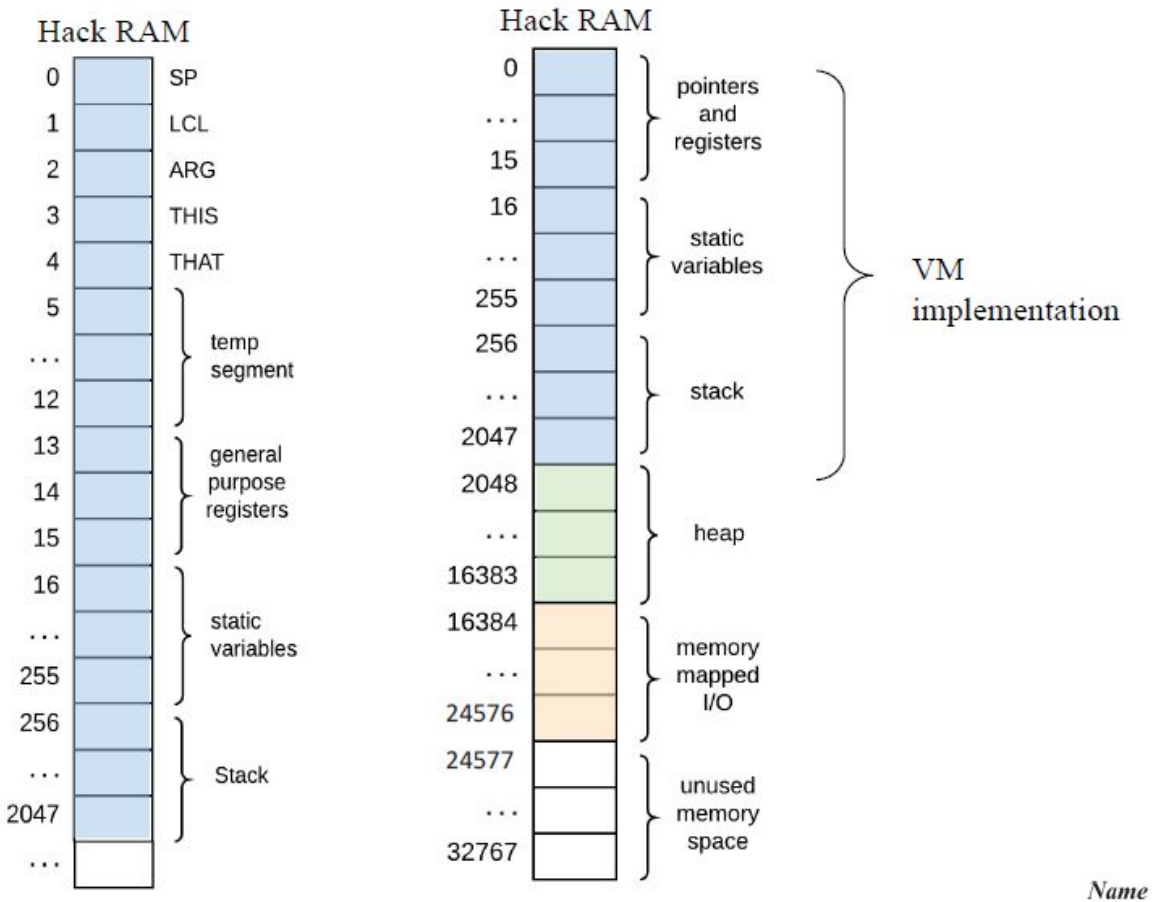
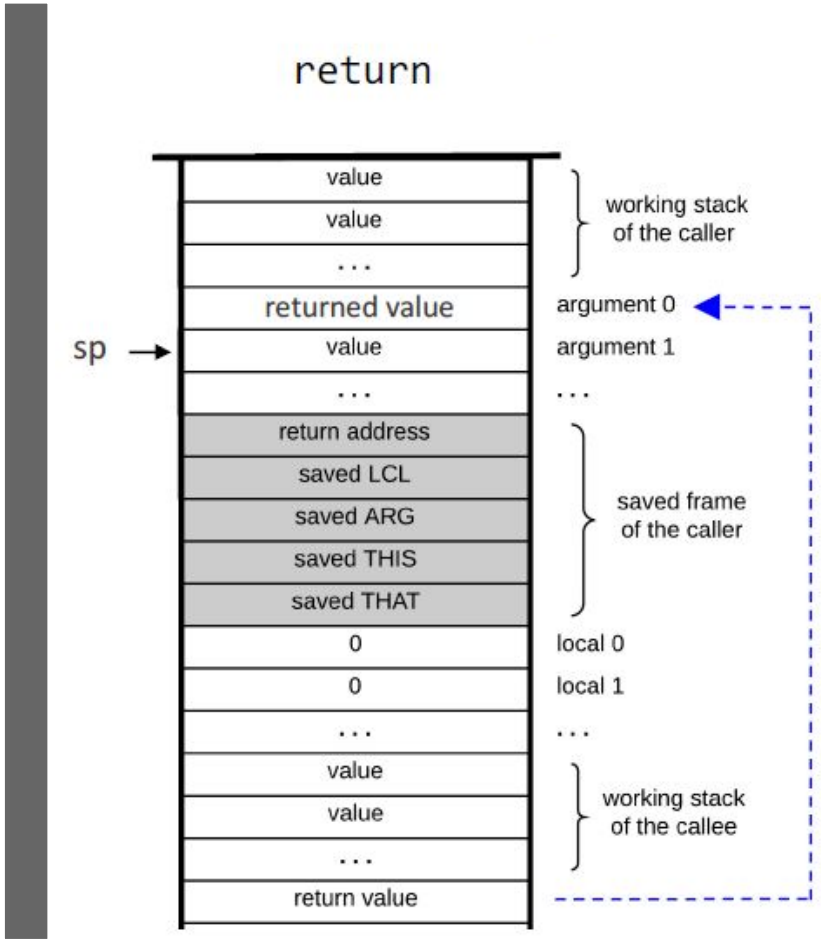
Standard mapping of the VM on the Hack platform





Name	*Location*	*Usage*
SP	RAM[0]	*Stack Pointer*: the memory address just following the memory address containing the topmost stack value
LCL	RAM[1]	Base address of the `local` segment
ARG	RAM[2]	Base address of the `argument` segment
THIS	RAM[3]	Base address of the `this` segment
THAT	RAM[4]	Base address of the `that` segment
TEMP	RAM[5-12]	Holds the `temp` segment
R13 R14 R15	RAM[13-15]	If the assembly code generated by the VM translator needs variables, it can use these registers.

VM command	*Assembly (pseudo) code, generated by the VM translator*	
`call` *f nArgs* (calls a function *f*, informing that *nArgs* arguments were pushed to the stack before the call)	`push` *returnAddress*	// generates a label and pushes it to the stack
	`push LCL`	// saves `LCL` of the caller
	`push ARG`	// saves `ARG` of the caller
	`push THIS`	// saves `THIS` of the caller
	`push THAT`	// saves `THAT` of the caller
	`ARG = SP-5-`*nArgs*	// repositions `ARG`
	`LCL = SP`	// repositions `LCL`
	`goto` *f*	// transfers control to the callee
	(*returnAddress*)	// injects the return address label into the code
`function` *f nVars* (declares a function *f*, informing that the function has *nVars* local variables)	(*f*)	// injects a function entry label into the code
	repeat *nVars* times:	// *nVars* = number of local variables
	`push 0`	// initializes the local variables to 0
`return` (terminates the current function and returns control to the caller)	*frame* = `LCL`	// *frame* is a temporary variable
	retAddr = *(*frame*-5)	// puts the return address in a temporary variable
	`*ARG = pop()`	// repositions the return value for the caller
	`SP = ARG+1`	// repositions `SP` for the caller
	`THAT` = *(*frame*-1)	// restores `THAT` for the caller
	`THIS` = *(*frame*-2)	// restores `THIS` for the caller
	`ARG` = *(*frame*-3)	// restores `ARG` for the caller
	`LCL` = *(*frame*-4)	// restores `LCL` for the caller
	`goto` *retAddr*	// go to the return address

Figure 8.5 Implementation of the function commands of the VM language. All the actions described on the right are realized by generated Hack assembly instructions.

Special symbols in translated VM programs

Symbol	***Usage***
`SP`	This predefined symbol points to the memory address within the host RAM just following the address containing the topmost stack value.
`LCL`, `ARG`, `THIS`, `THAT`	These predefined symbols point, respectively, to the base addresses within the host RAM of the virtual segments `local`, `argument`, `this`, and `that` of the currently running VM function.
`R13`–`R15`	These predefined symbols can be used for any purpose.
`Xxx.`*i* symbols	Each static variable *i* in file `Xxx.vm` is translated into the assembly symbol `Xxx.`*j*., where *j* is incremented each time a new static variable is encountered in the file `Xxx.vm`. In the subsequent assembly process, these symbolic variables will be allocated to the RAM by the Hack assembler.
functionName `$`*label*	Let `foo` be a function within a VM file `Xxx`. Each `label bar` command within `foo` should generate and insert into the assembly code stream a symbol `Xxx.foo$bar`. When translating `goto bar` and `if-goto bar` commands (within `foo`) into assembly, the full label specification `Xxx.foo$bar` must be used instead of `bar`.
functionName	Each `function foo` command within a VM file `Xxx` should generate and insert into the assembly code stream a symbol `Xxx.foo` that labels the entry point to the function's code. In the subsequent assembly process, the assembler will translate this symbol into the physical memory address where the function code starts.
functionName `$ret.`*i*	Let `foo` be a function within a VM file `Xxx`. Within `foo`, each function `call` command should generate and insert into the assembly code stream a symbol `Xxx.foo$ret.`*i*, where *i* is a running integer (one such symbol should be generated for each `call` command within `foo`). This symbol serves as the return address to the calling function. In the subsequent assembly process, the assembler will translate this symbol into the physical memory address of the command immediately after the function call command.