A simple debugger for the Beetle virtual machine version 1.0rc2

Reuben Thomas

17th April 2018

1 Introduction

This is the manual for the debugger for C Beetle [2], which provides access to Beetle's registers, allows the stacks to be displayed, and provides disassembly and single-stepping.

There are two main ways to access the debugger: either start `beetle` with no arguments, or, when passing an object file on the command line, give the `--debug` option, which causes the debugger to be entered on exception (interpreted as a negative value being passed to `HALT`). The debugger can also be used in batch mode, by supplying commands on standard input. Refer to the man page **beetle**(1) or `beetle --help` for more details and options.

2 Initialisation

When the virtual machine is started, an embedded Beetle is created. The registers are initialised as described in [2, section 2.5]; additionally, `'THROW` is set to $0, and `I` is uninitialised. `A` is set to zero: this has the effect that when a `STEP` or `RUN` command is given, a `NEXT` instruction will be performed. Thus, when initialisation is performed by a `LOAD` command (see section 3.5), the Beetle may be started with `RUN` or `STEP` immediately after the `LOAD` command, without the need for `FROM`. The main memory is zeroed.

3 Commands

The debugger is command-driven. All commands and register names may be abbreviated to their first few letters; where ambiguities are resolved with a set order of precedence, aimed at giving the most commonly used commands the shortest minimum abbreviations, and commands take precedence over registers. Alternatively, if you have `beetlei` (which uses `rlwrap`), you can use Tab-completion (press the Tab key to show possible commands and registers starting with the letters you have typed so far). All commands are case-insensitive.

The following registers are renamed for ease of typing:

Register	Name in debugger
`'THROW`	`THROW`
`'BAD`	`BAD`
`-ADDRESS`	`NOT_ADDRESS`

If an unrecognised command is given, or the command has too few arguments, or they are badly formed, an error message is displayed. Command lines containing extraneous characters after a valid command are generally accepted, and the extra characters ignored.

Numbers are all integral, and may be given in either decimal or hexadecimal (which must be followed directly by "h" or "H"), with an optional minus sign.

For some arguments a machine instruction may also be used, preceded by "`0`", for opcode. The value of a machine instruction is its opcode. Opcodes are byte-wide values; when used as a cell, the most significant three bytes are set to zero.

The syntax of the commands is shown below; literal text such as command names and other characters are shown in `Typewriter font`; meta-parameters such as numbers are shown in angle brackets, thus: $\langle number \rangle$. Square brackets enclose optional tokens.

There are three types of numeric meta-parameter: $\langle number \rangle$, which is any number; $\langle address \rangle$, which is a valid address (see [1, section 2.6]); and $\langle value \rangle$, which is a number or an opcode.

3.1 Registers

Beetle's registers may be displayed by typing their name. The registers may also (where appropriate) be assigned to using the syntax

$$\langle register \rangle = \langle value \rangle$$

where $\langle value \rangle$ is in the form given in section 3. An error message is displayed if an attempt is made to assign to a register such as `CHECKED`, which cannot be assigned to, or to assign an unaligned or out of range address to a register which must hold an aligned address, such as `SP`. The `FROM` command (see section 3.4) should be used in preference to assigning to `EP`.

Two additional pseudo-registers are provided by the debugger: they are called `S0` and `R0`, and are the address of the base of the data and return stacks respectively. They are set to the initial values of `RP` and `SP`, and are provided so that they can be changed if the stacks are moved, so the stack display commands will still work correctly.

The command `REGISTERS` displays the contents of `EP`, `I` and `A`, useful when following the execution of a program.

3.2 Stacks

The stacks may be manipulated crudely using the registers `SP` and `RP` but it is usually more convenient to use the commands

`TOD` $\langle number \rangle$
`DFROM`

which respectively push a number on to the data stack and pop one, displaying it, and

`TOR` $\langle number \rangle$
`RFROM`

which do the same for the return stack.

The command `DATA` displays the contents of the data stack, and `RETURN` the contents of the return stack. `STACKS` displays both stacks.

If a stack underflows, or the base pointer or top of stack pointer is out of range or unaligned, an appropriate error message is displayed.

3.3 Memory

The contents of an address may be displayed by giving the address as a command. If the address is cell-aligned the whole cell is displayed, otherwise the byte at that address is shown.

A larger section of memory may be displayed with the command `DUMP`, which may be used in the two forms

`DUMP` $\langle address \rangle$ + $\langle number \rangle$
`DUMP` $\langle address_1 \rangle$ $\langle address_2 \rangle$

where the first displays $\langle number \rangle$ bytes starting at address $\langle address \rangle$, and the second displays memory from address $\langle address_1 \rangle$ up to, but not including, address $\langle address_2 \rangle$. An error message is displayed if the start address is less than or equal to the end address or if either address is out of range.

A command of the form

$$\langle address \rangle = \langle value \rangle$$

assigns the value $\langle value \rangle$ to the address $\langle address \rangle$. If the address is not cell-aligned, the value must fit in a byte, and only that byte is assigned to. When assigning to an aligned memory location, a whole cell is assigned unless the number given fits in a byte, and is given using the minimum number of digits required. This should be noted the other way around: to assign a byte-sized significand to a cell, it should be padded with a leading zero.

3.4 Execution

The command `INITIALISE` initialises Beetle as in section 2.

The command `STEP` may be used to single-step through a program. It has three forms:

`STEP` [$\langle number \rangle$]
`STEP TO` $\langle address \rangle$

With no argument, `STEP` executes one instruction. Given a number, `STEP` executes $\langle number \rangle$ instructions. `STEP TO` executes instructions until `EP` is equal to $\langle address \rangle$.

The command `TRACE`, has the same syntax as `STEP`, and performs the same function; in addition, it performs the action of the `REGISTERS` command after each bForth instruction is executed.

The command `RUN` allows Beetle to execute until it reaches a `HALT` instruction, if ever. The code passed to `HALT` is then displayed. The code is also displayed if a `HALT` instruction is ever executed during a `STEP` command.

The command `FROM` sets the point of execution:

`FROM` [$\langle address \rangle$]

With no argument, `FROM` performs the function of Beetle's `NEXT` instruction, that is, it loads `A` from the cell pointed to by `EP`, and adds four to `EP`. With an argument, `FROM` sets `EP` to $\langle address \rangle$, and then performs the function of `NEXT`. `FROM` should be used in preference to assigning directly to `EP`.

The command `DISASSEMBLE` disassembles bForth code. It may be used in the two forms

`DISASSEMBLE` $\langle address \rangle$ + $\langle number \rangle$
`DISASSEMBLE` $\langle address_1 \rangle$ $\langle address_2 \rangle$

where the first disassembles $\langle number \rangle$ bytes starting at address $\langle address \rangle$, and the second from address $\langle address_1 \rangle$ up to, but not including, address $\langle address_2 \rangle$. The addresses must be cell-aligned, and the number of bytes must be a multiple of four. An error message is displayed if the start address is less than or equal to the end address, or if either the address or number of bytes is not aligned or is out of range.

The command `COUNTS` displays the number of times that each Beetle instruction has been executed during `STEP` or `TRACE` execution since the last initialisation (including loads).

3.5 Object modules

The command

$$\texttt{LOAD}\ \langle file\rangle\ [\langle address\rangle]$$

initialises Beetle as in section 2, then loads the object module in file $\langle file\rangle$ into memory at address $\langle address\rangle$ (or address $0 if the argument is omitted). The address must be cell-aligned; if it is not, or if the module would not fit in memory at the address given, or there is some filing error, an error message is displayed.

The command `SAVE` saves an object module. It has the two forms

$$\texttt{SAVE}\ \langle file\rangle\ \langle address\rangle\ +\ \langle number\rangle$$
$$\texttt{SAVE}\ \langle file\rangle\ \langle address_1\rangle\ \langle address_2\rangle$$

where the first saves $\langle number\rangle$ bytes starting at address $\langle address\rangle$, and the second saves from address $\langle address_1\rangle$ up to, but not including, address $\langle address_2\rangle$. The addresses must be cell-aligned, and the number of bytes must be a multiple of four. An error message is displayed if the start address is less than or equal to the end address, or if either the address or number of bytes is not aligned or out of range.

The module is saved to the file $\langle file\rangle$. An error message is displayed if there is some filing error, but no warning is given if a file of that name already exists; it is overwritten.

3.6 Exiting

The command `QUIT` exits Beetle. No warning is given.

References

[1] Reuben Thomas. The Beetle Forth virtual machine, 2018. `https://rrt.sc3d.org/`.

[2] Reuben Thomas. An implementation of the Beetle virtual machine for POSIX, 2018. `https://rrt.sc3d.org/`.