Goals

- Practice debugging RISC-V Code.
- Write RISC-V functions that use pointers.

Setup

Copy the lab files from the instructional servers to your lab account with

```
$ cp -r ~cs61c/labs/04/ ~/labs/04/
```

Alternatively, secure-copy (`scp`) them from the instructional servers to your own laptop with

```
$ scp -r cs61c-xxx@hive10.cs.berkeley.edu:~cs61c/labs/04/ ~/YOUR_FILEPATH
```

And if you want to secure-copy them back to your class account:

```
$ scp -r ~/YOUR_FILEPATH/04 cs61c-xxx@hive10.cs.berkeley.edu:~/YOUR_LABACCT_FILEPATH
```

Exercise 1: Debugging `megalistmanips.s`

In Lab 3, you completed a RISC-V procedure that applied a function to every element of a linked list. In this lab, you will be working with a similar (but slightly more complex) version of that procedure.

Now, instead of having a linked list of `int`s, our data structure is a linked list of `int` arrays. Remember that when dealing with arrays in `struct`s, we need to explicitly store the size of the array. In C code, here's what the data structure looks like:

```
struct node {
    int *arr;
    int size;
    struct node *next;
};
```

Also, here's what the new `map` function does: it traverses the linked list and for each element in each array of each `node`, it applies the passed-in function to it, and stores it back into the array.

```
void map(struct node *head, int (*f)(int)) {
    if (!head) { return; }
    for (int i = 0; i < head->size; i++) {
      head->arr[i] = f(head->arr[i]);
    }
    map(head->next, f);
}
```

Task: Find the mistakes inside the `map` function in `megalistmanips.s`. Read all of the commented lines under the `map` function in `megalistmanips.s` (before it returns with `jr ra`), and **make sure that the lines do what the comments say.**

Some **hints**:

- Why do we need to save stuff on the stack before we call `jal`?
- What's the difference between `add t0, s0, x0` and `lw t0, 0(s0)`?
- Pay attention to the types of attributes in a `struct node`!

Checkoff [1/2]

- Display the result of running the code in `megalistmanips.s`.

Exercise 2: Write a function without branches

Consider the discrete-valued function `f` defined on integers in the set `{-3, -2, -1, 0, 1, 2, 3}`. Here's the function definition:

- `f(-3) = 6`
- `f(-2) = 61`
- `f(-1) = 17`
- `f(0) = -38`
- `f(1) = 19`
- `f(2) = 42`
- `f(3) = 5`

Your **task** is to implement this function in RISC-V, with the condition that your code may NOT use **any** branch and/or jump instructions!

Luckily, you have access to an array of integers in the `.data` section of discrete_fn.s. How can you use this to your advantage?

Checkoff [2/2]

- Display the result of assembling and running the code in `discrete_fn.s`