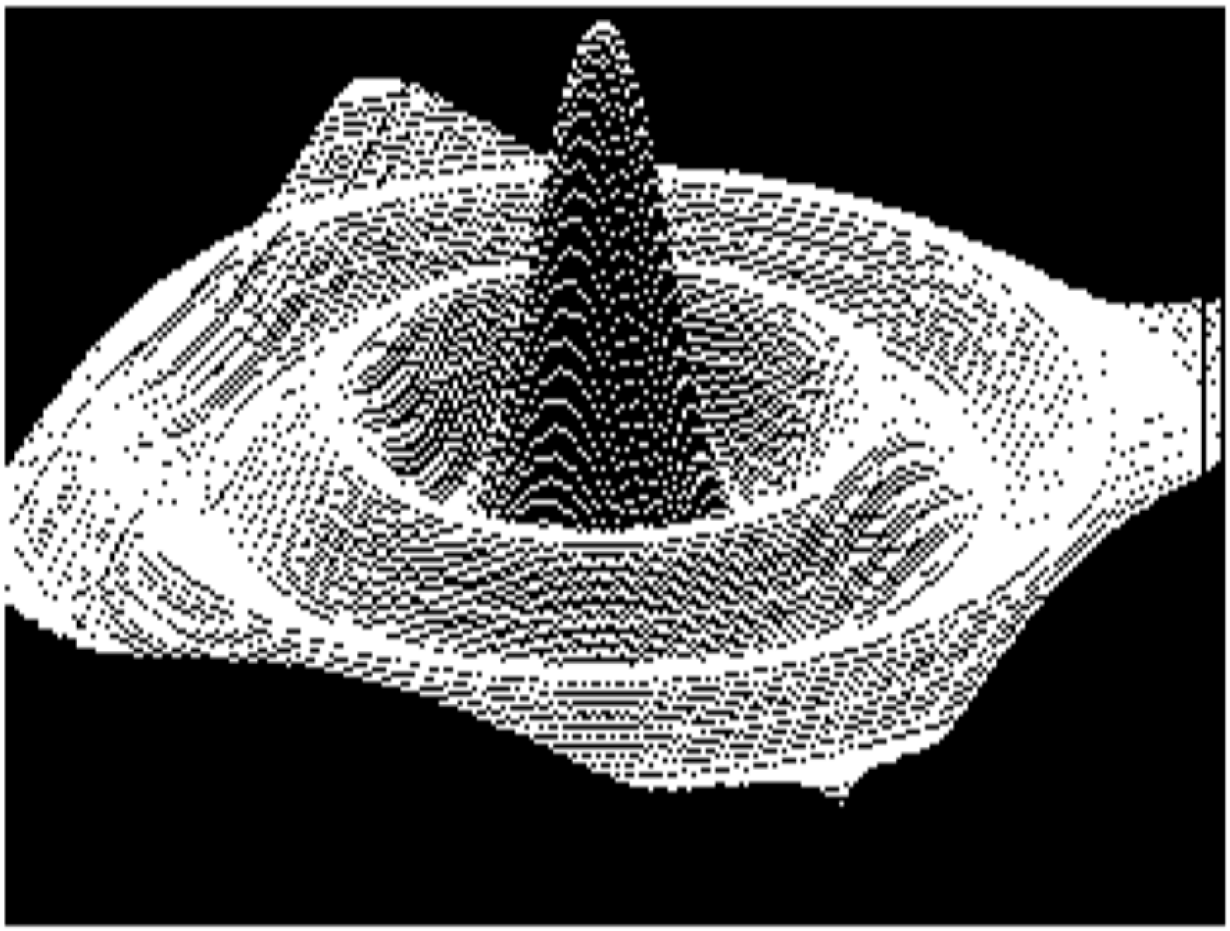
Thanks also goes to William Astle for writing the awesome 6809 assembler lwasm and for allowing me to include the binary versions with my compiler.

Thanks also goes to Boisy Pitre for allowing me to include his decb (CoCo DISK image) tool with the binary version of my compiler.

Thanks to bluetip

Example Programs

This is a tweaked version of James Diffendaffer's 3D plot program that I converted from working on a CoCo 3 to work on a CoCo 1 & 2



Original program (EXTENDED COLOR BASIC):

```
0 CX=255:CY=192:PMODE 4,1:PCLS:SCREEN 1,1
1 DIM R(255):FOR I=0 TO CX:R(I)=CY:NEXT I:GOTO 10
2 R=SQR(X*X+Y*Y)*1.5: IF R=0 THEN F=90:GOTO 4
3 F=90*SIN(R)/R
4 A=10*X+125-5*Y:B=5*Y+2.5*X+93:RETURN
10 FOR Y=10 TO -10 STEP -0.1
70 FOR X=10 TO -10 STEP -0.1
80 GOSUB 2
82 IF A<0 THEN A=0
83 IF A>255 then A=255
84 IF R(A)>B-F THEN R(A)=B-F:PSET(A,B-F)
90 NEXT X,Y
101 GOTO 101
```

Below is the same program modified to work with the compiler. The only differences are: (The SDECB IDE auto indents the code)
1) The variables are declared with the DIM command
2) **GMODE** instead of **PMODE**
3) **GCLS** instead of **PCLS**
4) Line 84 the **SET** command is used which includes a **colour value**, instead of **PSET**

```
' compileoptions -s20

' Declare the format Type of numeric variables
DIM CX AS _UNSIGNED _BYTE, CY AS _UNSIGNED _BYTE
DIM R(255) AS SINGLE
DIM A AS SINGLE, B AS SINGLE
DIM F AS SINGLE
DIM R AS SINGLE
DIM X AS SINGLE, Y AS SINGLE
DIM I AS _UNSIGNED _BYTE

0 CX = 255: CY = 192: GMODE 16, 0: GCLS: SCREEN 1, 1
1 FOR I = 0 TO CX: R(I) = CY: NEXT I: GOTO 10
2 R = SQR(X * X + Y * Y) * 1.5: IF R = 0 THEN F = 90: GOTO 4
3 F = 90 * SIN(R) / R
4 A = 10 * X + 125 - 5 * Y: B = 5 * Y + 2.5 * X + 93: RETURN
10 FOR Y = 10 TO -10 STEP -0.1
    70 FOR X = 10 TO -10 STEP -0.1
        80 GOSUB 2
```

```
        82 IF A < 0 THEN A = 0
        83 IF A > 255 THEN A = 255
        84 IF R(A) > B - F THEN R(A) = B - F: SET (A, B - F, 1)
90 NEXT X, Y
101 GOTO 101
```

This is a sample of using a 640x225, 4 colour mode on a CoCo 3

Thanks to bluetip on the CoCo Nation Discord server

```
' compileoptions -s20

' Declare the format of numeric variables
DIM XMAX AS _UNSIGNED INTEGER
DIM X AS _UNSIGNED INTEGER
DIM YMAX AS _UNSIGNED _BYTE
DIM Y AS _UNSIGNED _BYTE
DIM Z AS _UNSIGNED INTEGER

GMODE 159
PALETTE 0, 0
PALETTE 1, 63
PALETTE 2, 18
PALETTE 3, 36
GCLS 0
SCREEN 1, 0

XMAX = 639
YMAX = 224

FOR Z = 0 TO 32767
    SET (RND(XMAX), RND(YMAX), RND(2) + 1)
NEXT Z

FOR Y = 0 TO YMAX STEP 8
    LINE (0, Y)-(XMAX, Y), 1
NEXT Y

FOR X = 0 TO XMAX STEP 8
    LINE (X, 0)-(X, YMAX), 1
NEXT X

Finish:
GOTO Finish
```

TEST.BAS

CoCo 1/2 Hi-res dots

```
' compileoptions -s20

DIM X AS _UNSIGNED INTEGER
DIM I AS _UNSIGNED _BYTE

10 CLS: PRINT "WELCOME TO THE COCO COMPILER    TEST PROGRAM. IF
YOU ARE READINGTHIS THAN YOU HAVE SUCCESSFULLY COMPILED THE TEST
PROGRAM."
20 PRINT: INPUT "PRESS ENTER TO SEE THE FG6R     SCREEN FILL
WITH 20,000 DOTS"; A$
30 GMODE 16, 0: GCLS: SCREEN 1, 1
40 FOR X = 1 TO 20000
    50 SET (RND(256) - 1, RND(192~%%) - 1, 1)
60 NEXT X
70 SOUND 100, 20
80 SCREEN 0, 0
90 CLS
100 PRINT: PRINT " ALL GOOD, NOW COMPILE YOUR OWN   BASIC
PROGRAMS AND HAVE FUN"
110 PRINT: PRINT: PRINT "        COCO FOREVER!!!"
120 FOR I = 1 TO 200 STEP 25
    130 SOUND I, 1
140 NEXT I
```

GETFSIZE.BAS

Get the size of a file stored on the actual SD card's filesystem on the CoCoSDC (not the files in a .DSK image)

```
' compileoptions -s20

' Print length of a file on the CoCo SDC

DIM FileLength AS _UNSIGNED LONG ' LONG is a 32 bit number

CLS
PRINT "ENTER THE FILENAME TO GET"
INPUT "THE SIZE OF"; Filename$

SDC_OPEN Filename$, "R", 1 ' Open file for reading
A$ = SDC_FILEINFO$(1) ' Get fileinfo in A$
SDC_CLOSE (1) 'close file

PRINT: PRINT "RAW HEX CODES OF FILE INFO:"
FOR I = 1 TO LEN(A$)
    PRINT RIGHT$("0" + HEX$(ASC(MID$(A$, I, 1))), 2);
NEXT I
PRINT
' Size is stored LSB first
FileLength = ASC(MID$(A$, 29, 1)) + ASC(MID$(A$, 30, 1)) * 256 +
ASC(MID$(A$, 31, 1)) * 65536 + ASC(MID$(A$, 32, 1)) * 16777216

PRINT "LENGTH OF FILE: "; Filename$
PRINT "IS:"; FileLength; " BYTES"

Finished:
GOTO Finished
```

BASASSEM.BAS

Interface BASIC with assembly code

```
' compileoptions -s20

' Example how to interface BASIC and assembly code

DIM N AS INTEGER

5 CLS: PRINT @32,
10 INPUT "ENTER A NUMBER FROM -32000 TO 32000"; N
20 PRINT N; "+ 7 IS";
30 GOSUB 1000 ' Goto assembly code and do an add
35 PRINT N
40 INPUT "WHAT IS YOUR NAME"; Name$
50 GOSUB LearnStrings ' Copy the Name$ string to the top of the
screen
60 PRINT "NICE NAME "; Name$
70 END

1000 ' Example of some assembly code

' ADDASSEM:
; Comments in assembly code must start with a semicolon or an
asterisks
; Comment is BASIC code must use apostrophe or REM
; Let's access the BASIC numeric variable N
; and multiply it by 2
    LDD     _Var_N  ; Get the signed 16 variable N into D
    ADDD    #$0007  ; Add 7
    STD     _Var_N  ; Store the result back into variable N
; Can have the RTS here or use RETURN just past the ENDASSEM:
' ENDASSEM:
RETURN

LearnStrings:
' ADDASSEM:
; Example of how to access a string variable in assembly code
; Copy the string to the top of the text screen
    LDB     _StrVar_Name    ; Get the length of the string
    LDU     #_StrVar_Name+1 ; Point at the actual string data
    LDX     #$400           ; Point at the top left corner of
the text screen
!   LDA     ,U+             ; Get the next character
```

```
    STA     ,X+             ; Store the character on the screen
    DECB                    ; Decrement the counter
    BNE     <               ; Loop until counter is 0
    RTS
' ENDASSEM:

' The compiler adds a prefix to variables (which you can see at
the beginning of the .asm file it genereates)
' numeric variables are prefixed with _Var_ - numeric variables
are signed 16 bit numbers
' string variables are prefixed with _StrVar_ - strings are
stored beginning at the address of the variable
' the format is - byte 0 is the length of the string, the actual
string starts at byte 1
'
```

PAGEFLIP.BAS

Example of GMODE, GCOPY, CIRCLE and PAINT commands to do simple animation

```
' compileoptions -s16

' Example of doing page animation
' Must always setup the number of pages you will use as the
' first GMODE command in your program
' With this many pages you will require 64k and will need to use
' the cc1sl (CoCo Super Loader program) see the Manual.pdf

DIM S AS _BYTE ' Range of -128 to 127
DIM X AS _UNSIGNED _BYTE ' Range from 0 to 255
DIM Frames AS _UNSIGNED _BYTE ' RANGE 0 to 255

' First GMODE sets graphic mode and reserves 7 pages 0 to 6
GMODE 16, 6
GMODE , 0 ' Set the active graphics page to 0
GCLS 0 ' Colour the screen colour 0 which is black
Frames = 6 ' Number of frames to use
FOR X = 1 TO 2000 ' Draw some dots on the screen
    SET (RND(256) - 1, RND(192) - 1, 1)
NEXT X

FOR X = 1 TO Frames ' Copy page 0 to all the other pages
    40 GCOPY 0, X
NEXT X

' Draw and fill a circle on the screen
50 X = 20
FOR S = 0 TO Frames
    GMODE , S
    CIRCLE (X, 20), 11, 1
    PAINT (X, 20), 0, 1
    X = X + 10
NEXT S
' Ping Pong animation frames (Loops forever)
100 FOR S = 0 TO Frames
    GMODE , S: SCREEN 1, 1 ' Screen shows the selected page
    SOUND 10, 5
NEXT S
FOR S = Frames - 1 TO 1 STEP -1
    GMODE , S: SCREEN 1, 1 ' Screen shows the selected page
```

```
    SOUND 10, 5
NEXT S
GOTO 100
```

SDCDIR.BAS

Show a directory of the files on the CoCoSDC FAT32 filesystem

```
' compileoptions -s20

DIM X AS _BYTE
DIM Y AS _UNSIGNED _BYTE

CLS: PRINT "SDC ACCESS"
CurrentDir$ = "*.*" ' Start at the top folder on the SDC
' CurrentDir$="TEST1/*.*" ' Start at the folder TEST on the SDC
PRINT "PATH:"; CurrentDir$
' Set the initial directory location
X = SDC_INITDIR(CurrentDir$): GOSUB CheckForError
PRINT "DIRECTORY:"

NextPage:
X = SDC_DIRPAGE(0): GOSUB CheckForError ' Get 16 directory
entries
Y = 0
WHILE Y < 16
    A$ = SDC_DIRLIST$(Y) ' A$ = The 16 byte entry
    IF ASC(LEFT$(A$, 1)) = 0 THEN END ' If entry is blank then
we've reached the last entry
    FOR X = 1 TO 8: PRINT MID$(A$, X + 1, 1);: NEXT X
    PRINT ".";
    FOR X = 9 TO 11: PRINT MID$(A$, X + 1, 1);: NEXT X
    PRINT
WEND
GOTO NextPage

CheckForError:
IF X <> 0 THEN PRINT "ERROR NUMBER IS:"; X: END
RETURN
```

CONSTANT.BAS

Show an example of using constants in your program.

```
' compileoptions -s20


' Declare a numeric constant approximately equal to the ratio of
a circle's
' circumference to its diameter:
Const PI = 3.141593

' Declare some string constants:
Const circumferenceText = "THE CIRCUMFERENCE OF THE CIRCLE IS"
Const areaText = "THE AREA OF THE CIRCLE IS"

Do
    Input "ENTER THE RADIUS OF A CIRCLE OR ZERO TO QUIT"; radius
    If radius = 0 Then End
    Print circumferenceText; 2 * PI * radius
    Print areaText; PI * radius * radius ' radius squared
    Print
Loop
End
```