Hints for TinyJ Assignment 3

Before you work on the assignment, carefully read the first page of the assignment document. When writing the `execute()` method for an instruction, if you are not clear as to what that instruction should do when it is executed, refer back to the “Effects of Executing Each TinyJ Virtual Machine Instruction” pages you received earlier.

Remarks on the `execute()` Methods You Have to Write

1. Instructions That Do Not Use Pointers to Data Memory Locations and Have No Operands

   **ADD, SUB, DIV, MUL, MOD, AND, OR**

   As explained on p. 2 of the assignment document, **ADD**'s `execute()` method can be implemented as follows:
   \[ \text{EXPRSTACK}[-1] += \text{EXPRSTACK}[2]; \]

   The `execute()` methods for **SUB, DIV, MUL, MOD, AND, and OR** can be written analogously.

   Use 1 and 0 to represent `true` and `false`; then the `execute()` methods for **AND and OR** can use `&=` and `|=`
   where **ADD**'s `execute()` uses `+=`.

   **LE, GE, LT, GT, EQ, NE**

   One simple way to write the `execute()` methods for these instructions is to use conditional expressions.

   For example, **LE**'s `execute()` method could be implemented as follows:
   \[ \text{EXPRSTACK}[-1] = (\text{EXPRSTACK}[2] \leq \text{EXPRSTACK}[3]) \ ? 1 : 0; \]

   **CHANGESIGN, NOT**

   These instructions should change `EXPRSTACK[ESP]` without changing ESP. (Note: `*=` -1 is a concise way to negate an `int` variable; `&=` 1 is a concise way to change its value from 1 to 0 or vice versa.)

   **WRITELNOP, WRITEINT**

   These instructions should print to `System.out`: **WRITELNOP** should print a newline, and **WRITEINT**
   should print an integer that is popped off `EXPRSTACK`. (Note that `EXPRSTACK[ESP]` represents a value that is popped off `EXPRSTACK`.)

2. Instructions That Do Not Use Pointers to Data Memory Locations but Have Operands

   The operand of any **OneOperandInstruction** (such as **PUSHNUM** or **JUMP** instr) is in its **operand**
   field (which it inherits from **OneOperandInstruction**). The two operands of a **WRITESTRING**[str]
   are in its **firstOperand** and **secondOperand** fields (which it inherits from **TwoOperandInstruction**).

   Recall from p. 1 of the assignment document that **PC** represents the VM’s program counter: It stores
   the **code memory** address of the location from which the next instruction will be fetched for execution.

   **PUSHNUM 17** should push its operand onto `EXPRSTACK` (i.e., put 17 into `EXPRSTACK[ESP++]`).

   **JUMP 17** should put its operand 17 into PC, as PC represents the program counter.

   **JUMPONFALSE 17** should look at a value that is popped off `EXPRSTACK` (i.e., look at the value
   `EXPRSTACK[ESP]`) and put its operand 17 into PC just if that popped value is 0.

   **WRITESTRING 7 17** should print (to `System.out`) the characters that are stored in the data memory
   locations with addresses \( \geq \) its **firstOperand** 7 but \( \leq \) its **secondOperand** 17.

   These locations are represented by:
   \[ \text{TJ.data}[7], \text{TJ.data}[8], \ldots, \text{TJ.data}[16], \text{TJ.data}[17] \]

   You must perform a (char) cast on the value in each location before printing it.
3. Instructions That Use Pointers to Data Memory Locations

Before you write the execute() methods for these instructions, you should read the section “Use of CodeInterpreter.POINTERTAG” on p. 4 of the assignment document.

Also recall from p. 1 of the assignment document that FP stores a pointer to the data memory location at offset 0 in the currently executing method activation’s stackframe. (Thus FP + k will be a pointer to the location at offset k in that stackframe.)

**PUSHSTATADDR 7** This should push 7+POINTERTAG (i.e., push a pointer to the data memory location whose address is 7) onto EXPRSTACK; in other words, it should put the pointer 7+POINTERTAG into EXPRSTACK[ESP++].

**PUSHLOCADDR 7** This should push FP + 7 (i.e., push a pointer to the data memory location at offset 7 in the currently executing method activation's stackframe) onto EXPRSTACK; in other words, it should put the pointer FP + 7 into EXPRSTACK[ESP++].

**LOADFROMADDR** If p is the pointer in EXPRSTACK[ESP-1], execution of this instruction should copy the value in the data memory location to which p points (i.e., copy the value TJ.data[p-POINTERTAG]) into EXPRSTACK[ESP-1]; the copied value should overwrite the pointer p.

**SAVETOADDR** If v is the value in EXPRSTACK[ESP-1] and p is the pointer in EXPRSTACK[ESP-2], then this instruction should store v into the data memory location to which p points (i.e., it should store v into TJ.data[p-POINTERTAG]), and should also decrease ESP by 2 to pop v and p off EXPRSTACK.

4. Instructions Associated with Calls of Static Methods and Return from Called Methods

Recall from p. 1 of the assignment document that ASP stores a pointer to the first unused location in the stack-dynamically allocated part of data memory—i.e., ASP points to the first location above the currently allocated locations in that part of data memory.

ASP must therefore be increased / decreased by k when k stackframe locations are allocated / deallocated.

We will write **S-PUSH y** for TJ.data[ASP++ - POINTERTAG] = y;
We will write **S-POP y** for y = TJ.data[--ASP - POINTERTAG];

51: **PASSPARAM** should be **S-PUSH** a value that is popped off EXPRSTACK—i.e., it should be **S-PUSH** EXPRSTACK[--ESP]

52: **CALLSTATMETHOD 671** should be **S-PUSH** PC and then Set PC to 671 [saves return addr (here, 53) into new frame]
and then S-PUSH FP [transfers control to the called method’s code]

671: **INITSTKFRM 7** should be **S-PUSH** FP and then Set FP to ASP-1 [saves caller’s FP at offset 0 in the new frame]
and then **S-PUSH** FP [makes FP point to offset 0 in the new frame]
and then Increase ASP by 7 [allocates space for callee's local variables]

713: **RETURN 4** should be **S-PUSH** FP and then Set ASP to FP+1 [deallocates space used by callee's variables]
and then **S-POP** FP [restores caller’s FP]
and then **S-PUSH** PC [puts the saved return address into PC]
and then Decrease ASP by 4 [deallocates space used by formal parameters]