User Guide

Scamp-B

Introduction

Scamp-B is a graphical simulation offering a window into those halcyon days of computing when data was input through switches, and register contents were displayed on banks of lamps. A time when the programmer had an intimate relationship with the machine at its most basic level.
The inspiration for Scamp-B came from my involvement with Phil Tipping's PlasMa hardware project, a real machine with real lamps and switches and a roadmap to emulate not just a mainframe style central processor, but also a selection of peripheral devices. PlasMa is designed to run a selection of instruction sets offering varying capabilities and complexity. Two of these are based on Princeton University's Toy-A and Toy-B code sets. Scamp-B has intercepted Toy-B, but renamed it TOS-B to allow for divergence without compromise.
Scamp-B aims to improve the programmer's experience through the integration of a 256 word memory map display and an assembler editor. The assembler which also front-ends the PlasMa simulator PlasMaSim, is another inspiration from the PlasMa project.

Description

Scamp-B consists of 16 general purpose 16 bit registers, an instruction register and a program counter. The program counter is also 16 bits but the top, most significant, 8 bits aren't required as the memory's maximum address is 256, hex 'FF'. The registers are displayed in binary, on or off, format, whereas the memory display uses a hexadecimal representation of its contents. Calculations are performed using two's complement 16 bit integer values in the range -32768 to +32767.

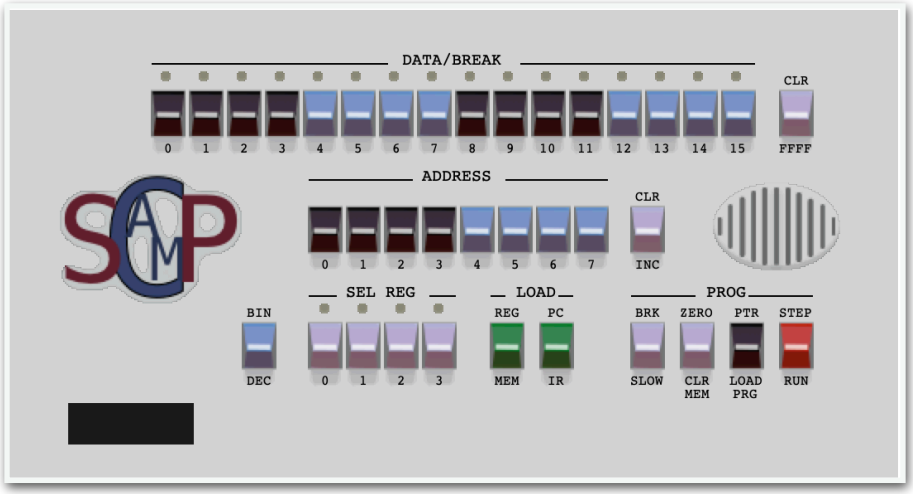
Layout

Scamp-B uses three windows to display its functionality. The main window contains all the key switches for program control and a display of all the registers. The Memory Map window displays the memory's contents in a 16x16 table array and the Oper window (operators console), with its eight key switches facilitates program input and output. The assembler editor is controlled via its own menu.

Operation

When the Scamp-B program is loaded it automatically runs a TOS-B program called 'programb.plh'. This is for demonstration purposes and may be deleted or renamed if not required.

Main Controls



- **DATA/BREAK** keys
  Principally used for setting a hexadecimal word's worth of data for copying to registers or memory. They are also used for setting a register break pattern or a PlasMaSim program name.
- **CLR/FFFF** key
  Sets the DATA/BREAK keys to 'FFFF' or clears them to all off.
- **ADDRESS** keys
  Used solely for setting a memory address for writing DATA.
- **CLR/INC** key
  Either increments the address set on the ADDRESS keys or clears them to all off.
- **BIN/DEC** key
  Switches the register display to either binary mode or hexadecimal/decimal mode.
- **SEL REG** keys
  Used to set a register number (0-F) to write DATA to or to BREAK on.
- **LOAD** keys
  One key directs data from the **DATA** keys to **REG**ister or **MEM**ory. The other directs data from the **DATA** keys to the Program Counter(**PC**) or Instruction Register**(IR**).
- **PROG** keys
  Used for controlling program running.
  SLOW key slows down program execution and disables Oper clear.
  BRK key enables break on register value.
  ZERO key zeros all the registers, clears the Oper, resets the Program Counter and loads the Instruction Register from memory location zero.
  CLR MEM key sets all memory locations to hex 0.
  LOAD key loads the program set on the DATA keys or opens a file dialog.
  PTR key loads data from the Paper Tape Reader.
  RUN key runs the program loaded in memory.
  STEP key executes the current instruction and moves the Program Counter to the next one.

NB.

Keys with labels above and below have two operations, whereas a single label denotes a single action. The keys sense mouse clicks above or below the white centre line depending on their operation.

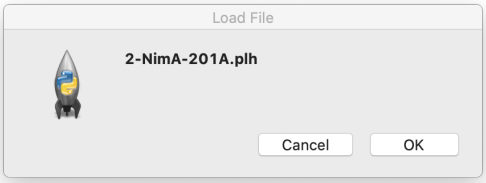
The black rectangle at bottom left displays the Scamp-B program run time.

Loading a Program

If the DATA keys are set to all OFF and the LOAD PRG key clicked, a file dialog is opened showing files of type '.plh'. These are assembled source files in sixteen bit hexadecimal format containing TOS-B codes and data.

Existing programs may also be loaded from the DATA/BREAK keys provided that they are named inline with the PlasMaSim scheme, i.e. four hexadecimal characters followed by file type '.plh', with the first character always '2' denoting a TOS-B program. For example '2002.plh', 2901.plh','2FFF.plh'. These names can be enhanced to be more descriptive as long as they contain the four hex characters and type '.plh'. On clicking the LOAD PRG key, the DATA/BREAK keys are matched with filenames in Scamp-B's local directory.

If the keys are set to read a non-existant program or there are two or more programs that match, an error is displayed on the memory map screen. If a unique program is found then a dialogue box appears as shown below.



Programming

Using the Keys

Programs can be entered into memory using the DATA and ADDRESS keys.

Here is a very simple three instruction example that writes to the Oper.

We will write a decrementing loop that prints the loop value to the Oper. To do this we will use the 'djnz' instruction that decrements a register, checks it for none zero then jumps to an address. A full list of instructions is shown in, TOS-B Instruction Set

First we need to load up a register with the number of times the loop should run. For this we'll use the 'lda' instruction that loads a value into a register.

Then we'll write our decremented loop number to the Oper using the system 'Oper Write' instruction.

Finally we'll check the current register value for zero, if it's not, decrement it and loop back using the 'djnz' instruction.

The register we'll use is REG-4

First load REG-4 with decimal 15 (hex F) using 'lda' instruction 'B40F'.

Set the DATA/BREAK keys to hex 'B40F' = `1011.0100.0000.1111` and the ADDRESS keys to '0' (all off). To write the data to the memory location click the green MEM key, and hex 'BF04' will appear in the first location in the Memory Map Window.

The Oper Write instruction code is hex '4402'. So set `0100.0100.0000.0010` on the DATA/BREAK keys, increment the ADDRESS keys to `0000.0001` and write to memory with the MEM key.

Now write the 'djnz' instruction hex '7402' to memory location 2 and finally write a Halt instruction hex '0000' to address 3.

So to summarise, the program should consist of the following instructions.

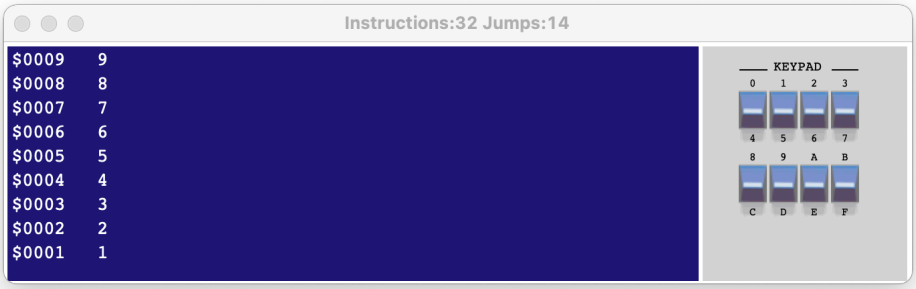
address 0	B40F	load REG-4 with value 'F'
address 1	4402	write value in REG-4 to the Oper
address 2	7401	decrement REG-4 if not zero jump to address 1
address 3	0000	halt

The Memory Map screen should now look like the graphic below.

To initialise the program we have just written we need to set the Program Counter to the first address. To achieve this click on the ZERO key. You will see a box appear around the first instruction and its binary value appear in the Instruction Register.

Now the program can be run by clicking on the RUN key, or stepped through one instruction at a time with the STEP key.

If everything is correct the decremented loop values will appear on the Oper as shown below on run completion. The hexadecimal value (preceded by $) followed by its decimal equivalent.



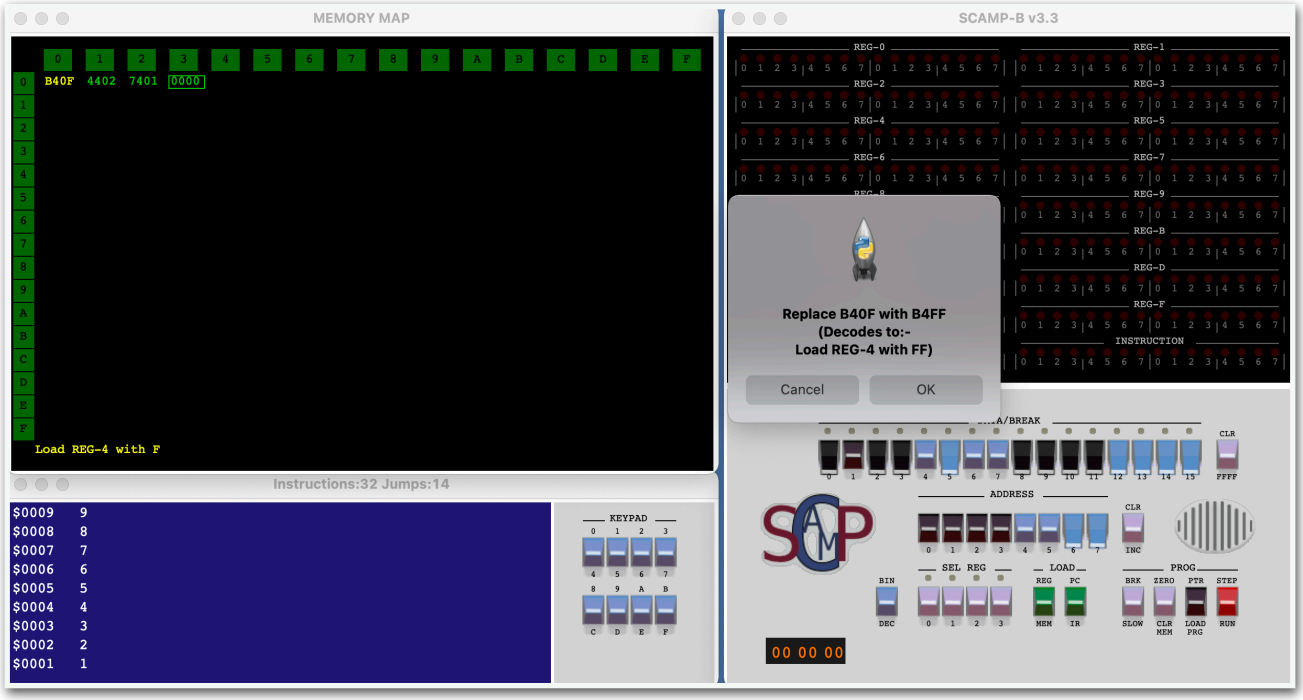
To rerun the program click on the ZERO key followed by the RUN key.

Memory Editing

In addition to using the DATA and ADDRESS keys to write to memory, the mouse can be used to select an address to change, but cannot be used to write to blank memory locations.

Using the programming example from before, we could decide to change it to loop more times by setting a larger number in REG-4 at the start.

To change the ‘lda’ instruction ‘B40F’ to ‘B4FF’ we set the DATA keys to ‘B4FF’. Now RIGHT-CLICK on the ‘lda’ instruction at memory location ’0’, the text will change to yellow and a dialogue box will be displayed as shown below.



The dialogue box shows a decode of the intended instruction whilst the yellow text at the bottom of the screen shows the unchanged instruction decode.

Click OK in the dialogue box to commit the change. Then click ZERO key followed by RUN key to rerun the program. You will notice that certain values display a character to the right of the decimal value. This is because Scamp-B's Oper write instruction displays ASCII decodes where it can.

Controlling a Program

As already mentioned in Main Controls, a program can be executed one instruction at a time using the STEP key or can be slowed down using the SLOW key, or RUN normally.

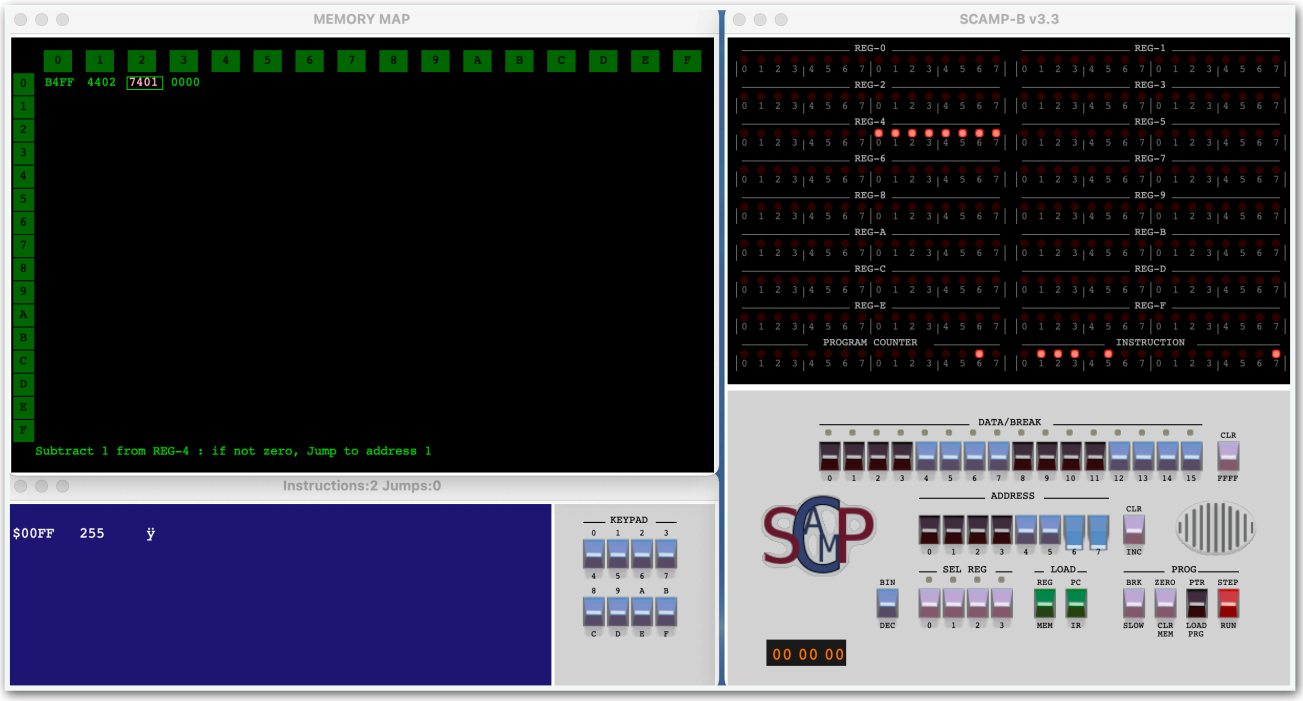
Also break points can be set to stop a program at a particular point in its execution.

Setting a break on instruction

As a program runs it loads an instruction from memory into the instruction register which is then executed. Setting a break on instruction stops the program running at the point where the instruction is about to be executed.

To set a break on instruction simply click on the instruction's location in memory where running must stop. The instruction will change colour to indicate a break is set. Multiple breaks may be set in this manner. To clear a break click on it again.

So we could choose to stop our program on the 'djnz' instruction and look at register 4 to see its value before the jump instruction is tested for zero. The graphic below shows a break set at memory address hex '02' after the RUN key has been clicked.



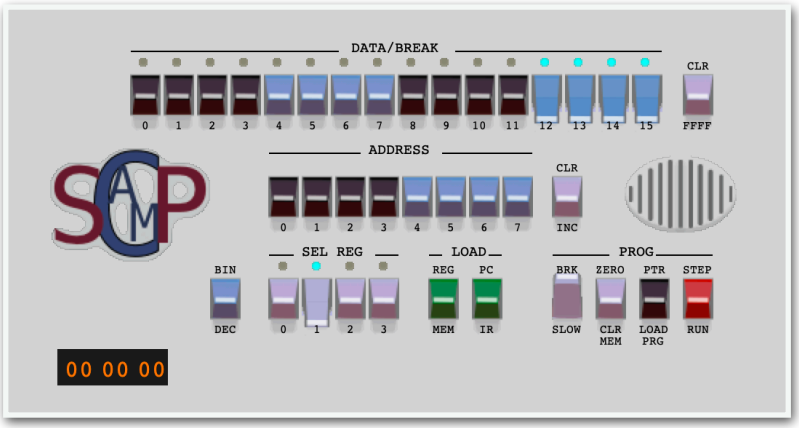
The program has stopped after the first Oper write whilst register 4 has not yet been decremented and tested for zero. Repeated clicks of the RUN key will show register 4 counting down with subsequent writes made to the Oper.

The text displayed at the bottom of the Memory Map screen shows an interpretation of the instruction ('7401') stopped on by the break..

NB. The instruction interpretation displayed on the screen's bottom line is replaced with the break instruction whilst the mouse button is held down.

Setting a break on register value

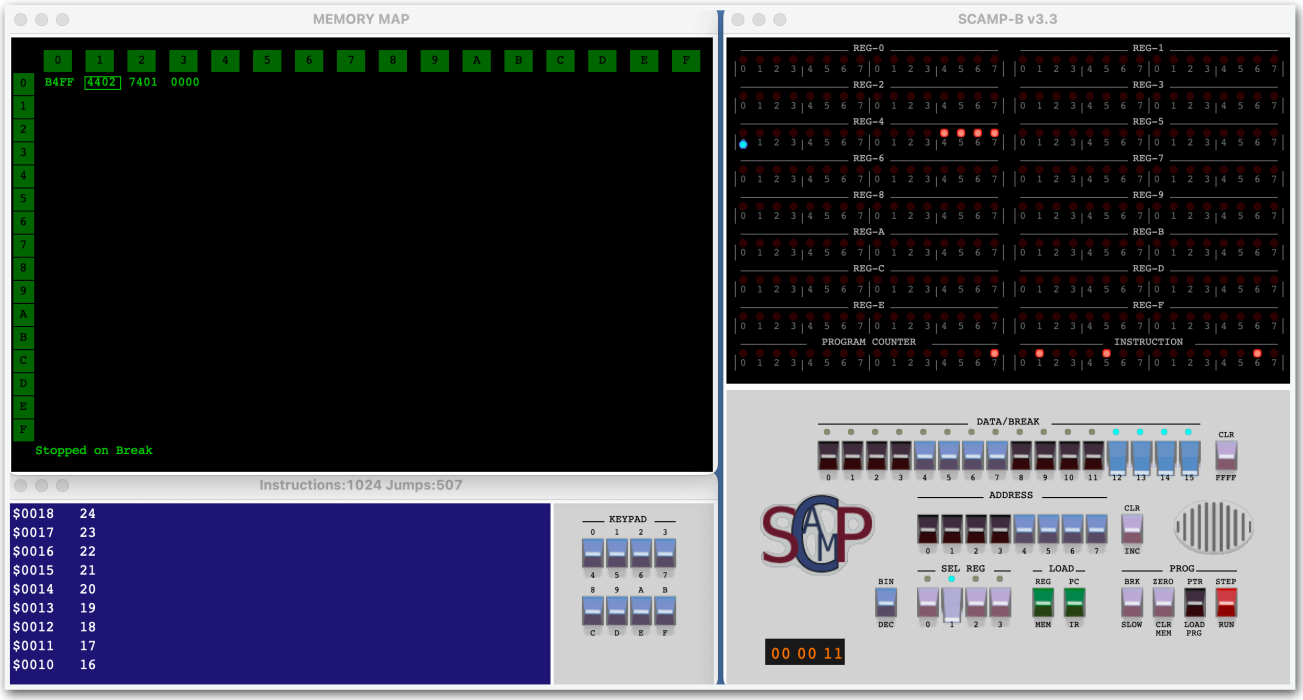
This facility is useful for trapping a point in a program where a register contains a particular value. To set a break on register value the value is input on the DATA/BREAK keys, the register number is input on the SEL REG keys and the BRK key is set. Using our program again we could choose to stop the program when register 4 reaches value '0F'. To achieve this, unset the Memory Break by clicking on it. Then set the Register Break as shown below.



The DATA/BREAK keys are set to ‘000F’ which is the value we want to stop on, the SEL REG keys are set to ’4’ for register 4, and the BRK key is switched up. The blue lights are illuminated showing the break value set.

Unlike the break on instruction example that we saw before, the instruction will be executed and the register loaded.

When the RUN key is clicked the program will run until the break point is detected and the following graphic will be displayed.



This shows register 4 has stopped with contents ‘000F’ and register 4 now has a blue lamp displayed. The decrement and jump instruction has been obeyed, and the Oper write instruction ‘4402’ is loaded into the Instruction Register.

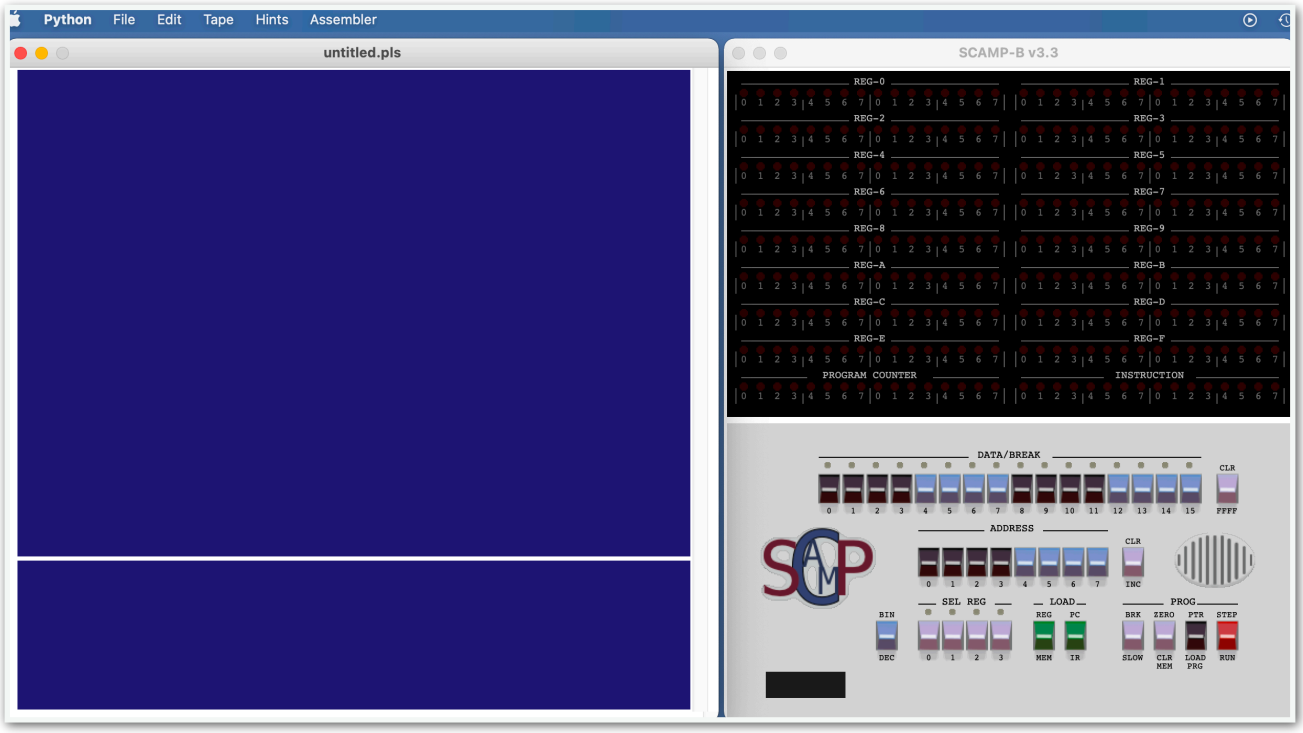
On clicking the STEP key, the ‘4402’ instruction writes the current value of register 4 to the Oper, and the break on register value is cancelled.

To continue running the program click the RUN key

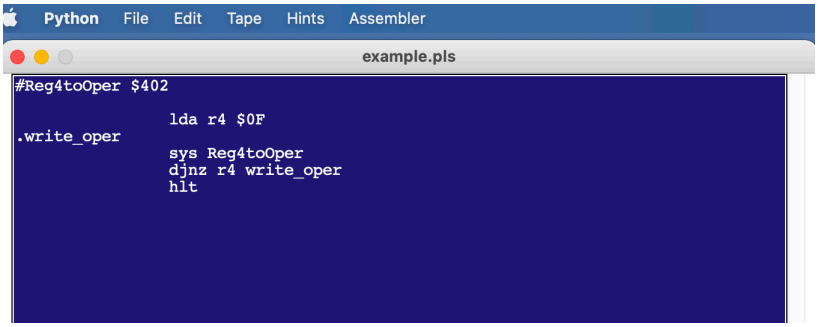
NB. The blue lamps indicate the break values as set by the BRK key. The BRK key must be switched up to implement the break. To change a break value the BRK key must be switched off and on again to re-read the DATA/BREAK and SEL REG keys.

Writing Assembly Code

When Scamp-B loads it runs the start-up program ‘programb.plh’, and an assembly editor window is created but iconised. If ‘programb.plh’ is renamed or deleted Scamp-B will load with the assembly editor window open, as shown in the graphic below. This shows the Mac version where the editor menu is displayed at the top of the screen from which the editor can be opened. On Windows and Linux the menu is integrated with the editor window so must be opened from the icon.



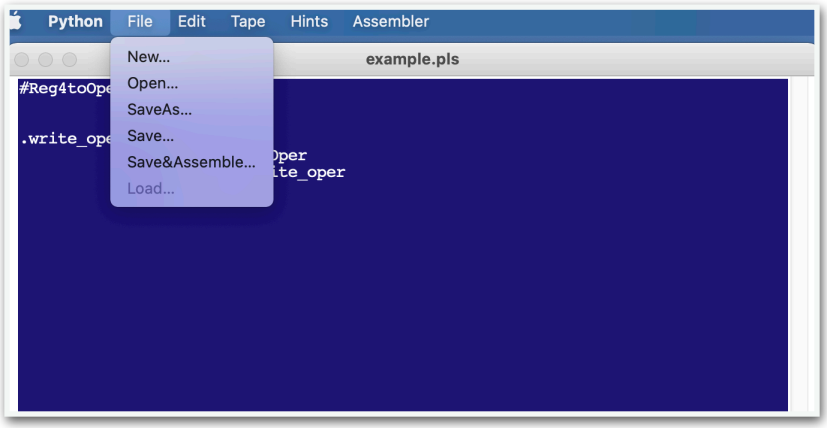
If we take the simple example from before and use the mnemonics taken from the TOS-B instruction set, our program might look like the following graphic.



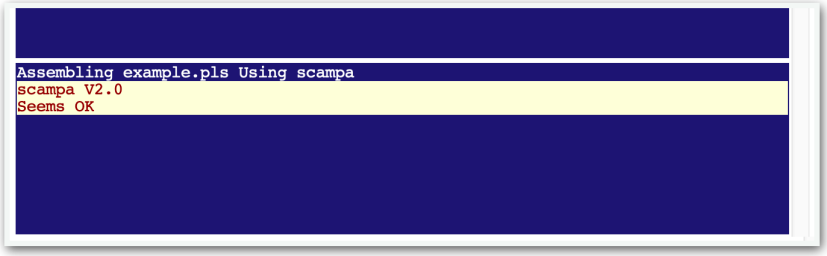
Some of the benefits of writing in assembly language are demonstrated here. The first line '#Reg4toOper $402' is an Equate statement that assigns the name Reg4toOper to the value $402 (hexadecimal) enabling that value to be referenced by name.

Further down the screen is a Label '.write_oper' that assigns a name to a code address. For further information see the Assembler Basics Guide.

To convert the assembly language 'example.pls' to TOS-B hex code 'example.plh' use the File menu option 'Save&Assemble…' as shown below.



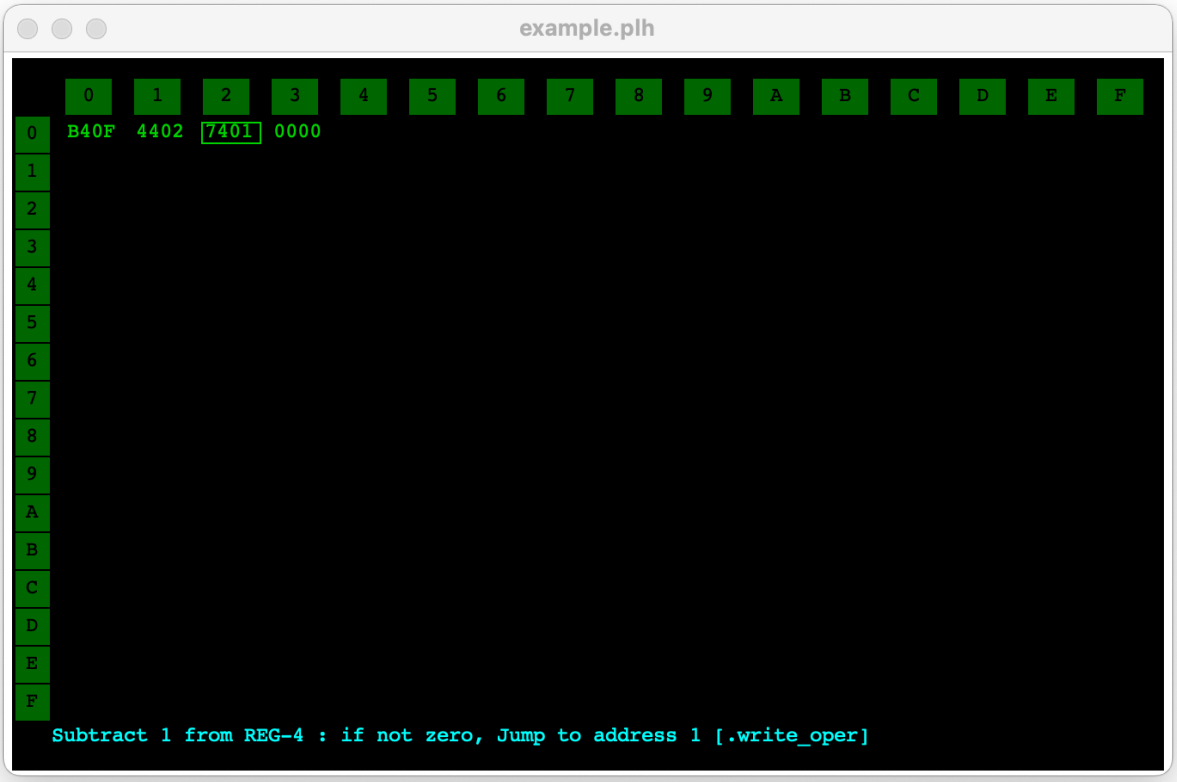
If there are no detected errors in the '.pls' file the lower console screen will show a Seems OK message.



The TOS-B hex code produced can now be loaded by selecting File menu 'Load…'. This action iconises the editor window.

To further develop the program open the editor window, change the assembly language instructions, re-assemble and re-load.

When stepping through a program that has been assembled, the prompt line at the bottom of the Memory Map window will be coloured cyan. This indicates that there is a map (.plm) file present for the program. This adds additional information on the prompt line where it finds a match between a code address and a label. The graphic below shows an example of this.



It can be seen here that address 1 has the label [.write_oper].

This facility may sometimes cause confusion, where a value loaded into a register happens to map to a valid address the label will be shown on the prompt.

Assembler Editor Menus

File Menu:

New	Create a new .pls file.
Open	Open an existing .pls file.
SaveAs	Save current .pls file to a new name.
Save	Save current .pls file.
Save&Assemble	Save current .pls file and create .plh file.
	N.B. File containing errors will still be saved but .plh file won't be created.
Load	Load the assembled .plh file into memory for execution.
	This option is greyed out if the editor thinks there is nothing to load.

Edit Menu:

Copy	Copy selected text to the copy buffer.
Paste	Paste the copy buffer at the cursor location.
UnDo	Undo previous edits.
ReDo	Re-do previous undos.
Clear Selection	Delete selected text.
Clear All	Clear the editor window.

Tape:

None	No paper tape file is produced
PTR8	The assembler creates an 8 bit paper tape image file.

Hints:

Off	Hints are disabled
On	Assembler mnemonics create a pop up help (experimental)

Assembler:

scampa — Selects ‘scampa’ as the assembler program.

Creates

- Hex code file .plh
- Map file containing equates and labels .plm
- Optional 8 bit paper tape file .pl8

plasm2 — Selects ‘plasm2 as the assembler program.

Creates

- Hex code file .plh
- Listing file containing detailed interpretation of source, helpful when fault finding .pll
- Optional Map file containing sorted equates and labels .plm (always selected by Scamp-B).
- Optional 8 bit paper tape file .pl8
- Optional 7 bit plus parity paper tape file .pl7

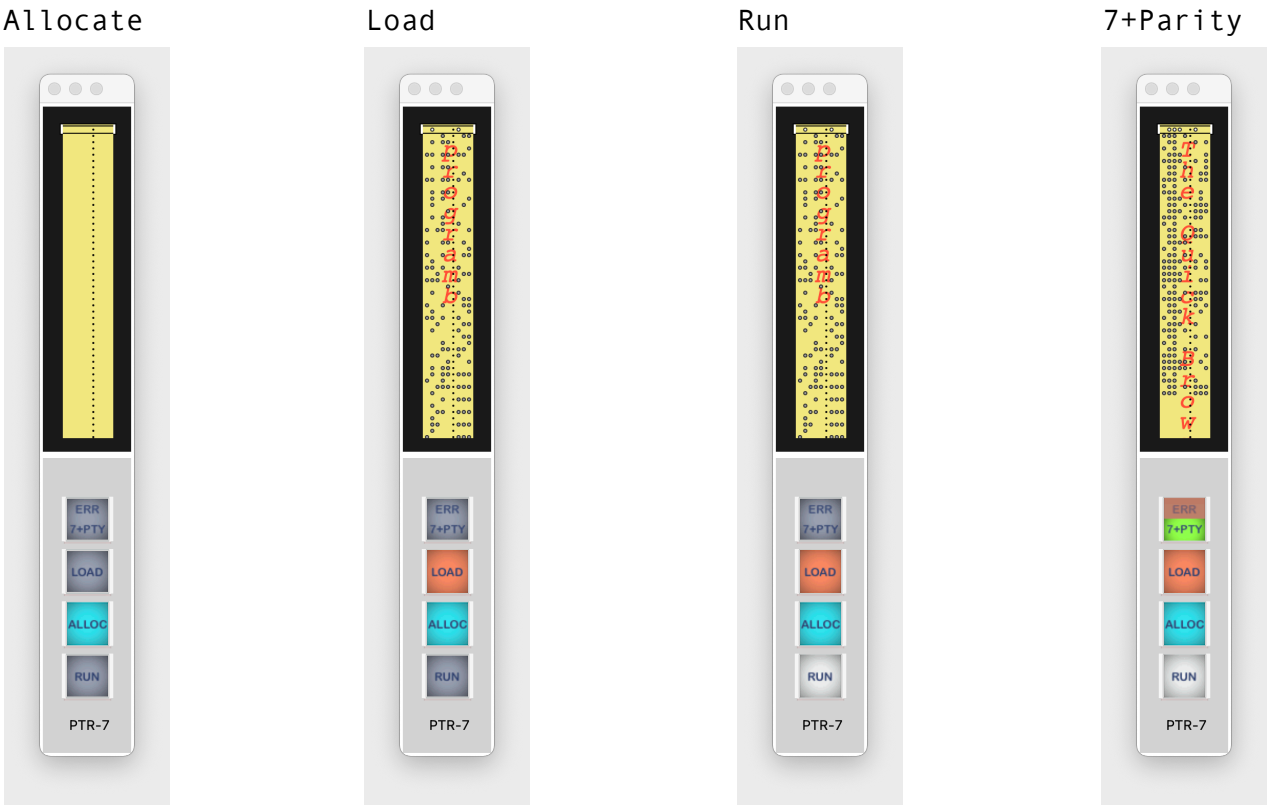
scampa is bundled with Scamp-B whereas plasm2 is a downloadable item from the PlasMa project. Scampa only assembles TOS-B mnemonics and is still playing catchup with plasm2 which can assemble both Toy-A and Toy-B, and offers better error detection and resolution.

NB. All filenames are derived from the source (.pls) filename.

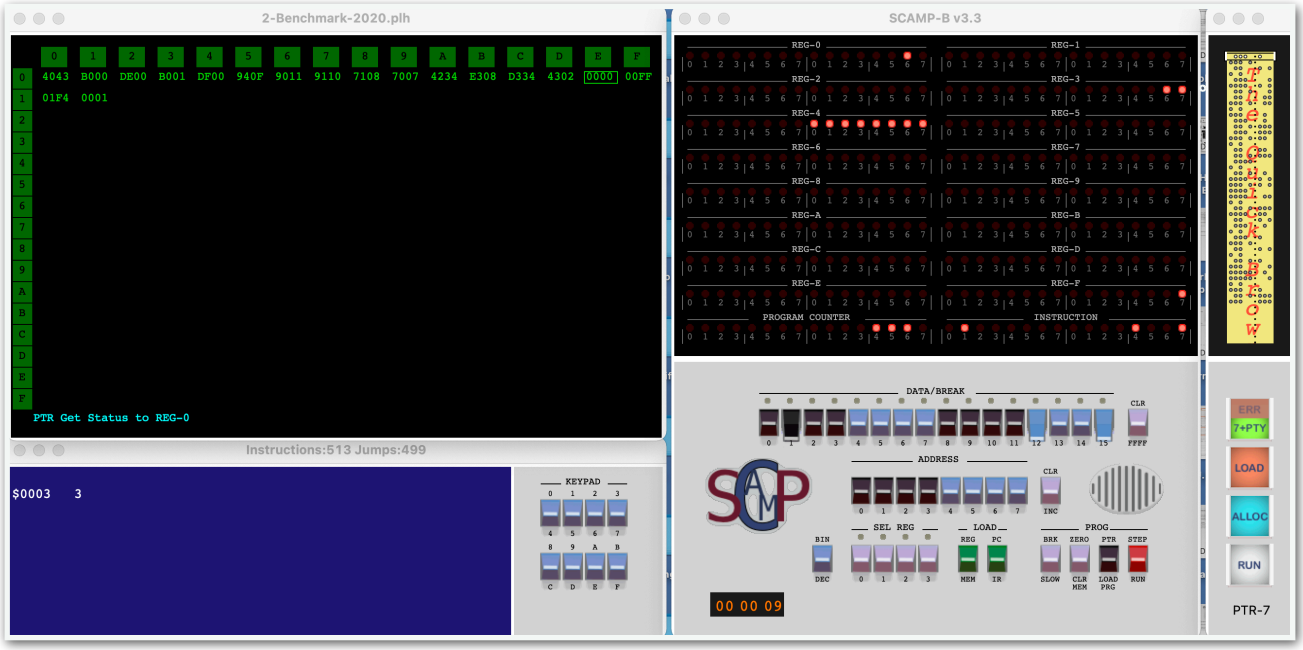
Paper Tape Reader

Bundled with Scamp-B is a paper tape reader program. This runs alongside Scamp-B as a peripheral device and communicates asynchronously using TOS-B system PTR instructions, or the LOAD PTR key on the main panel.

The graphic below shows the device in its various stages of operation.



When the paper tape program is running with Scamp-B it attaches to the side as shown below.



PTR Device Operation

Using TOS-B system instructions a paper tape can be read as follows:

Call PTR Get Status and wait for PTR device to respond with status 1 RDY

This call will light the ALLOC lamp on PTR device

Call PTR Request Data

This will wait for a tape to be loaded on PTR device and for RUN to be selected

Call PTR Get Status

check for status 3 EOT - tape read to end - finish

PTR device will send status 3 EOT when there is no more data

check for status 4 DATA_RDY

PTR device will send 4 DATA_RDY when it has a byte in its buffer

Call PTR Get Data

This will send a byte of data from PTR device to Scamp-B

Call PTR Get Status

check for status 3 EOT - tape read to end - finish

PTR device will send status 3 EOT when there is no more data

check for status 4 DATA_RDY

PTR device will send 4 DATA_RDY when it has a byte in its buffer

Call PTR Get Data

This will send a byte of data from PTR device to Scamp-B

……ETC

To load a tape directly without using a TOS-B program, press the LOAD key on PTR device and select a file to load. When the tape is loaded press RUN on the PTR device. Then switch the LOAD PTR key up on Scamp-B's main panel. On some platforms this may appear to hang for a moment, but eventually the tape data will appear in Scamp-B's memory.

If an address is set on the ADDRESS keys the tape will be loaded starting at that address in memory.

If the 7+Parity button is pressed the tape will light the parity error lamp on detecting a parity error.

The PTR program is sent a close down message when Scamp-B is closed. This invokes a timeout on Windows that causes a closure delay if PTR isn't present.

Appendix

TOS-B Instruction Set

	Function	Action	Assembler Mnemonic
0	Hlt	Halt program	hlt
1	Add	d <– s + t	add rd rs rt
2	Subtract	d <– s - t	sub rd rs rt
3	Multiply	d <– s * t	mul rd rs rt
4	System Call	See Function Table	sys function
50	Jump	PC <– aa	Jmp aa
51	Jump Indexed	PC <– s + t	Jmpi rs rt
60	Jump if greater	If (d > 0) PC <– aa	Jp rd aa (rd < 8)
61	Jump if greater, indexed	If (d > 0) PC <– s + t	Jpi rd rs rt (rd < 8)
70	Jump and decrement	d = d - 1; if (d <> 0) pc <– aa	djnz rd aa (rd < 8)
71	Jump and decrement, indexed	d = d - 1; if (d <> 0) pc <– s + t	djnzi rd rs rt (rd < 8)
80	jump and link	d <– pc + 1; pc <– aa	Jlk rd aa (rd < 8)
81	Jump and link, indexed	d <– pc + 1; pc <– s + t	Jlki rd rs rt (rd < 8)
90	Load	d <– mem[aa]	ld rd aa (rd < 8)
91	Load, indexed	d <– mem[s + t]	ldi rd rs rt (rd < 8)
A0	Store	mem[aa] <– d	st aa rd (rd < 8)
A1	Store, indexed	mem[s + t] <– d	sti rs rt rd (rd < 8)
B0	Load Address	d <– aa	lda rd aa (rd < 8)
B1	Load Address, indexed	d <– s + t	ldai rd rs rt (rd < 8)

	Function	Action	Assembler Mnemonic
C	XOR	d <– s ^ t	xor rd rs rt
D	AND	d <– s & t	and rd rs rt
E0	Shift Right	d <– d >> aa	shr rd aa (rd < 8)
E1	Shift Right, indexed	d <– d >> [s + t]	shri rd rs rt (rd < 8)
F0	Shift Left	d <– d << aa	shl rd aa (rd < 8)
F1	Shift Left, indexed	d <– d << [s + t]	shli rd rs rt (rd < 8)

In the table above the characters marked in RED denote the destination register d, source register s and source register t. In many instructions the d(estination) register can only be in the range 0 to 7. This is identified in the assembler mnemonic column by (rd < 8).

TOS-B System Functions

Function	Action	Description	Code
Oper Read Key group	d <– Oper Keypad	Halt the program and wait for 4 digits	d_1
Oper Write	d –> Oper display	Display register contents on Oper	d_2

In the table above the characters marked in RED denote instruction variables provided through register d. Where a Code includes a '_' its value is irrelevant but must be something, usually '0'.

The PlasMa project owns the copyright to the system function set that TOS-B has embraced, so the definitions of all other system functions may not be included in this document. They can however be downloaded from the PlasMa project and are contained in the **PlasMaSim2 + Plasm2** package.

Scamp-B's TOS-B system functions don't follow exactly the Toy-B I/O specification shown in 'PlasMa-InstructionSet-Toy-B-115'. Functions affected are as follows:

Stop system Timer
- Pseudo implementation.

Write MT Lights
- Inner and outer rings have linked behaviour.

Read Switches
- Address Keys are selection 4, selections 0-3 return Data/Break Keys.

TTY Write (Oper Write)
- Ascii character values are also displayed.

%Directives

%s<n>	defines the PlasMa microcode (Toy-A = 1, Toy-B = 2, etc) This is not required by scampa but must be set to 2 if plasm2 is used.
%m<n>	defines a memory start address for all subsequent data and code (<n> or <$n> for hexadecimal).
%d	defines a data section whereby nothing that follows will be interpreted as code.
%h	defines a hexadecimal section whereby all subsequent data is assumed hexadecimal and will not be interpreted.
%c	defines a code section where anything that follows will be interpreted.
%t	defines a text section whereby anything that follows will be formatted as zero terminated ascii strings.

#Equates

#id <n>	defines a relationship between a numeric quantity <n> (<n> or <$n> for hexadecimal) and an identifying string ‘id’.

.Labels

.id	defines an identifier ‘id’ for an address in code or data.

;Line Comments

;	defines a line comment which will be ignored by the assembler.

{Block Comments}

{}	defines a range to be ignored by the assembler. Must start and end on blank line.

Expressions

Any numeric data can be modified by an expression using the following operators:

Arithmetic:

+ add, - subtract, * multiply, / divide, % modulo

Logic:

& and, | inclusive OR, ^ exclusive OR, < shift left, > shift right

A comprehensive plasm assembler manual is available for download from the PlasMa project.

Running Instructions

Scamp-B is a bundled one directory python application. This means that the python language support files are included in its working directory. The advantage of this is that it is freestanding and doesn't require the host machine to have a version of python loaded. The disadvantage however is that the working directory is littered with a lot of meaningless but very important files which must not be deleted.
As of Scamp-B34 all TOSB (.pls, .plh, .plm, etc) files reside in a subdirectory called 'programs'.
The file structure is as follows:

scamp-b35	The main distribution directory
/programs	subdirectory for all program files
/sounds	subdirectory for tone wav files
/images	subdirectory containing scamp-b's gifs
scamp-b35	main program application
scampa	assembler (application called by scamp-b)
ptr9	paper tape reader application
makewaves	tone generater application (link)

All other files and directories are for the python environment.

Running the executable's scamp-b35 and ptr9 depend on your platform.
On Windows and RPi clicking on the executables will open the programs.
On PC Linux and macOS they must be invoked from a terminal by typing:

```
./scamp-b35 &
./ptr9 &
```

There is one other application program called 'scampa' which is the inbuilt assembler. This command line program is called by Scamp-B so needn't be run independently.

When Scamp-B first runs it loads and runs a TOSB program called 'programb.plh'. This is a start-up demonstration and can be removed or renamed if not required. This program resides in the top working directory alongside the application and support files.

N.B. Screen Size:

Scamp-B requires a screen area of circa 1350 pixels by 650 pixels. This does vary a little depending on the platform used. On Linux Ubuntu it stretches to 1376 wide, a bit over the size of a 1366 screen, and this was only achieved by overlapping the PTR program a little.

There are other differences with the various platform window managers and not many have been tried. In particular the way windows are iconised and the way file dialogs are handled.

Running Issues

The PC Linux version requires Ubuntu 20.04. If it's run under version 18.04 there is an error: GLIBC_2.29 not found.

MacOS version occasionally falls asleep when loading paper tape from ptr9 via LOAD PTR switch. Moving the mouse wakes it up.

Modification History

Scamp-B v3.5:

1.Oper clear action now on both ‘CLR MEM’ and ‘ZERO’ keys. Oper clear override enabled through use of ‘SLOW’ key. Oper clear resets counters and program timer.

2.STEP key now updates instruction count and jump count in Oper title.

3.Illegal instruction trap added.

scampa v2.1:

1.Fixed: Incorrect interpretation of ‘sti’ instruction.

2.Fixed: Hex address passed to ‘st’ instruction as literal causing assembler error.

ptr-9:

1.Improved behaviour on macOS when using LOAD PTR.