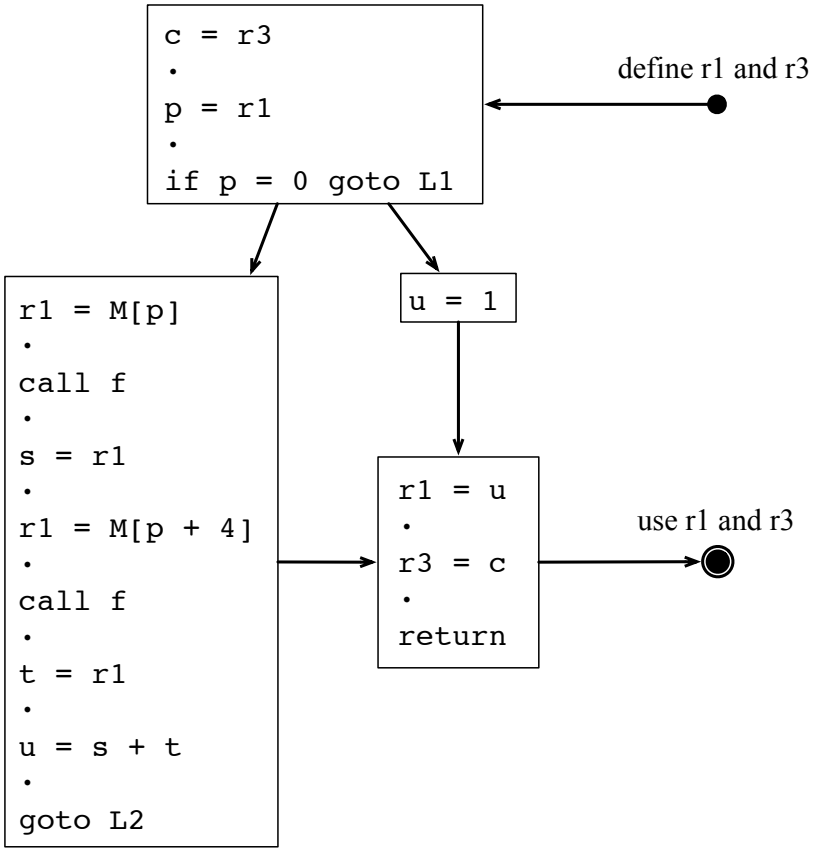
DCC888 – Register Allocation

Name: _____ ID: _____

1. Suppose a function f is using register r to hold a local variable and calls procedure g, which also uses r for its own calculations. Then r must be saved before g uses it and must be restored after g is done with it. We say that r is a *caller-save* register if the caller, e.g., f, must save and restore the register, and r is *callee-save* if it must be saved by the callee, e.g., g.

   In this question you will have to perform register allocation on the program below, assuming a target machine with three physical registers: r_1, r_2 and r_3. Let us assume that, due to compiler conventions, r_1 and r_2 are caller-save registers, and r_3 is callee-save. Furthermore, r_1 is used to pass arguments and to read back the return value of functions.



(a) Perform liveness analysis in the program, showing, at each program point, or control flow edge, which variables are alive there.

(b) Draw the interference graph of this program, given the liveness information from the previous question.

(c) Three registers only is not enough to allocate every variable in this program, even more if we consider that r_2 will be overwritten by function `f`. In order to perform register allocation in this program, you will have to spill one or more variables. Insert loads and stores in the original program, to map variables to memory.

(d) Draw the interference graph after spilling. This interference graph must be *Kempe Simplifiable*. In other words, you must be able to color it by applying Kempe's heuristics.

(e) Show the final mapping of variables to registers that will be produced by iterated register coalescing.

(f) Which variables have you been able to coalesce in this example?

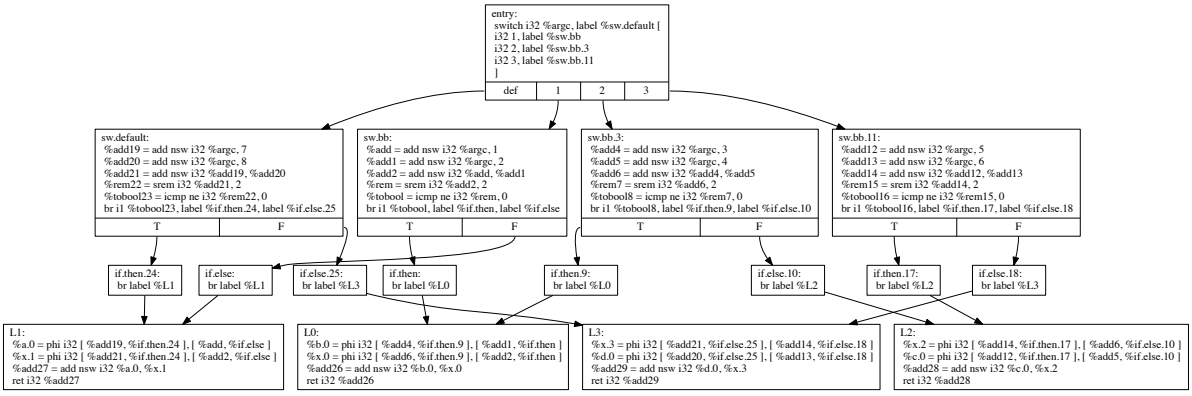


Figure 1: Program that contains the structure used by Chaitin to describe C_4.

2. To prove that the Register Assignment Problem was NP-Complete, Gregory Chaitin has shown how Graph Coloring can be reduced to this problem. To do so, he has shown that, for any graph, there exists a program that represents it via its interference pattern. The reduction of Chaitin works for general programs. It does not work for programs in Static Single Assignment form, as we shall see in this exercise.

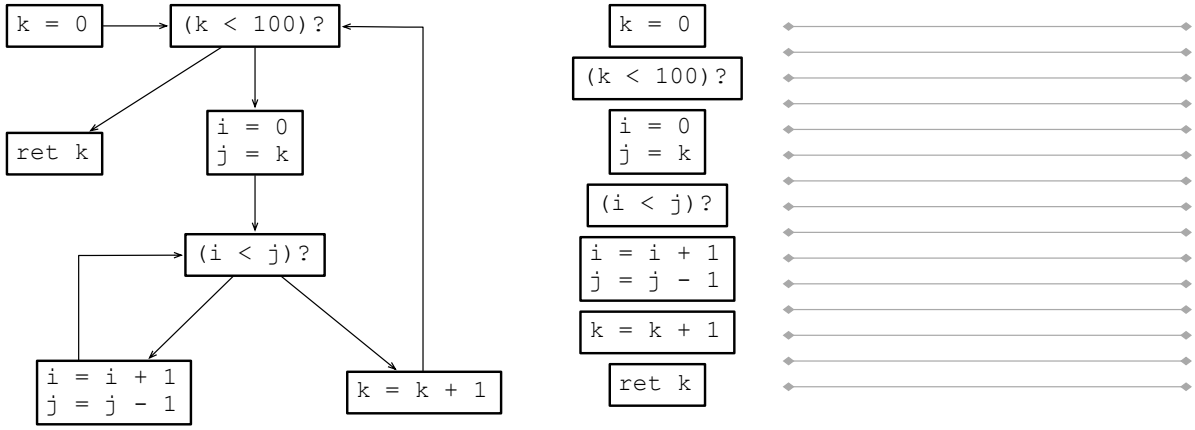
   (a) Consider the CFG in Figure 1. It was produced from a C program, that we have compiled using LLVM, to represent the cycle of four vertices, C_4. Can you write a C program that would lead to a similar structure of vertices and nodes?

   (b) Take a look into the interference pattern between variables in the program in Figure 1. Do you think we would get C_4 as a subgraph of the interference graph?

   (c) How many registers do you think you would need to assign every variable in the program in Figure 1 into registers?

   (d) Try a bit to produce a graph that contains C_4 as an *induced subgraph*. You shall see that this is rather difficult in LLVM. Can you come up with some intuition on why this task is so hard?

3. Consider the program on the left, in the figure below. Next to this program, we have a linearization of its basic blocks.



(a) Can you use the lines on the right side of the figure above, to find a register assignment for the variables in our program, assuming that you have three registers available?

(b) Repeat the same procedure, but this time, assuming only two registers. You will have to spill. Linear scan asks you to spill the register with the furthest use in the future. Is this the best heuristics for this program?

(c) Consider, now, a variation of the same program, which we can see at the bottom of the page. Perform register assignment for this program, assuming only two registers. Use the space in the middle to linearize the basic blocks.