CS 474: Object Oriented Programming Languages and Environments
Spring 2018

Second Smallalk project (revised)

Due time: **9:00 pm on Wednesday 3/21/2018**

Instructor: Ugo Buy
TA: Parneet Kaur and Siddhesh Chavan

Total points: 120
Full credit: 100 points (20 points extra credit)

This project is about building an *Assembly Language Interpreter (ALI)* for a *Simple Assembly Language (SAL)*. Fortunately for you SAL has a limited set of instructions, described below, which makes it easier to write the ALI. SAL programs are executed on a virtual (emulated) machine consisting of a memory, which stores program code and program data, an *accumulator* register, an additional register and a *Program Counter (PC)*, which keeps track of the instruction being currently executed. Your ALI can execute SAL programs one line at a time (in a sort of debug mode) or at once until a halt instruction is encountered. Also, your ALI will include a graphical user interface that displays various kinds of information:

1. An editor field that shows the state of computer memory in symbolic format (as opposed to binary or hex format).
2. An *action button* for executing one line of code, starting from the first line of memory, numbered Line 0. See also the description of the *Program Counter (PC)* below.
3. An *action button* for executing the entire program, starting from the current value of the PC.
4. A set of widgets for displaying the content of the accumulator, additional register and PC. These widgets are updated each time one of the two action buttons above are selected.

The computer hardware uses 30-bit words and consists of the following components:

1. *Memory.* A 30-bit, word-addressable memory (RAM) for data, holding 256 words. Words are addressed by their location, starting from location 0 all the way up to location 255. Each location may either hold a signed integer in 2's complement notation or a SAL instruction.
2. *Accumulator.* A 30-bit register. It is also known as *Register A* or *A* for short.
3. *Additional register.* A 30-bit register also known as *Register B* or *B* for short.
4. *Program counter (PC).* An 8-bit program counter (PC). The PC holds the address (number in program memory) of the next instruction to be executed. Before the program starts execution, the PC holds the value 0. It is subsequently updated as each instruction is executed.
5. A *zero-result bit.* This bit is set if the last ADD instruction produced a zero result. This bit is cleared if the last ADD instruction produced a result different from zero. The initial value is zero. The bit is changed only after ADD instructions are executed.
6. An *overflow bit.* This bit is set whenever an ADD instruction produces an overflow (i.e., a result that cannot be stored in 2's complement notation with 30 bits). It is cleared if the ADD instruction did not produce an overflow. The initial value is zero.

The registers are used to hold data in arithmetic operations (i.e., additions). The program counter holds the index value (starting at 0) of the next instruction to be executed. SAL has the instruction set shown in Table 1.

DEC *symbol*	Declares a symbolic variable consisting of a single letter (e.g., X). The variable is stored at the memory location of this instruction.
LDA *symbol*	Loads byte at data memory address of *symbol* into the accumulator.
LDB *symbol*	Loads byte at data memory address *symbol* into B.
LDI *value*	Loads the integer *value* into the accumulator register. The value could be negative but must be in the range of 30-bit 2's complement numbers.
ST *symbol*	Stores content of accumulator into data memory at address of *symbol.*
XCH	Exchanges the content registers A and B.
JMP *number*	Transfers control to instruction at address *number* in program memory.
JZS *number*	Transfers control to instruction at address *number* if the zero-result bit is set.
JVS *number*	Transfers control to instruction at address *number* if the overflow bit is set.
ADD	Adds the content of registers A and B. The sum is stored in A. The overflow and zero-result bits are set or cleared as needed.
HLT	Terminates program execution.

Table 1: Instruction set of SAL.

You may assume that SAL programs are entered correctly by users of ALI. (You are not required to perform error diagnosis or correction). Information in data memory and the registers can be displayed in binary, decimal, or hexadecimal format. Program source code should be displayed in symbolic (i.e., textual) format. Finally, you must use inheritance in your implementation. One good place for inheritance is to define an abstract superclass for SAL instructions with concrete subclasses for each particular instruction. Beware of infinite loops in SAL. Note that instances of the Smalltalk class *SmallInteger* use a 30-bit 2's complement notation.

Extra credit. If you implement correctly the *Command* pattern from the Gang-of-Four book, you will receive a 20% increase on your project grade, up to 20 extra points.

You must work alone on this project. Your project code should be in a special package called CS474. Save all your code by filing out that package in the file xxx.st, where xxx denotes your last name. Submit the file by clicking on the link provided with this assignment. No late submissions will be accepted.