A simple shell for the SMite virtual machine version 0.1

Reuben Thomas

12th January 2019

1 Introduction

This is the manual for the shell for C SMite [1], which provides access to SMite's registers, allows the stacks to be displayed, and provides assembly, disassembly and single-stepping.

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

2 Initialisation

When the virtual machine is started, an embedded SMite is created. The registers are initialised as described in [1, section 2.5Using the interface callssubsection.2.5]; additionally, `HANDLER` is set to 0, and `I` is uninitialised. The main memory is zeroed.

3 Commands

The shell 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 `smitei` (which uses `rlwrap`), you can use Tab-completion (press the Tab key to show possible commands and instructions starting with the letters you have typed so far). All commands are case-insensitive.

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 preceded by "0x"), with an optional minus sign. The following constants are also available:

For some arguments an action may also be used, preceded by "0", for opcode. The value of an action is its opcode.

Name	Value
WORD_SIZE	the number of bytes in a word

Table 1: Constants

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 [2, section 2.6Exceptionssubsection.2.6]); and $\langle value \rangle$, which is a number or an action opcode.

3.1 Registers

SMite'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 `ENDISM`, which cannot be assigned to, or to assign an out of range address to a register such as `PC`.

The command `REGISTERS` displays the contents of `PC` and `I`, 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 Code and data

$\langle opcode \rangle$ NUMBER $\langle number \rangle$ BYTE $\langle number \rangle$ POINTER $\langle number \rangle$

Code and literal data values may be directly assembled into memory. Assembly starts at the last value explicitly assigned to `PC` in the shell, defaulting to 0 whenever SMite is initialised (see section 2). A byte literal (`BYTE`) takes one byte (a one-byte instruction may be used), and a word literal (`NUMBER`) as many bytes as required. `POINTER` assembles the literals required to push the given native pointer on to the stack in the form required for the `CALL_NATIVE` instruction.

3.4 Memory

The contents of an address may be displayed by giving the address as a command. If the address is word-aligned the whole word 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

$$\text{DUMP } [\langle address \rangle \ [\texttt{+} \ \langle number \rangle]]$$
$$\text{DUMP } \langle address_1 \rangle \ [\langle address_2 \rangle]$$

where the first displays $\langle number \rangle$ bytes (or 256 if the number is omitted) starting at address $\langle address \rangle$ (or 64 bytes before PC if the address is omitted, or 0 if that would be negative), 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 word-aligned, the value must fit in a byte, and only that byte is assigned to. When assigning to an aligned memory location, a whole word 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 word, it should be padded with a leading zero.

3.5 Execution

The command INITIALISE initialises SMite as in section 2.

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

$$\text{STEP } [\langle number \rangle]$$
$$\text{STEP TO } \langle address \rangle$$

With no argument, STEP executes one instruction. Given a number, STEP executes $\langle number \rangle$ instructions. STEP TO executes instructions until PC 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 and STACKS commands after each instruction is executed.

The command RUN allows SMite 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 DISASSEMBLE disassembles smite code. It may be used in the two forms

$$\text{DISASSEMBLE } [\langle address \rangle \ [\texttt{+} \ \langle number \rangle]]$$
$$\text{DISASSEMBLE } \langle address_1 \rangle \ \langle address_2 \rangle$$

where the first disassembles $\langle number \rangle$ bytes (or 64 if the number is omitted) starting at address $\langle address \rangle$ (or 16 bytes before PC, or 0 if that would be negative, if the address is omitted), and the second 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 the address or number of bytes is out of range.

3.6 Object modules

The command

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

initialises SMite 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). If the module would not fit in memory at the address given, or there is an I/O 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 $\langle address_2 \rangle$ (defaulting to 0 and the address at which the next instruction would be assembled). 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 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.

References

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

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