CS 243 Homework 3

Winter 2024

Due: February 7, 2024 at 11:59pm

Directions:

- Submit written answers via Gradescope.
- Complete the corresponding Gradiance quizzes by the due date.
- You have **two late days** on assignments for the entire quarter. See course website for details. **There are no late days for Gradiance.**
- This is an individual assignment. You are allowed to discuss the homework with others, but you must write the solution individually. If you look up any material in the textbook or online, you should cite it appropriately.

Problem 1. Optimizing dynamically typed languages

Consider a dynamically-typed language similar to JavaScript and Python. The language has three kinds of instructions:

- $x = T(\ldots)$, where T is one of $\{\texttt{int}, \texttt{float}, \texttt{string}\}$. This sets the variable x to a value of type T.
- $x = y$, where x and y are variables.
- $\text{print}(x)$, where x is a variable.

Unlike statically-typed languages like C and Java, the type of a program variable is allowed to change in this language. As an example, the program `x = int(1); x = float(3.14)` will result in variable `x` having an `int` value at first, but afterwards a `float` value.

Suppose we are writing an optimizing compiler for this language. Because there are special instructions for printing an integer (`iPrint`), printing a float (`fPrint`), and printing a string (`sPrint`), your task is to identify those print(...) instructions whose operand type can be determined at compile time. This problem has four parts: (a) Define your dataflow analysis by filling in the table below. (b) State clearly how you perform the optimization. (c) Is your dataflow analysis monotone? Explain your answer. (d) Is your dataflow analysis distributive? Explain your answer.

Part a. Dataflow analysis specification. You may assume that each instruction is in its own basic block.

Direction of your analysis	
Lattice elements and meaning	
Meet operator	
Is there a top element? If yes, what is it?	
Is there a bottom element? If yes, what is it?	
Transfer function	
Boundary condition	
Interior points	

Part b. How do you optimize the program?

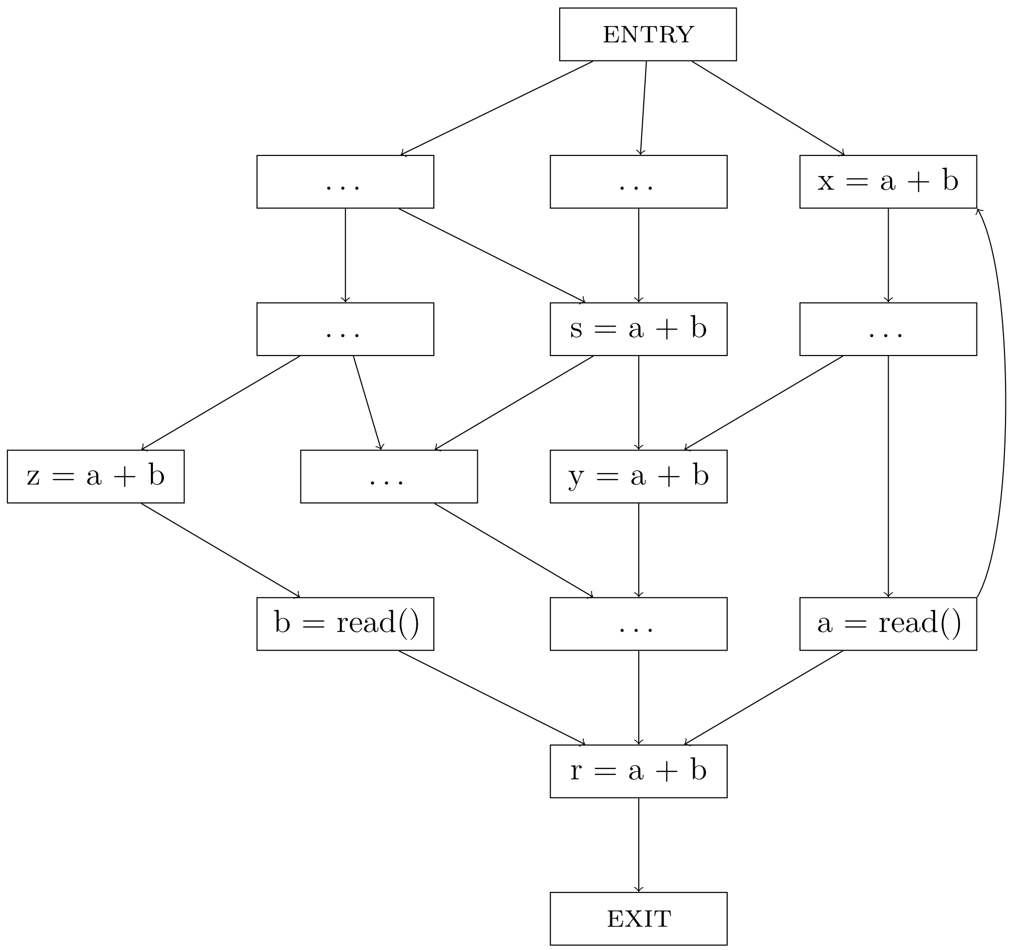
Part c. Is your dataflow analysis monotone? Sketch a proof for it if so, or provide a counterexample if not.

Part d. Is your dataflow analysis distributive? Sketch a proof for it if so, or provide a counterexample if not.

Problem 2. Partial Redundancy Elimination.

Apply partial redundancy elimination to the following program. For simplicity, we omitted instructions that do not redefine variables; you must assume that all variables are used in all basic blocks shown. As a special case, the expression "read()" can return different values when called at different times, and thus does not participate in the lazy code motion algorithm.

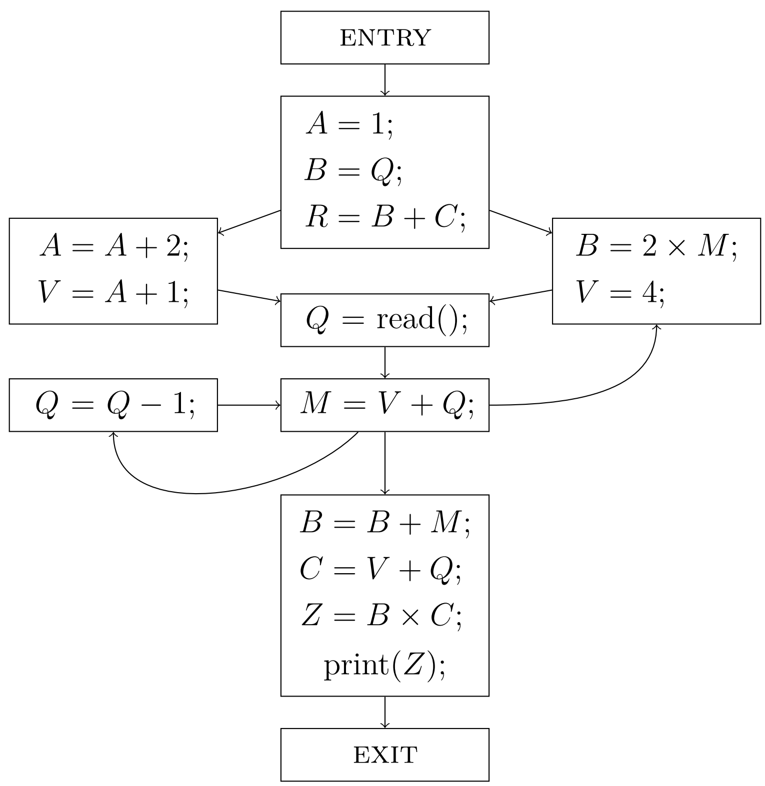
You do not need to show the intermediate steps – just show the optimized code. You may add basic blocks to the flow graph, but only show those that are not empty in your solution. (Existing "…" basic blocks are not empty. They merely do not redefine any variable.)



Problem 3. Your task is to optimize the code below. You are only allowed to run the following four optimization techniques (in any order and multiple times if necessary):

- Dead code elimination using liveness analysis (as discussed in class and Homework 2),
- Constant propagation (as discussed in Lecture 4),
- Partial redundancy elimination (as discussed in Lecture 5),
- Copy propagation (as discussed in § 9.1.5 (pp. 590–591) of the textbook).

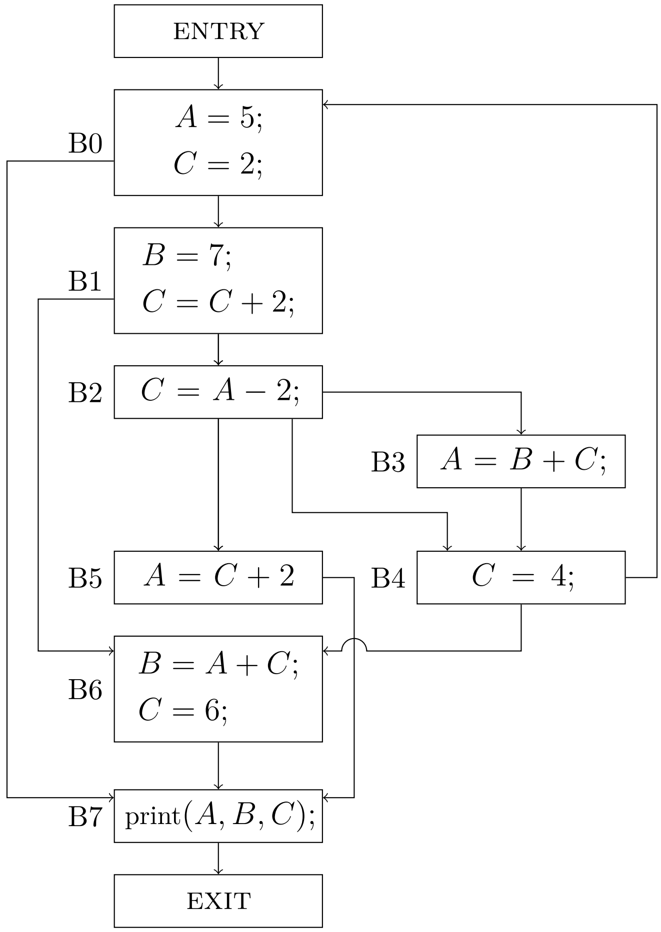
You cannot modify the control flow graph or eliminate empty basic blocks, except to pre-process it for PRE. Assume that expressions can take both registers and constants. You should not make any assumptions about the return value of read(), and assume print(Z) is some function that uses Z.



1. What are the optimizations and their order to produce the best optimized code for this specific program? Remember that you may run the same optimization more than once. You are allowed to abuse the optimizations, but try your best to minimize the number of passes of optimization to do the job.

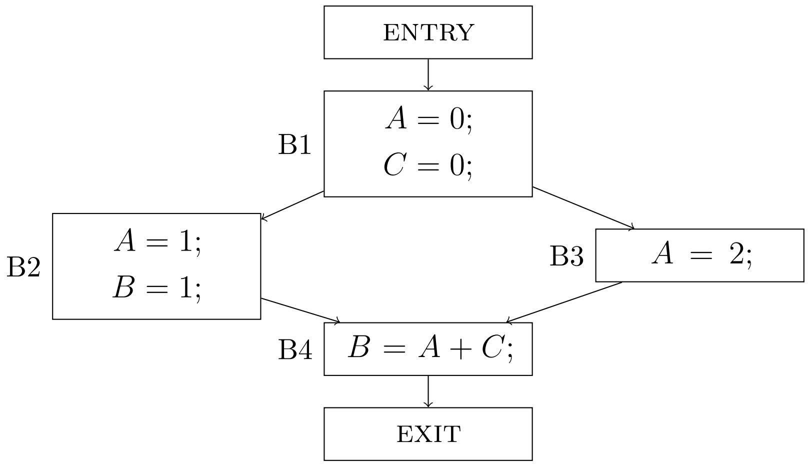
2. What is the final optimized program?

Problem 4. Register Allocation.

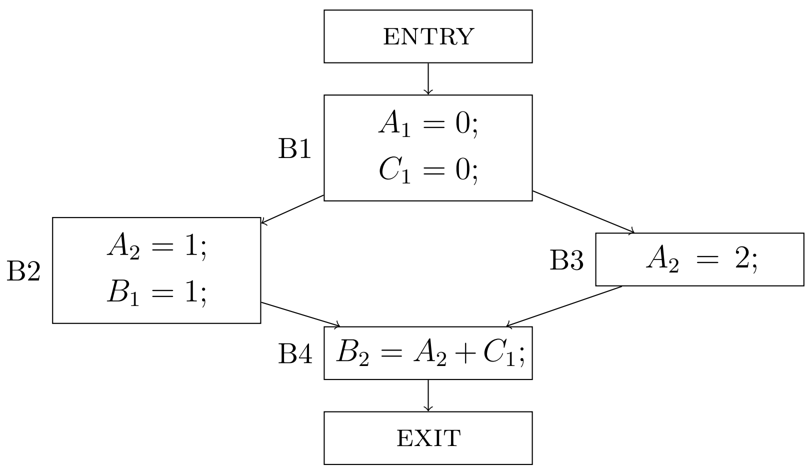


Perform register allocation for the above control flow graph. Specifically, show the results of each of the following steps:

1. Show the merged live ranges for the following program by updating the graph with unique variable notations, e.g., replace definitions and usages of B with B_1, B_2, etc. Note: if different definitions form a merged live range, use the same variable notation (you may convince yourself that this avoids ambiguity when a used variable may come from different definitions). We provide you with an example. For the following simple program:

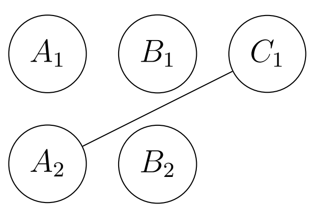


The resulting updated graph is:

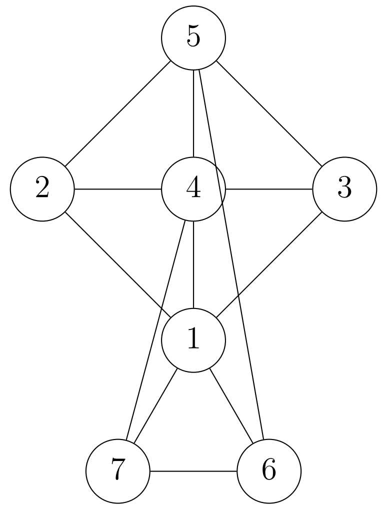


Taking the definitions of A in B2 and B3 as example, A is live at the beginning of B4, and both definitions reach that point, therefore they can be merged. **Note:** you do not need to consider any optimization such as PRE or constant propagation.

2. Draw the register interference graph with edges between nodes that represent merged live ranges. Using the same example program, the register inference graph should look like:



3. Apply the coloring algorithm for a machine with 3 registers to the following interference graph. Show a possible resulting "stack" of registers and show which ones, if any, are marked as spilled. When picking a node to spill, choose the node that is **lowest in numerical order**, and always pick the **first available color**.



4. Is this graph colorable using 3 or fewer colors? If so, give such a coloring using the fewest number of colors (you do not need to use the coloring algorithm from class). Otherwise, briefly explain why not.

Problem 5. Register Allocation and Live Ranges.

Part a.

1. Let n be the largest number of overlapping live ranges seen in a program. Let $v_1, \ldots, v_n$ be the set of n live ranges that overlap. What does this say about the corresponding nodes in the interference graph? Using your answer, give a bound on the minimum number of registers needed to execute this program without spilling or otherwise modifying the live ranges. You may assume the live range analysis is ideal, i.e. if two live ranges overlap, then there exists a program path that may be executed such that both variables are live at the same time.

2. Let n be the maximum number of *other* live ranges any live range overlaps with. For each of the following values of r, is register allocation: (i) always possible (ii) sometimes possible (iii) never possible? For (i) and (iii), give a brief proof sketch. For (ii), show examples for each case. You may assume the live range analysis is ideal.

   (a) $n - 1$ registers

   (b) n registers

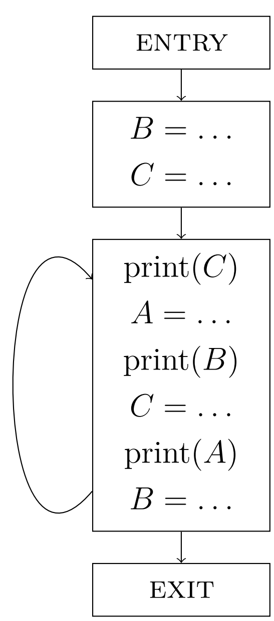
   (c) $n + 1$ registers

Part b. Observe the control-flow graph below and answer the following questions.

1. What is the largest number of overlapping live ranges seen at any program point?

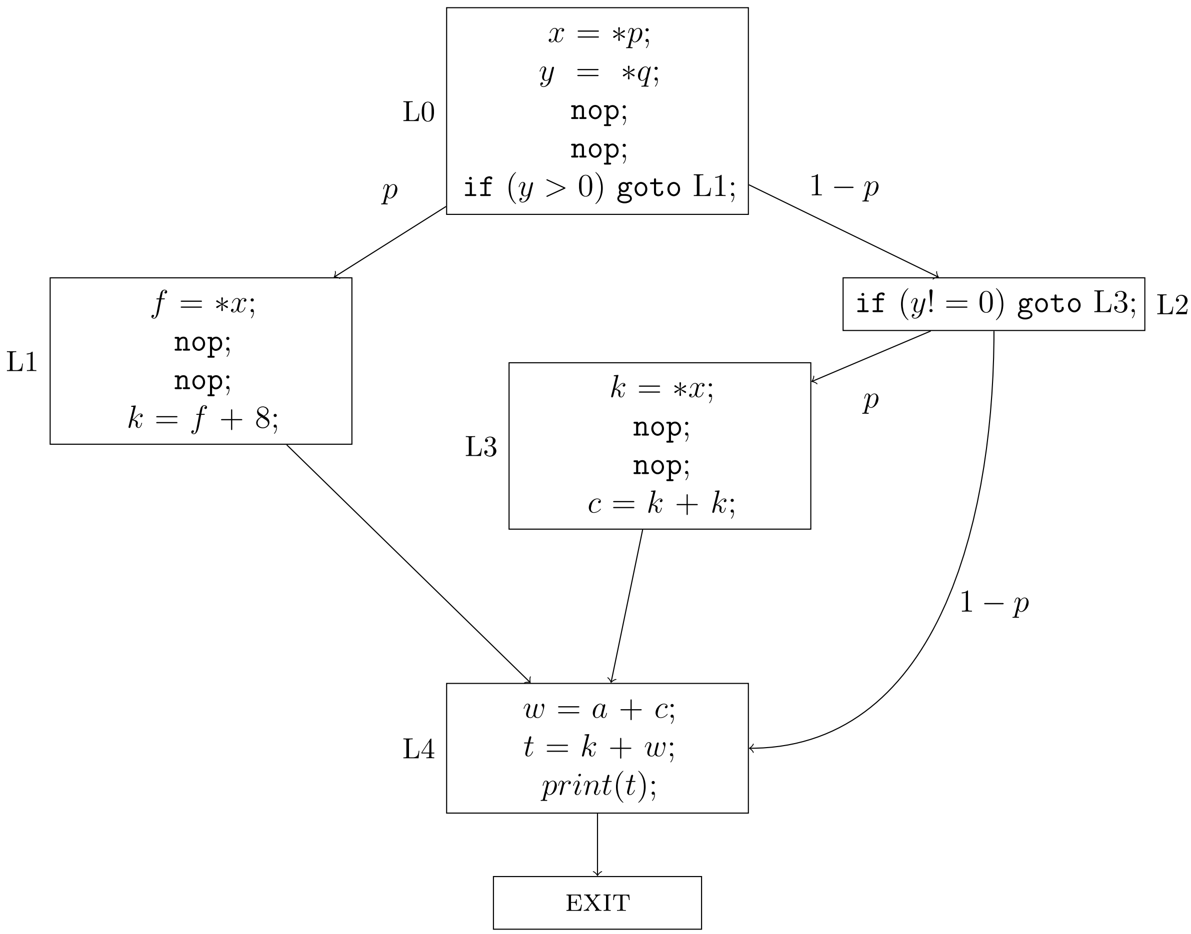
2. What is the minimum number of registers you need in order to successfully assign all variables without spilling?

3. Now, imagine that, as part of register allocation, you can insert `MOVE x y` operations that copy a value from register `x` to another register `y`. Can you allocate all of the variables with fewer registers than before? If so, show the modified program and give the new number of required registers.



Problem 6. Global Instruction Scheduling

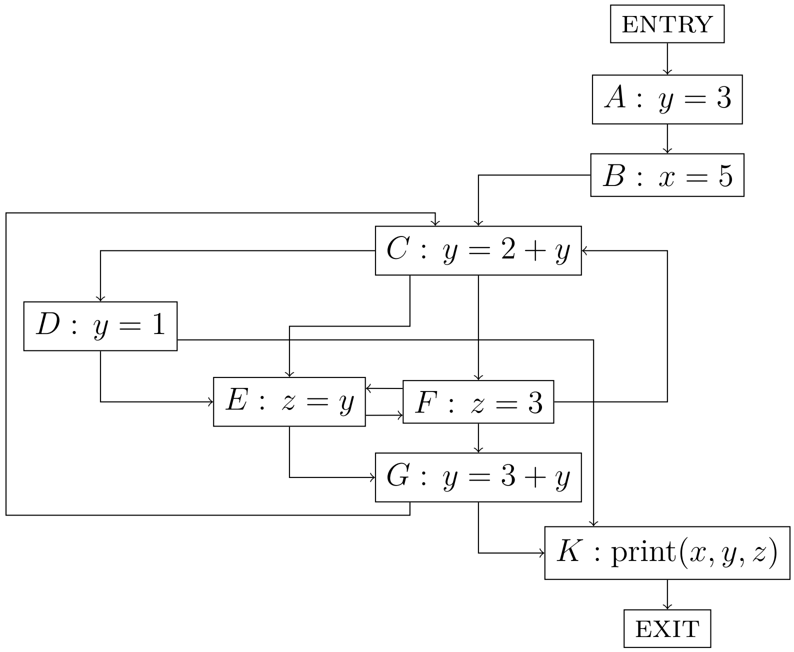
Assume you have a machine with a statically scheduled processor that can only issue one operation every clock. All operations have a latency of one clock cycle, with the exception of its memory load operation, which has a latency of three clock cycles. Consider the following locally scheduled program:



All variables in the program are defined before the entry (L0) of the program. Assume that only variable t is live at the exit of the program. Each branch in the flow graph is labeled with the probability that it is taken dynamically. To answer the following questions, you may apply any of the code motions discussed in class, but no other optimizations.

Is this the best globally scheduled code that can be generated given that $p = 0.8$? If not, provide the improved code along with its expected execution time. **Note: for an `if` branch instruction, assume that if the branch corresponding to "`goto` L" is taken, it costs two cycles; otherwise it costs only one.**

Problem 7. Dominance Relation.



Part a. A basic block d post-dominates another block n (sometimes written d pdom n) if all paths from n to the exit pass through d. Write a dataflow analysis that computes the set of nodes that post-dominate it (its post-dominators). In your analysis, let B be the set of basic blocks in the program, and b_{exit} a dummy basic block for the exit.

Direction of your analysis	
Lattice elements and meaning	
Meet operator	
Is there a top element? If yes, what is it?	
Is there a bottom element? If yes, what is it?	
Transfer function	
Boundary condition	
Interior points	

Part b. A node d *strictly* post-dominates n if d pdom n and $d \neq n$. Compute the strict post-dominators for each basic block. You do not have to write the block itself.

A :

B :

C :

D :

E :

F :

G :

K :

Part c. The immediate post-dominator of a node n is the unique node that strictly post-dominates n but does not strictly dominate any other node that strictly post-dominates n. Each node other than the exit node has an immediate post-dominator. Connect each node with its immediate post-dominator to form the *post-dominator tree*. **Hint:** as its name implies, the post-dominator tree is a tree (it is connected and has no cycles when interpreted as an undirected graph.)

Problem 8. [OPTIONAL] How many hours did it take for you to finish this assignment?