BCPL

For the BBC Micro

Lardo Boffin

BCPL Overview

This document shows you how to get started with BCPL on the BBC Micro. It assumes you know what BCPL is (Basic Combined Programming Language – a precursor to B and then C) and how to actually program in a language such as BASIC.

What you need to get going

- A BBC Micro, model B or better (or emulator such as BeebEm)
- A fast modern storage device such as an MMC card (SD card reader), Gotek, Econet or CF Card emulating a HDD in ADFS
- Either Sideways RAM (16K), a modern EEPROM (e.g. a Boobip EEPROM chip installed) or a spare ROM slot in which to put an EPROM for the BCPL ROM
- Ideally a 6502 co-processor of some sort (e.g. a Pi based co-proc) otherwise you will have very little RAM for programs!

In order to learn the language:

- A - read the book “BCPL the language and its compiler” by Martin Richards and Colin Whitby-Stevens
- B - play with it

There are only about 160 pages so there is not a huge amount of reading

Reading this guide will hopefully help you actually use it in practice on the BBC Micro. You can download some “.SSD” files from GitHub (LardoBoffin/BBC-BCPL_Overview/MasterDiscs) with the discs setup ready for use as described in BCPL User Guide – this includes BCPL_Source_Disc.ssd and BCPL_Test_Disc.ssd.

This document assumes your storage device either has the capability to use two ‘disc drives’ under DFS or be able to use ADFS and hold enough files in a single directly to have all necessary files present in one place. If you only have access to a single DFS drive expect to do a lot of disc swapping!

We will be assuming from here on in that you are using two drives in DFS, e.g. with an MMC card reader.

Initially full instruction will be given as to when to switch drives in order to go from compiling (source disc) to running (test disc) etc. but after a while this should be second nature. If you are running ADFS and have put the files from both discs into a single folder (or downloaded the ADF image and saved yourself the hassle) then there is simply no need to type * DRIVE ## from time to time to switch drives as all the files will already be present.

In the folder MasterDiscs there are:

Folder	File	Notes
ADFS	BCPL_Master_Discs.adl	All three DFS disc images but in a single image with sub directories
DFS	BCPL_Master_Disc.ssd	All of the standard BCPL files in a single image. There are 31 files so not much use in DFS!
	BCPL_Source_Disc.ssd	The 16 files recommended for building BCPL programs
	BCPL_Test_Disc.ssd	The 12 files recommended for running and testing BCPL programs
	BCPL_Calculations_Package.ssd	The calculations package files
	BCPL_Standalone_Generator.ssd	The runtime builder and files. This also includes NEEDCIN and EX
ROM	BCPL 7.0.rom	The BCPL ROM image file

Hello, World

The first thing you will want to do is create a hello world program. Apparently the first ever “Hello, World” program was written in BCPL (if the internet is to be believed).

Starting BCPL

To start off with we need to boot up BCPL. To do this from BBC BASIC type *BCPL. If you have the BCPL ROM as the highest priority ROM, then it will boot up in BCPL. The prompt will now be:

!

and not the usual “>”.

What can I type now?

You can still use all * commands (e.g. *DIR, *DRIVE #, *FX, *BASIC etc.) but not BASIC commands such as MODE. You will be in whatever MODE you were in before typing *BCPL with no immediate way to change this other than switching back to BASIC (by typing *BASIC), changing the mode and then typing *BCPL again. Later on a program will be written that allows you to type MODE 7 or similar from the command prompt.

Say hello to ED

In order to create your first program, you will need a text editor to create a source file – this will be assumed to be ED, the editor that comes with BCPL, so load up the BCPL Source SSD disc image.

To create a new, blank file type (don't type in the "!", this is the command prompt!!!):

!ED HelloW (press return)

If a file of the name already exists, then it will open the file instead. Otherwise you will get a blank screen other than the words "New File" at the bottom of the screen.

Type the following:

```
GET "LIBHDR"
LET start() BE
        WRITES("Hello, World*N")
```

Don't worry about upper and lower case – this is not C.

Press Ctrl+F8 to save and exit the editor. This will save the file to the STORE, not the filling system.

The STORE and saving files

The program will be saved in the STORE; this is an area in RAM set aside for work in progress files. You will not see the file in DFS if you type *. at this point. If you type:

!STORE

you will see the files currently held in the STORE in RAM. More on this later. Much later. You can compile from the STORE and not save to disc but if your beeb crashes you have lost everything, so best to save to disc – type:

!SAVE HelloW

Don't put double quotes around the file name. If you use a different filename at this stage (e.g. just type !SAVE Hello) BCPL will get confused as it will not be able to find the file Hello in the STORE in order to save it to disc. You will get Error 17.

Building the program

Having saved to disc type:

!BCPL HelloW HW

You will see:
BCPL – RCP V2.2
Text read
RCP CINTCODE generation

CINTCODE size = 20 words
!
This means the file has been successfully compiled and a CINT code (the 'compiled' code that can be run directly under BCPL) file produced (and currently only saved in the STORE). Any other message means either the wrong filename was typed, the file HW already exists or the text was typed incorrectly into ED. Check everything and try again!
If you typed the message a little differently the number of words may be different.

The line BCPL HelloW HW has three elements as shown in the table:

Section	Notes
BCPL	Calls the BCPL compiler
HelloW	The source file to be compiled
HW	The name of the target CINt code file, you choose this

Saving the CINT code file

Type:
!SAVE HW

This will copy the CINT code file from the STORE to the filling system. Again you do not have to do this but if there is a crash you have lost the compiled file.

Running the program

To run the program simply type:

!HW

You will see:
Hello, World
!

Your first BCPL program!

Hello, World with green screen coolness!

The first Hello, World program just uses the built in standard BCPL function WRITES("") to produce the output. It is possible to extend this and change the screen colour and MODE etc. to make it look prettier. This can be done by using the VDU command that has kindly been added to this version of BCPL. To do that however we need to tell the compiler that we are going to use it and where to find it. According to the user guide it can be found in the Section "VDU" of the library file LIB. So…

Go back to the previous hello world source SSD file and type:

!ED HelloW

Amend the program as follows:

```
NEEDS “VDU”
GET “LIBHDR”
LET start() BE
$(
        VDU(“22,0”)
        VDU(“19,1,2”)
        VDU(“17,1”)
        VDU(“12”)
        WRITES(“Hello, World*N”)
$)
```

Note that we are now putting $(and $) around the start function – this is because there is more than one line of code.

Then: -

- Press Ctrl+F8
- Type !SAVE HelloW
- Type !BCPL HelloW HW

Switching to the Test Drive

We were bound to need it sooner or later.

Type !*Drive 1 (to switch to the test drive SSD)
Type !SAVE HW

I feel the need, the need for NEEDCIN

Type !NEEDCIN HW LIB HWEXE

You should see something like:
222 word file HWEXE created
0 NEED(S) unsatisfied

Type !SAVE HWEXE
Type !HWEXE and watch the fun! Wow, such mode 0 green.

The line NEEDCIN HW LIB HWEXE has four elements as shown in the table:

Section	Notes
NEEDCIN	This command extracts the NEEDed CINt code from the library
HW	The is the BCPL cint code created by the BCPL ## ## command from earlier
LIB	This is the library file where the SECTION “VDU” is to be found

HWEXE	This is the name of the final file with the original BCPL cint code plus the library code. You choose what this is

From here on in I won't be prefixing anything with !, so saying type NEEDCIN #### is the same as saying type !NEEDCIN ####. The BCPL prompt should be familiar by now. ☺

As an exercise you can replace the VDU("22,0") – which I'm sure you know changes the screen mode – to use the command MODE(0) that has also been included in the Beeb BCPL version! Hint it will need another section.

Hello, World with green screen coolness and no BCPL ROM required

The second program introduces sections of the provided library, LIB, and NEEDCIN to combine your program with the library code. This program goes a little further and removes the need for the BCPL ROM to be present in order to run the program – it will be compiled as a run-time version that can be called from BASIC! Note that the SAG disc has had the files EX and NEEDCIN added to it to make life easier.

Go back to the previous hello world source SSD file and type:

ED TESTRT

Type in the following program into the new empty file:

```
//
//Filename TESTRT
//
//29/05/2016
//
//v1.0

NEEDS "VDU"
NEEDS "INTERP"
NEEDS "MODE"
NEEDS "RUNPROG"

GET "LIBHDR"

LET START() BE
$(
  MODE(0)
  VDU("19,1,2")
  VDU("17,1")
  WRITES("*N")
  WRITES("Hello world from BCPL Runtime!")
  WRITES("*N")
  RUNPROG("**BASIC")
```

```
  STOP(0)
$)
```

Don’t forget to Ctrl+F8 to exit and then SAVE TESTRT.

The key thing to note with this program is the additional NEEDS sections (see points 3, 4 and 7 below) at the start and the RUNPROG (see point 6 below) at the end. The NEEDS listing tells the compiler which sections of the runtime library to include when it is being built. The RUNPROG although not strictly necessary gracefully drops the user back into BASIC after running the program.

Standalone Generation Process

1. Create a very simple BCPL program and test it

2. Make any required changes for the SAG. This is basically removing any function calls that are not supported

3. List all of the library routines used in your program and note the section they belong to (see chapter 5 for a list) and add each one as a NEEDS "..."

4. Add NEEDS "INTERP" at the end of the list

5. Look at the IO options required (none for my simple program)

6. Decide on termination options. I went for a very simple return to basic by calling *BASIC at the end. It may well be fine to do nothing and force the user to do a CTRL+Break to exit, i.e. do nothing

7. Add all of the NEEDS either to main program or one of the files in it, or ideally a separate file (I added them to the main program file but will do it properly later on)

8. Compile the BCPL programs as normal but with the switch NONAMES as this reduces file size slightly, in my case "BCPL TESTRT RT NONAMES". Note that you do not do the normal NEEDCIN at this stage, unless you are including a library you have written yourself. All standard library stuff, e.g. VDU will be picked up from the run time library later on

9. Decide on whether to use SYSLIB1 or 2. This mainly comes down to file IO I believe. We are not using any so will stick with SYSLIB1

10. Time for NEEDCIN! Ideally use EX to run a commmand file (I have used this) to do all the heavy lifting of files

11. Use PACKCIN to remove unwanted stuff and reduce the file size a bit

12. Ignore the ROM stuff for now

13. Decide on the location of the global vector (effectively the load location of the program which needs to be at PAGE or above). I have stuck with &1900 assuming a DFS of some sort

14. Use FIXCIN to create the program (ignore ROMs for now)

15. Use FILETRN to copy the file to keep its execution address (I think I used *COPY which seems to work as well)

Say hello to EX (brother of ED?)

All of stages 9 through to 14 are managed by a single command file used in EX (a program that runs command files such as this). I took mine directly from the manual and it works well in this simple example: -

```
.KEY INFILE/A,OUTFILE/A
NEEDCIN <INFILE> SYSLIB1 $TEMP1
PACKCIN FROM $TEMP1 TO $TEMP2
DELETE $TEMP1
FIXCIN FROM $TEMP2 TO <OUTFILE> GV=1900
DELETE $TEMP2
```

To create the file above type:

ED COMPILE (this will create a blank file, then type in the above, carefully, Ctrl+F8 to exit, switch to drive 1 and then SAVE COMPILE)

The .KEY line handles the file names. The NEEDCIN line handles points 9 and 10. The PACKCIN line handles point 11. The DELETE lines tidies up an unwanted temp file. The FIXCIN line handles points 13 and 14.

So assuming that our original BCPL compiled file is called RT (and the EX command file is called COMPILE) I would call -

EX COMPILE RT HELLO

Building and running the Standalone program

The complete sequence to build and deploy this file is therefore: -

- Load BCPL_Source_Disc into drive 0
- Load BCPL_Standalone_Generator into drive 1 (we don't need BCPL_Test for this project)
- On drive 0 type "BCPL TESTRT RT NONAMES"
- Switch to drive 1 and "SAVE RT"
- On drive 1 type "EX COMPILE RT HELLO"

After this has finished running you end up with a file called HELLO that can be run outside of BCPL by typing *HELLO

In terms of file sizes, the initial RT (compiled file) is &90 (144) bytes long and after being converted to runtime is &F82 (3970) bytes long. This is therefore &EF2 (3826) bytes of interpreter and function code. While this is quite a jump in size it is impressive given that it includes the BCPL interpreter! Don't be expecting to do anything useful in MODE 0 on a non-expanded machine though... Adding further function calls will increase the size of the end file as it will need to include more library functions.

When run the HELLO program should change the screen to MODE 0, turn the text green and say hello. It then calls RUNPROG("**BASIC") to terminate and return control to BASIC.

There are a whole raft of options and further settings but this is good enough for now for a very simple demo.