Expand All | Collapse All

Task 4.1 - Push & Pop (6 points)

Complete the `vm_push` & `vm_pop` methods.

These methods should return Hack Assembly code that do the following:

`vm_push`

- Read the value from the correct memory segment, then push that value to the stack.
- Constant values need to be emulated.

`vm_pop`

- Pop a value from the stack, then write that value to the correct memory segment.

Test Cases:

- Write at least 2 test cases per method.
- Each test case should be in a file named `METHODTestXX.vm` where `METHOD` is the name of the method and `XX` is a number starting at `01`.
- See the section *Writing Tests* below for details on how to write test cases.
- Your mark for this task may be **scaled down as much as 50%** for poor/missing testing.

Task 4.2 - Arithmetic Operations (up to 2 points)

Complete any 1 of the following methods:

These methods should return Hack Assembly code that do the following:

`vm_add`

- Pop 2 values from the stack, **add** them, then push then result back to the stack.

`vm_sub`

- Pop 2 values from the stack, **subtract** them, then push then result back to the stack.

`vm_neg`

- Pop 1 value from the stack, **negate** it (i.e. flip its sign), then push the result back to the stack.

Test Cases:

- Write at least 1 test case per method.
- Each test case should be in a file named `METHODTestXX.vm` where `METHOD` is the name of the method and `XX` is a number starting at `01`.
- See the section *Writing Tests* below for details on how to write test cases.
- Your mark for this task may be **scaled down as much as 50%** for poor/missing testing.

Task 4.3 - Logic Operations (up to 4 points)

Complete any 2 of the following methods:

These methods should return Hack Assembly code that do the following:

`vm_eq`

- Pop 2 values from the stack, and compare them, then push the result back to the stack.
  - If they are **equal**, then push TRUE (-1) back to the stack, otherwise push FALSE (0)

`vm_gt`

- Pop 2 values from the stack, and compare them, then push the result back to the stack.
  - Compare the second value from the top of the stack to the value at the top of the stack (See chapter 7.3 in the Text book)
  - If the second value is **greater** than the top value, then push TRUE (-1) back to the stack, otherwise push FALSE (0)

`vm_lt`

- Pop 2 values from the stack, and compare them, then push the result back to the stack.
  - Compare the second value from the top of the stack to the value at the top of the stack (See chapter 7.3 in the Text book)
  - If the second value is **less** than the top value, then push TRUE (-1) back to the stack, otherwise push FALSE (0)

`vm_and`

- Pop 2 values from the stack, perform a bit-wise **and** on them, then push the result back to the stack.

`vm_or`

- Pop 2 values from the stack, perform a bit-wise **or** on them, then push the result back to the stack.

`vm_not`

- Pop 1 value from the stack, perform a bit-wise **not/invert** on it, then push the result back to the stack.

Test Cases:

- Write at least 1 test case per method.
- Each test case should be in a file named `METHODTestXX.vm` where `METHOD` is the name of the method and `XX` is a number starting at `01`.
- See the section *Writing Tests* below for details on how to write test cases.
- Your mark for this task may be **scaled down as much as 50%** for poor/missing testing.

Task 4.4 - Jump Operations (8 points)

Complete the `vm_label`, `vm_goto` & `vm_if` methods.

These methods should return Hack Assembly code that do the following:

`vm_label`

- Creates a label that can be used with jump instructions.

`vm_goto`

- Performs an unconditional jump to the location marked by the provided label.

`vm_if`

- Pop a value from the stack. If that value is **not FALSE** (not 0), jump to the location marked by the provided label.

Test Cases:

- Write at least 2 test cases per method.
- Each test case should be in a file named `METHODTestXX.vm` where `METHOD` is the name of the method and `XX` is a number starting at `01`.
- See the section *Writing Tests* below for details on how to write test cases.
- Your mark for this task may be **scaled down as much as 50%** for poor/missing testing.

Task 4.5 - Function Operations (12 points)

Complete the `vm_function`, `vm_call` & `vm_return` methods.

These methods should return Hack Assembly code that do the following:

`vm_function`

- Marks the beginning of a function with a given name and a number of local variables.
- This includes:
  - Generating a label for the program to jump to when the function is called.
  - Initialising the local variables to 0 by pushing the correct number of 0s to the stack.

`vm_call`

- Calls a function with a given name and a number arguments.
- This includes:
  - Generating a label for the program to return to when the function is returns.
  - Saving the stack frame.
  - Updating the memory segment pointers to their new locations.
  - Jumping to the label for the function.

`vm_return`

- Returns from the current function.
- This includes:
  - Copying the return value to the correct location on the stack.
  - Restoring the memory segment pointers with the values from the stack frame.
  - Jumping to the return label (which is stored in the stack frame).

Test Cases:

- Write at least 2 test cases per method.
- Each test case should be in a file named `METHODTestXX.vm` where `METHOD` is the name of the method and `XX` is a number starting at `01`.
- See the section *Writing Tests* below for details on how to write test cases.
- Your mark for this task may be **scaled down as much as 50%** for poor/missing testing.