The Virtual Machine: Program Flow

Prof. Naga Kandasamy
ECE Department, Drexel University

By default, a computer program proceeds sequentially by executing one instruction after the other. This flow can be overridden by branch instructions. Our stack-based VM specification supports *if-goto* and *goto* commands for conditional and unconditional branches, respectively. The *if-goto* command conditions the jump on the value of the stack's topmost element — it pops the result from the stack and if the result is non-zero, jumps to the target address; else, executes the next command in the program. The *goto* affects an unconditional jump in that it simply jumps to the target address.

Usage of the *if-goto* and *goto* commands are summarized below.

```
The if-goto mechanism:
    1. Pop result of expression from stack.
    2. If result is non-zero, goto (LABEL).

Example:
if (condition)
    S1
else
    S2

VM code:
// Code to evaluate condition and place result on stack
if-goto L1
    // Code to execute S2
    goto L2
Label L1
    // Code to execute S1
Label L2
    // Rest of code

Example:
while (condition)
    S1

VM code:
label L1
    // Code to evaluate not(condition) and place result on stack
    if-goto L2
    // Code to execute S1
    goto L1
label L2
    // Rest of code
```

The following VM program calculates minimum of two variables stored in the first and second locations within the local segment. The result is stored in the local segment's third location.

```
1  // min.vm
2  // c = min(a, b)
3
4  push local 0            // Push a into stack
5  push local 1            // Push b into stack
6  lt                      // is a < b?
7  if-goto A_LESS_THAN_B
8      push local 1
9      pop local 2         // c = b
10     goto END_IF_ELSE
11 label A_LESS_THAN_B
12     push local 0
13     pop local 2         // c = a
14 label END_IF_ELSE
15     // Rest of code
```

Lines 4 and 5 push variables on the working stack of the program. The *lt* command in line 6 pops the two operands, performs the operation, and pushes the result to the stack. (Recall that Boolean True is represented as `FFFF` (or -1_{10}) and False as 0.) The subsequent *if-goto* command pops the result from the stack and if the result is non-zero, jumps to the target address; else, executes the next command in the program. The *goto* command affects an unconditional jump in that it simply jumps to the target address.

The *lt* command can be translated into the following assembly-code statements.

```
1      @SP
2      M = M - 1
3      A = M
4      D = M               // D <-- operand2
5      @SP
6      M = M - 1           // Adjust SP
7      A = M
8      D = M - D           // D <-- operand1 - operand2
9      @IF_LT_6
10     D;JLT               // if operand1 < operand2 goto IF_LT_6
11     @SP
12     A = M
13     M = 0               // Push False on stack
14     @SP                 // Adjust SP
15     M = M + 1
16     @END_IF_ELSE_6
17     0;JMP
18 (IF_LT_6)
19     @SP
20     A = M
21     M = -1              // Push True on stack
22     @SP                 // Adjust SP
23     M = M + 1
24 (END_IF_ELSE_6)
```

Note the form of the labels in lines 18 and 24. The VM program may have multiple *lt* commands, located at different line numbers. Therefore, when generating the jump labels corresponding to each of the different *lt*

commands, it is important to avoid duplicate label names (this will result in incorrect program flow). One way to associate unique labels with each *lt* command is to tag the `'IF_LT_'` and `'END_IF_ELSE_'` strings with the line number of the specific *lt* command in the VM code, as done in lines 18 and 24. The *eq* and *gt* commands must be handled similarly.

Generating Hack assembly code for relational operators is straightforward. Following is an example of Python code that does it.

```
def generate_relation_code(operation, line_number):
    """Generate assembly code to perform the specified relational operation.
    The two operands are popped from the stack and result of the operation
    pushed back in the stack.
    """
    s = []
    label_1 = ''
    label_2 = ''

    s.append('@SP')
    s.append('M=M-1')
    s.append('A=M')
    s.append('D=M')              # D = operand2
    s.append('@SP')
    s.append('M=M-1')            # Adjust SP
    s.append('A=M')

    if operation == 'eq':
        # Your code goes here

    if operation == 'lt':
        s.append('D=M-D')        # D = operand1 - operand2
        label_1 = 'IF_LT_' + str(line_number)
        s.append('@' + label_1)
        s.append('D;JLT')        # if operand1 < operand2 goto IF_LT_*
        s.append('@SP')
        s.append('A=M')
        s.append('M=0')          # Push result on stack
        s.append('@SP')          # Adjust SP
        s.append('M=M+1')
        label_2 = 'END_IF_ELSE_' + str(line_number)
        s.append('@' + label_2)
        s.append('0;JMP')
        s.append('(' + label_1 + ')')
        s.append('@SP')
        s.append('A=M')
        s.append('M=-1')         # Push result on stack
        s.append('@SP')          # Adjust SP
        s.append('M=M+1')
        s.append('(' + label_2 + ')')

    if operation == 'gt':
        # Your code goes here

    return s
```

The *if-goto* and *goto* commands can be translated into the following Hack assembly-language commands.

```
// if-goto
@SP
M = M - 1
A = M
D = M           // Pop result off the stack
@label
D;JNE           // if D != 0 goto label

// goto
@label
0;JMP           // goto label
```

Finally, inserting labels into the stream of generated Hack assembly commands can be done as follows.

```
def generate_pseudo_instruction_code(label):
    """Generate pseudo-instruction for label."""
    s = []

    s.append('(' + label + ')')
    return s
```

To summarize, The VM language features three program flow commands:

- label *label*: This command labels the current location in the program's code. Only labeled locations can be jumped to from other parts of the program. The scope of *label* is the function in which it is defined (more on this later when we discuss functions). The string *label* contains a sequence of letters, digits, underscore (_), dot (.), and colon (:) that does not begin with a digit.
- goto *label*: This command effects an unconditional *goto* operation, causing execution to continue from the location marked by the *label*. The jump destination must be located in the same function.
- if-goto *label*: This command effects a conditional *goto* operation. The stack's topmost value is popped; if the value is non-zero, execution continues from the location marked by the *label*; otherwise, execution continues from the next command in the program. The jump destination must be located in the same function.