The SSM Instruction Set

- Basic instructions are colour-coded green. Additional instructions that your compilers are likely to use (eventually) are colour coded blue.
- The notation for describing the effect of an instruction on the opstack is borrowed from the JVM specification. Example: **sub: …, *x*, *y* → …, *x*-y**
  Items on the stack (4-byte words) are listed with the uppermost item to the right. An ellipsis (…) denotes the lower part of the opstack. So, in this example, ***y*** is on top with ***x*** below ***y***. Both items are popped and the result (***x-y***) is pushed. Any items which happen to be on the opstack below ***x*** and ***y*** are left unchanged.
- **mem[*a*]** denotes the single byte of memory at address ***a***, while **mem4[*a*]** denotes the four-byte word which starts at memory address ***a***.
- **u16(*x*)** denotes the unsigned 2-byte integer obtained by discarding the two high-order bytes from the 4-byte word ***x***.
- **byte(*x*)** denotes the single byte obtained by discarding the three high-order bytes from the 4-byte word ***x***.
- **pad(*b*)** denotes the 4-byte word obtained by high-padding the single byte ***b*** with three zero bytes.

SSM Instructions (1-byte)

Opcode	Mnemonic	Description
0 (0x00)	**noop**	No effect.
1 (0x01)	**halt**	Halt the machine. The opstack should normally be empty (normal termination); otherwise ***n*** is popped from the opstack and **u16(n)** is interpreted as an error code for abnormal termination (configurable[1]).
2 (0x02)	**pop**	Discard the top item from the opstack: **…, *x* → …**
3 (0x03)	**dup**	Push a duplicate of the top item onto the opstack: **…, *x* → …, *x*, *x***
4 (0x04)	**swap**	Swap the top two items on the opstack: **…, *x*, *y* → …, *y*, *x***
5 (0x05)	**rot**	"Rotate" the top three items on the opstack (pull third item to the top): **…, *x*, *y*, *z* → …, *y*, *z*, *x***
6 (0x06)	**add**	Twos-complement integer addition: **…, *x*, *y* → …, *x+y***
7 (0x07)	**sub**	Twos-complement integer subtraction: **…, *x*, y → …, *x-y***
8 (0x08)	**mul**	Twos-complement integer multiplication: **…, *x*, *y* → …, *x*y***
9 (0x09)	**div**	Twos-complement integer division: **…, *x*, *y* → …, *x/y***
10 (0x0a)	**test_z**	If top item is zero, replace by 1, otherwise replace by 0: **…, 0 → …, 1** if ***x*** = 0 or **…, *x* → …, 0** if ***x*** ≠ 0
11 (0x0b)	**test_n**	If top item is negative, replace by 1, otherwise replace by 0: **…, *x* → …, 1** if ***x*** < 0 or **…, *x* → …, 0** if ***x*** ≥ 0

[1] Defaults: 0 = normal termination, 1 = "Null Pointer", 2 = "Array Index Out of Range", 3 = "Heap Exhausted".

SSM Instructions (1-byte) continued

Opcode	Mnemonic	Description
12 (0x0c)	**get_dp**	Push the value of the DP register onto the opstack: **…** → **…, DP**
13 (0x0d)	**get_fp**	Push the value of the FP register onto the opstack: **…** → **…, FP**
14 (0x0e)	**get_sp**	Push the value of the SP register onto the opstack: **…** → **…, SP**
15 (0x0f)	**load**	Load a 4-byte word from memory: **…, *x*** → **…, mem4[u16(*x*)]**
16 (0x10)	**loadb**	Load a single byte from memory: **…, *x*** → **…, pad(mem[u16(*x*)])**
17 (0x11)	**store**	Store a 4-byte word in memory: **…, *x, y*** → **…** with effect: **mem4[u16(*x*)] := *y***
18 (0x12)	**storeb**	Store a single byte in memory: **…, *x, y*** → **…** with effect: **mem[u16(*x*)] := byte(*y*)**
19 (0x13)	**jump**	Unconditional jump: **…, *x***→ **…** with effect: control jumps to the instruction at address **u16(*x*)**
20 (0x14)	**jump_z**	Conditional jump: **…, *x, y*** → **…** with effect: control jumps to the instruction at address **u16(*y*)** if ***x*** = 0
21 (0x15)	**jump_n**	Conditional jump: **…, *x, y*** → **…** with effect: control jumps to the instruction at address **u16(*y*)** if ***x*** < 0
22 (0x16)	**call**	Function call[2]: **…, *f*, x_1, …, x_n, *n*** → **…, *v*** with effect: call the function at address **u16(*f*)** passing ***n*** four-byte arguments on the call-stack; push the function's return value ***v*** on the opstack on return
23 (0x17)	**ret**	Return from call[3]: **…, v, *n*** → with effect: pop ***n***+1 four-byte words off the call-stack, restore the previous opstack, push ***v*** on the restored opstack, jump to the current call's return address

[2] The function code executes in the context of a new, initially empty, opstack. The old opstack is restored when the call returns.

[3] ***n*** should match the total number of four-byte words (parameters plus locals) allocated for the current call; ***n***+1 words will be popped in total since one additional word is always used for stack management (return address and saved FP).

SSM Instructions (2-byte)

Opcode	Mnemonic	Data (1 byte)	Description
24 (0x18)	**pushb**	***b***	Push a single byte on the opstack: **…** → **…, pad(*b*)**
25 (0x19)	**sysc**	***n***	Execute system call number ***n***.

System Calls

Syscall number	Literal	Arity[4]	Description
0 (0x00)	**OUT_BYTE**	1	Pop ***n*** from the opstack. Write the low-order byte of ***n*** on the standard output stream.
1 (0x01)	**OUT_CHAR**	1	Pop ***n*** from the opstack. Print the ASCII character specified by the low-order byte of ***n*** to the standard output stream.
2 (0x02)	**OUT_LN**	0	Print a line-ending to the standard output stream.
3 (0x03)	**OUT_DEC**	1	Pop twos-complement integer ***n*** from the opstack. Print the decimal representation of ***n*** to the standard output stream.
4 (0x04)	**OUT_STR**	1	Pop ***x*** from the opstack. Print the string stored at address **u16(*x*)** to the standard output. The following memory layout is assumed for string data: the first two bytes provide the string length (an unsigned integer ***n*** in the range 0 ≤ ***n*** < 65536); the following ***n*** bytes contain the ASCII codes of the letters in the string.
5 (0x05)	**READ_BYTE**	0	Read a single byte from the standard input stream, high-pad with three zero bytes, and push on the opstack.
6 (0x06)	**READ_INT**	0	Attempt to read a valid decimal integer string from the standard input stream. If successful (ie a valid string was entered and parsed as integer ***n***) push ***n*** followed by a 1: **…** → **…, *n*, 1**. If an invalid string was entered, just push 0: **…** → **…, 0**.
7 (0x07)	**PUSH_ARGC**	0	Push ***n*** on the opstack where ***n*** is the number of command-line arguments that were provided to the SSM on start-up.
8 (0x08)	**PUSH_ARG**	1	Pop ***n*** from the opstack. Use ***n*** as an index into the command-line parameter array and push ***a*** on the opstack, where ***a*** is the memory address of the corresponding string data. The machine will halt with an error if ***n*** is out of range.
9(0x09)	**MALLOC**	1	Pop ***n*** from the opstack. Allocate a contiguous block of ***n*** bytes of memory in the heap and push ***a*** on the opstack, where ***a*** is the memory address of the start of the allocated block; push **0** if allocation fails (heap exhausted).
10(0x0a)	**CALLOC**	1	The same as MALLOC but the allocated memory is filled with zeroes.
11(0x0b)	**FREE**	1	Pop ***x*** from the opstack. Deallocate the previously allocated block of memory starting at address **u16(*x*)**. It is an error if **u16(*x*)** is not an address previously returned by MALLOC or CALLOC, or if already deallocated.

[4] The "arity" of a system call is the number of four-byte words that it consumes from the opstack.

SSM Instructions (4-byte)

Opcode	Mnemonic	Data (2 bytes)	Description
26 (0x1a)	**loadi**	***a***	Load a four-byte word from memory: **…** → **…, mem4[*a*]**
27 (0x1b)	**loadbi**	***a***	Load a single byte from memory: **…** → **…, pad(mem[*a*])**
28 (0x1c)	**storei**	***a***	Store a four-byte word in memory: **…, *x*** → **…** with effect: **mem4[*a*] := *x***
29 (0x1d)	**storebi**	***a***	Store a single byte in memory: **…, *x*** → **…** with effect: **mem[*a*] := byte(*x*)**
30 (0x1e)	**jumpi**	***a***	Unconditional jump: control jumps to the instruction at address ***a*** (opstack is unchanged)
31 (0x1f)	**jumpi_z**	***a***	Conditional jump: **…, *x*** → **…** with effect: control jumps to the instruction at address ***a*** if ***x*** = 0
32 (0x20)	**jumpi_n**	***a***	Conditional jump: **…, *x*** → **…** with effect: control jumps to the instruction at address ***a*** if ***x*** < 0
33 (0x21)	**calli**	***a***	Function call[5]: **…, x_1, …, x_n, *n*** → **…, *v*** with effect: call the function at address ***a*** passing ***n*** four-byte arguments on the call-stack; push the function's return value ***v*** on the opstack on return
34 (0x22)	**salloc**	***n***	Allocate space on the call-stack: **…, *n*** → **…** with effect: allocate ***n*** four-byte words of space on the call-stack (decrement SP by 4****n***)
35 (0x23)	**sfree**	***n***	De-allocate space from the call-stack: **…, *n*** → **…** with effect: de-allocate ***n*** four-byte words of space from the call-stack (increment SP by 4****n***)

SSM Instructions (5-byte)

Opcode	Mnemonic	Data (4 bytes)	Description
36 (0x24)	**push**	***x***	Push a four-byte word on the opstack: **…** → **…, *x***

[5] Note that the top value on the opstack (***n***) determines how many additional items (the function arguments) will be consumed from the opstack and pushed onto the call-stack. The function code executes in the context of a separate opstack of its own (initially empty).