ECE220: Computer Systems & Programming **Fall 2020 ZJUI**
Lab 11 **Tuesday 1 December**

Analyzing LC-3 Programs

In today's lab, you must write a depth-first search using a singly-linked list. The DFS will explore the instructions that can be reached from a single starting point in an LC-3 assembly program, allowing you to write a tool that provides helpful feedback to students learning LC-3. For example, to test whether the main code can reach a HALT instruction, whether subroutines can reach a RET instruction, and whether it is possible for any code to leave the code defined by the student. The goal is for you to start learning how to use pointer-based data structures, such as linked lists. Note that you need not use such skills in MP9 (they are needed for MP10), the midterm will include questions on the topic.

Begin by checking out the **`lab11`** subdirectory in your Subversion repository. The directory contains a copy of this document (**`lab11.pdf`**), a C header file **`lab11.h`**, a C source file **`lab11main.c`** that provides most of the code, a **`Makefile`**, and a C source file, **`lab11.c`**, with which you can start this lab. Finally, the subdirectory **`samples`** includes three sample LC-3 object files, and the subdirectory **`outputs`** contains the correct outputs for those files. You can immediately "**`make`**" the executable, but without your code, the analysis of LC-3 programs will neither be correct nor particularly useful.

What the Given Code is Doing

The code given to you reads an LC-3 object file into memory and does some basic analysis of the bits for each of the individual instructions in the file. Take a look at the header file. The **`code_stat_t`** records information about the whole program ("code statistics"). It includes a dynamically-allocated array of instructions (the **`inst`** field), which are **`instruction_t`**s, with length specified by the **`len`** field of the **`code_stat_t`**. Each instruction includes some information about incoming and outgoing instructions (those that can precede and follow the one described by the **`instruction_t`**). Each instruction also includes a bit vector of flags (in the field of the same name) that records additional specific information from the per-instruction analysis. The flags also include an **`INST_FLAG_FOUND`** bit that you must use to avoid infinite loops in your search.

The Task

You must implement the function **`do_search`**, in **`lab11.c`**. The function is invoked for the main LC-3 program and on every subroutine reachable from the main program. Note that an LC-3 program may appear to have subroutines that are not actually subroutines, since data values may appear to be JSR instructions.

The signature for your function is as follows:

```
inst_flag_t do_search (code_stat_t* cs, int32_t first);
```

The **`cs`** parameter provides information on the code, as discussed previously. The **`first`** parameter indicates the starting point for your search, and is an array index for the array of instructions in the **`cs`**.

The function must explore the instructions reachable from the starting point and perform a bitwise OR of the flags for all reachable instructions, then return the result of the bitwise OR. Note that most code contains loops, which could cause a problem for a DFS. Your code must use the found flag mentioned earlier to avoid re-exploring the same instructions repeatedly. More specific instructions are given as comments in the **`lab11.c`** file.