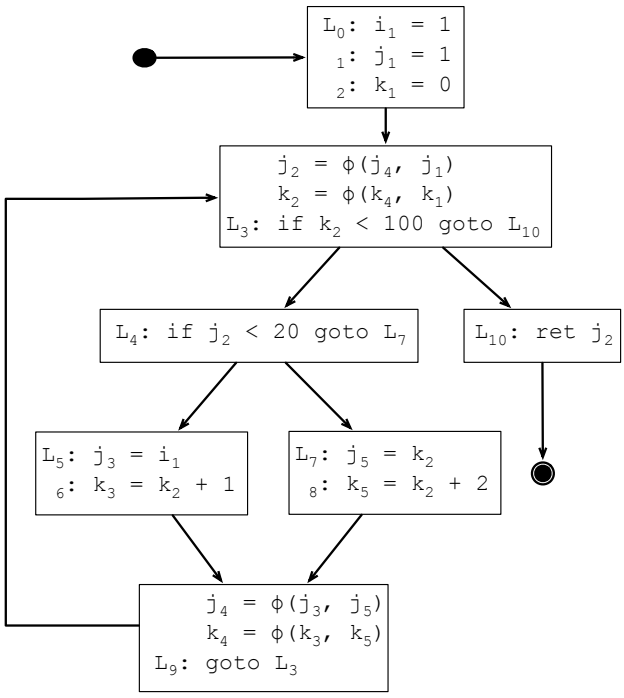
DCC888 – SSA Based Register Allocation

Name: _____ ID: _____

1. This question refers to the program below, which is in Static Single Assignment form.



   (a) Draw the interference graph of this program.

   (b) Find a simplicial elimination ordering for this graph.

   (c) Find a coloring of the graph, by assigning colors greedily to the nodes in the inverse order of the simplicial elimination ordering.

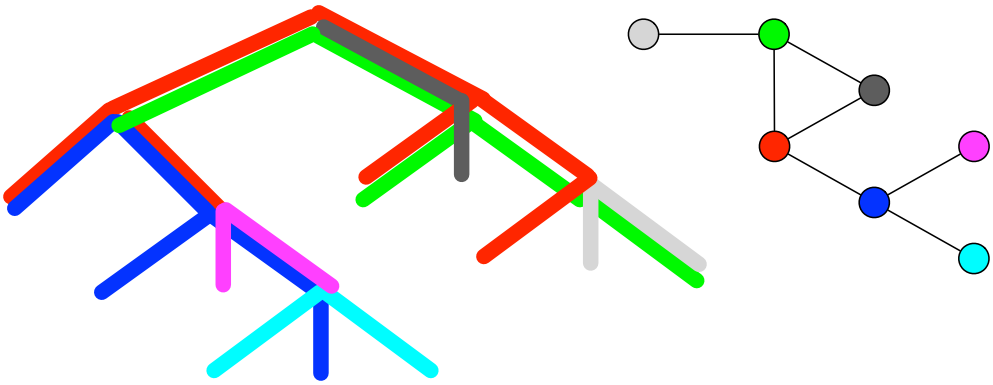
   (d) Assume that you have only two registers, `R0` and `R1` to compile this program. Because the register pressure is higher than two, you will have to spill variables. Which variable will you spill? Justify your answer based on some heuristic to mitigate the cost of spilling.

   (e) Re-write the program, this time after placing loads and stores to map the spill to/from memory. Assume the existence of two instructions:

   - `*m = v`: stores the value of `v` into the address `m`.
   - `v = *m`: load the value stored in address `m` into variable `v`.

   (f) Repeat steps (1.a-1.c) for the new program that you have produced in (1.e).

2. Consider the chordal graph below, which is represented in two different formats: on the left we have its representation as a set of subtrees onto a tree; on the right we have the traditional graph representation, with vertices and edges:



(a) Find an optimal coloring to for the graph on the right.

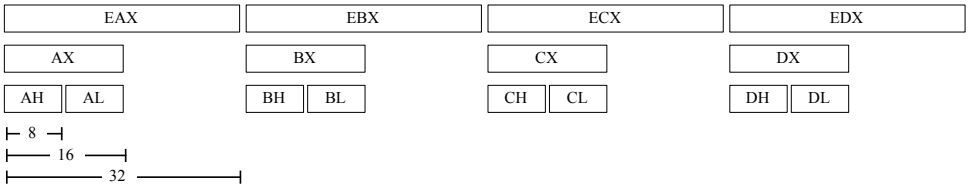
(b) Could you use the tree representation of a chordal graph to guide you in this task of finding an optimal coloring of the graph?

(c) Imagine that you have access to the dominance tree of a program, and the live ranges of the program's variables onto this tree. Describe a general procedure to perform register assignment and spilling based on this data-structure. This register allocation algorithm is called *tree-scan*.

3. Prove the following theorem:

> In a strict-SSA form program, D_v, the definition point of variable v, dominates every program point where v is alive.

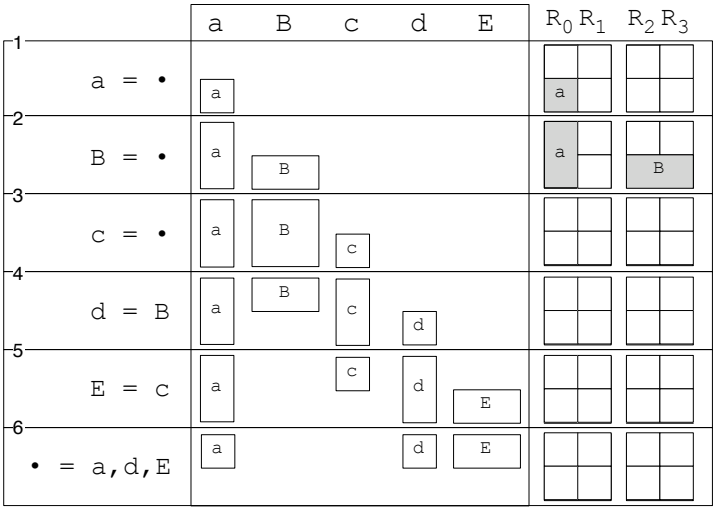
4. Many important computer architectures support registers of different sizes. For instance, the bank of floating-point registers used in ARM has 32 single precision locations that can be combined into 16 pairs of double precision registers. More impressively, SPARC V8 supports two levels of register aliasing: first, two 32-bit floating-point registers can be combined to hold a single 64-bit value; then, two of these 64-bit registers can be combined yet again to hold a 128-bit value. The general purpose registers used in the x86 form a even more complicated set up, which we see in the figure below:



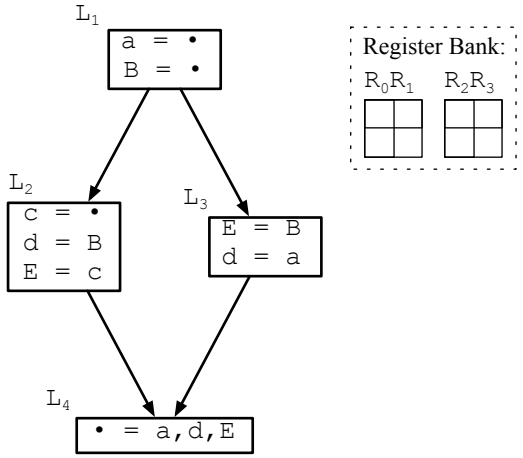
The rest of this question refers to an architecture with registers of different sizes sharing the same physical space, like the three examples we gave above.

(a) Let us assume a simple architecture with two registers, S_0 and S_1, which are further divided into two halves. S_0 contains two parts, R_0 and R_1, and S_1 contains the registers R_2 and R_3. In this architecture, would it be possible to compile the straight line program $a = \bullet; B = \bullet; c = \bullet; d = B; E = c; \bullet = a, d, E$, where each lower case variable fits into a single register R_i, and each upper case variable requires a double precision register S_j? The symbol $\bullet$ denotes an immaterial expression. You are not allowed to split the live ranges of the variables. Remember: you need to provide a reasoning that grounds your answer.

(b) What if we are allowed to split live ranges? In particular, if we can split live ranges at every program point, then the interference graph of the program belongs into the class of the so called *elementary* family of graphs. Color the third column of the next figure with the variables that you are keeping in registers. The first two assignments have been made for you. Will you need any register copy to avoid spills, and still preserve the semantics of the program in this example?



(c) Consider now the program below, which contains five variables. Assume the same architecture of the previous question. Convert this program to SSA form, using the space on the left, and answer the following question: do we still have the property that the maximum number of live variables (MaxLive) equals the minimum number of registers necessary to compile the program (MinReg)? Here you must adopt a broader definition of MaxLive: this metric gives you the maximum width of live variables in a program point. We will keep the convention that a lower case letter denotes a single precision value, and an upper case letter denotes a double precision value.



(d) If you are allowed to split live ranges at each program point, then how many registers will you need to compile the above program? Show, with a drawing, how your allocation would be in this case. Most likely you will need register copies and swaps. Emphasize the placement of these copies in your schematic allocation.