Summarry

The liveness analysis happens before spilling and register allocation. It all happens in the liveness.ml file, and is called from the cigrid.ml file.

The input is the unspilled assembly, making it easy to work on the temporary registers. To complete a liveness analysis we have to:

- Create a graph
- Run the algorithm on it

To create the graph I've chosen a hashmap. Because of the nature of the graph we have to create, we won't always have the nodes that are our successors ready when adding a node, this is problematic if we want to create a direct reference.

Why hashmap

To avoid that the hashmap is a loose reference. When creating a node we trust that there is going to be a node with that name in the fututre and the hashmap is going to resolve that reference with looking up the name.

This could lead to performance issues as looking up successors all the time is O(1) but still significantly slower than just having the reference. That could easily be solved by converting it to a direct reference model, but for code I am expected to analyze it just doesn't matter.

Inner workings

To create the graph, I unpack the functions and all the blocks inside of them. For each block I create a named instruction as a starting point, because that is how jmp instructions refer to them and how I am later able to find them in the hashmap.

An alternative would be to name it instruction N and keep a table, of references, this would require another pass through the graph later to fix all the remaining wrong references.

I then handle straight line code inside of each block and make each instruction a node. A node is a record that contains the name, the list of successors (just a list of strings to later find them in the hashmap), the def and use lists, and live-in and live-out as mutable lists as we have to change them a lot. At this point live in and live out are empty. The name and the successor are just _instrN as this makes it simple to connect them.

After the graph is generated and stored in a global hashmap I run the function that performs the liveness analysis.

The liveness analysis has two main parts, the live_upd function that analyzes one node at a time, checking it's successors and calculating the live_in and live_out parameters as described in the lecture slides (it also removes duplpicates for readability), and updates them if necessary.

And the outer liveness function that in a loop runs the live_upd on all the nodes and then checks if any of them tripped the changed flag. This is to ensure that the algorithm stops as soon as there is an iteration, that doesn't change anything.