Sabanci University

Operating Systems

CS 307

Programming Assignment - 4: LC-3 Language to Write Programs for the LC-3 Virtual Memory Simulator

Release Date: 3 May 2025
Deadline: 12 May 2025 23.55

1 Introduction

For the virtual memory simulator that you are going to implement in the Programming Assignment - 4 (PA4), we are introducing a new language, the LC-3 Language, which you can use to write your own programs to run on the simulator.

LC-3 Language supports up to 4 registers (R1, R2, R3, and R4), which can be called the **value registers**, to store values in a variable-like manner, and the actual **variables** that are stored in the heap segment of the simulator, similar to the variables in the actual programming languages for other platforms and architectures.

Note: You can only store integer values within the range $[-2^{15}, 2^{15})$ in the value registers and variables, and all the integers in the LC-3 architecture are assumed to be signed integers (Source: The Wikipedia page for the LC-3 architecture).

2 Language Syntax

2.1 Basic Rules

The expressions are written into the individual lines in LC-3 Language, similar to Python; therefore, it does not have a delimiter such as `;` as in C or C++ for the lines, and the if-else blocks and while blocks have to be concluded with an `end` keyword as it is going to be explained in the subsequent sections.

LC-3 Language also supports comments added into the codes as single-line comments; you can use the characters `//` to begin your comment sections whether inline or covering an entire line.

Note: Multi-line comments as blocks on their own are not supported in LC-3 Language.

In order to write your codes in LC-3 Language and compile them for the simulator, you should perform the following steps:

- Write your codes into a text file with the extension `.lc3`, e.g., `filename.lc3`.

- If you would like to compile and assemble your code in a single step, run the following command (assuming you are using the CS 307 server):

```
    $ python3 lc3lang.py filename.lc3
```

  - By this step, you will have generated the following files: `filename.asm`, which is the assembly code corresponding to your code, and the files `filename_code.obj` and `filename_heap.obj`, which you can pass to the simulator as the code segment and the heap segment of your program, respectively.

- Alternatively, you can run the compiler, `lc3c.py`, and the assembler, `lc3a.py`, separately, which is going to be explained in the subsequent sections.

2.2 Assignment Operation

The assignment operator in LC-3 Language is `=` as in most programming languages; the left-hand side of an assignment can be a variable or a value register, while the right-hand side can be a value register, a variable, a constant value, and the supported arithmetic expressions containing the use of the aforementioned tokens.

Below, you can find examples for the assignment operation in LC-3 Language:

```
R1 = R2
```

```
R1 = var
```

```
R1 = 2
```

```
R1 = R2 + 2
```

```
R1 = 2 + R2
```

```
R1 = var + 2
```

```
R1 = 2 + var
```

```
R1 = 3 + 2
```

```
R1 = R2 - 2
```

```
R1 = 2 - R2
```

```
R1 = var - 2
```

```
R1 = 2 - var
```

```
R1 = 3 - 2
```

```
var1 = R2
```

```
var1 = var2
```

```
var1 = 2
```

```
var1 = R2 + 2
```

```
var1 = 2 + R2
```

```
var1 = var2 + 2
```

```
var1 = 2 + var2
```

```
var1 = 3 + 2
```

```
var1 = R2 - 2
```

```
var1 = 2 - R2
```

```
var1 = var2 - 2
```

```
var1 = 2 - var2
```

```
var1 = 3 - 2
```

2.3 Arithmetic Operations

The supported arithmetic operations in LC-3 Language are `+` and `-`, which can be utilized within the arithmetic expressions containing value registers, variables, and constant variables; you can use arithmetic expressions at the right-hand side of the assignments in LC-3 Language.

Below, you can find examples for the arithmetic expressions supported in LC-3 Language:

```
R1 + 2
```

```
2 + R1
```

```
var + 2
```

```
2 + var
```

```
3 + 2
```

```
R1 - 2
```

```
2 - R1
```

```
var - 2
```

```
2 - var
```

```
3 - 2
```

2.4 Comparisons

LC-3 Language supports the following comparisons between value registers, variables, and constants:

- `==` for the check of **equals**
- `!=` for the check of **not equals**

- `<` for the check of **strictly less than**
- `<=` for the check of **less than or equal to**
- `>` for the check of **strictly greater than**
- `>=` for the check of **greater than or equal to**

Note: You can only use comparisons to serve as conditions in the conditional expressions and loops in LC-3 Language.

Below, you can find some examples for the comparisons supported in LC-3 Language:

```
R1 == R2
```

```
R1 != var
```

```
var != R1
```

```
R1 > 5
```

```
5 < R1
```

```
var > 2
```

```
2 < var
```

```
3 >= 2
```

```
2 <= 3
```

2.5 Conditional Expressions

Conditional expressions are supported by LC-3 Language as `if-else-end` blocks and `if-end` blocks depending on the comparisons; you can check some of the examples of conditional expressions in LC-3 Language below:

```
if (o2 >= 1)
     R1 = R1 + R4
else
     R1 = R1 - 5
end
```

```
if (iteration < 6)
    iteration = iteration + 1
end
```

Furthermore, LC-3 Language supports nested `if-else-end`/`if-end` blocks, one of whose examples is presented as follows:

```
if (value < threshold)
    value = value + 2
    if (value > threshold)
         value = threshold
    end
else
    value = value - 1
end
```

2.6 Loops

LC-3 Language supports while loops through the `while-end` blocks, a basic example for which is available below:

```
while (iteration < limit)
    tempVar = currentSum
    currentSum = currentSum + previousSum
    previousSum = tempVar
    iteration += 1
end
```

In addition to the basic while loops, LC-3 Language supports nested while loops as well:

```
while (x != 0)
    inner = y

    while (inner != 0)
         result = result + 1
         inner = inner - 1
    end

    x = x - 1
end
```

Moreover, you can write a while loop inside a conditional expression, or vice versa:

```
if (x != 0)
    inner = y

    while (inner != 0)
         result = result + 1
         inner = inner - 1
    end

    x = x - 1
end
```

```
while (o1 != 0)
    if (o2 < 1)
        R1 = R1 + R4
    else
        R1 = R1 - 5
    end

    o1 = o1 - 1
end
```

Note: LC-3 Language does not support for loops.

2.7 Instructions Supported as Keywords

LC-3 Language supports direct invocation of the `yield` and `brk` instructions, which you are going to implement in PA4, within the source code. You can write the following keywords in your codes to invoke these instructions directly:

```
YIELD // The yield instruction as you are going to
   implement in tyld()
```

```
BRK // The brk instruction as you are going to
   implement in tbrk()
```

3 Compiler

The compiler for LC-3 Language, `lc3c` resides at the file `lc3c.py`, and can be run with the following command (assuming you are using the CS 307 server):

```
$ python3 lc3c.py filename.lc3
```

The `lc3c` compiler takes an `.lc3` file, in which you had written your code, as its argument, and returns you two files:

- `filename.asm`: The file where your code in LC-3 Language is compiled by `lc3c` into the assembly language for the LC-3 architecture. You are going to pass this file into the assembler, which is going to be explained in the next section.
- `filename_heap.obj`: The binary file which serves as the initial heap segment of your program and contains the initial values for the variables as you initialized them in your code; you are going to pass this file into the simulator, `vm` as an argument, which is explained in the PA4 document.

4 Assembler

The assembler for LC-3 Language, `lc3a` resides at the file `lc3a.py`, and can be run with the following command (assuming you are using the CS 307 server):

```
$ python3 lc3a.py filename.asm
```

The `lc3c` compiler takes an `.asm` file, which you are supposed to generate using `lc3c`, the compiler for LC-3 Language, as its argument, and returns you a single file:

- `filename_code.obj`: The binary file containing the code segment of your program; you are going to pass this file into the simulator, `vm` as an argument, which is explained in the PA4 document.

5 Extra Resources

If you are curious about the computer architectures and assembly instructions, you can check, in the scope of LC-3 architecture, which is a computer architecture designed for educational purposes, and the PA4 as a kind of introduction, the following resources:

- The slides from University of Texas at Austin on the LC-3 architecture.
- The Wikipedia page for the LC-3 architecture.
- The blog by Andrei Ciobanu on the virtual machine which constitutes the basis of PA4 (link is already provided in the PA4 document).