CS243 Homework 3

Winter 2021

Due: February 10th, 2021 at 4:00 pm

Directions:

- Submit via Gradescope.
- You may use up to two of your remaining late days for this assignment, for a late deadline of February 12th, 2021 at 4:00 pm.
- This homework includes a Gradiance quiz (please see the website) and the following questions, to be submitted via Gradescope.
- 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. Finding a min.

Consider a simplified language where variables can only have values in $\{1, 2\}$. The only constructs that can define a variable are:

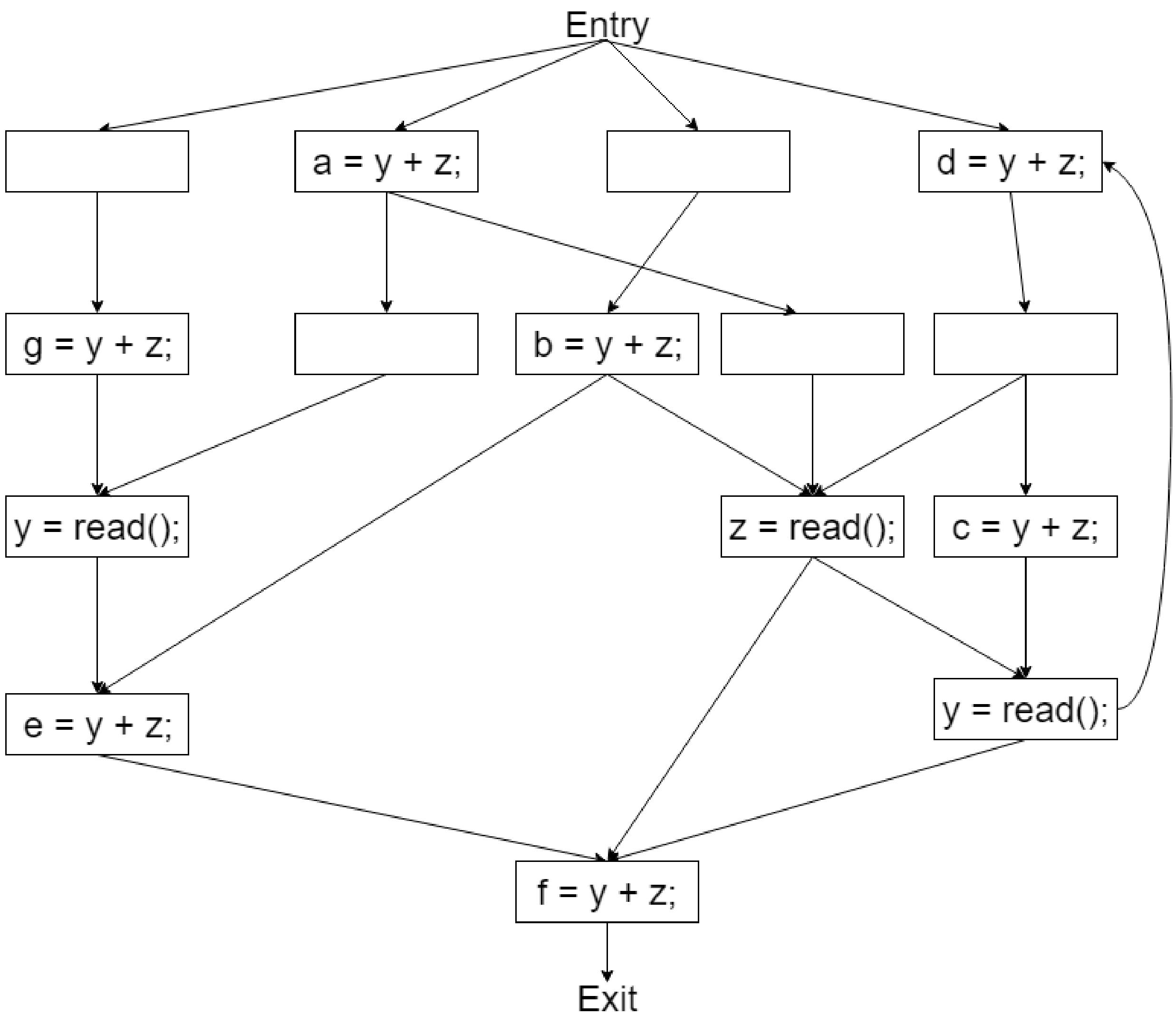
- `x = ` c where $c \in \{1, 2\}$
- `x = y` where `x` and `y` are variables
- `x = 3 - y` where `x` and `y` are variables (note that for $y \in \{1, 2\}$, $3 - y \in \{1, 2\}$).

Your task is to define a dataflow framework with any post-processing steps to find the minimum value for each variable at every program point. Assume that uninitialized variables can take on any value in $\{1, 2\}$. You may assume that each instruction is in its own basic block.

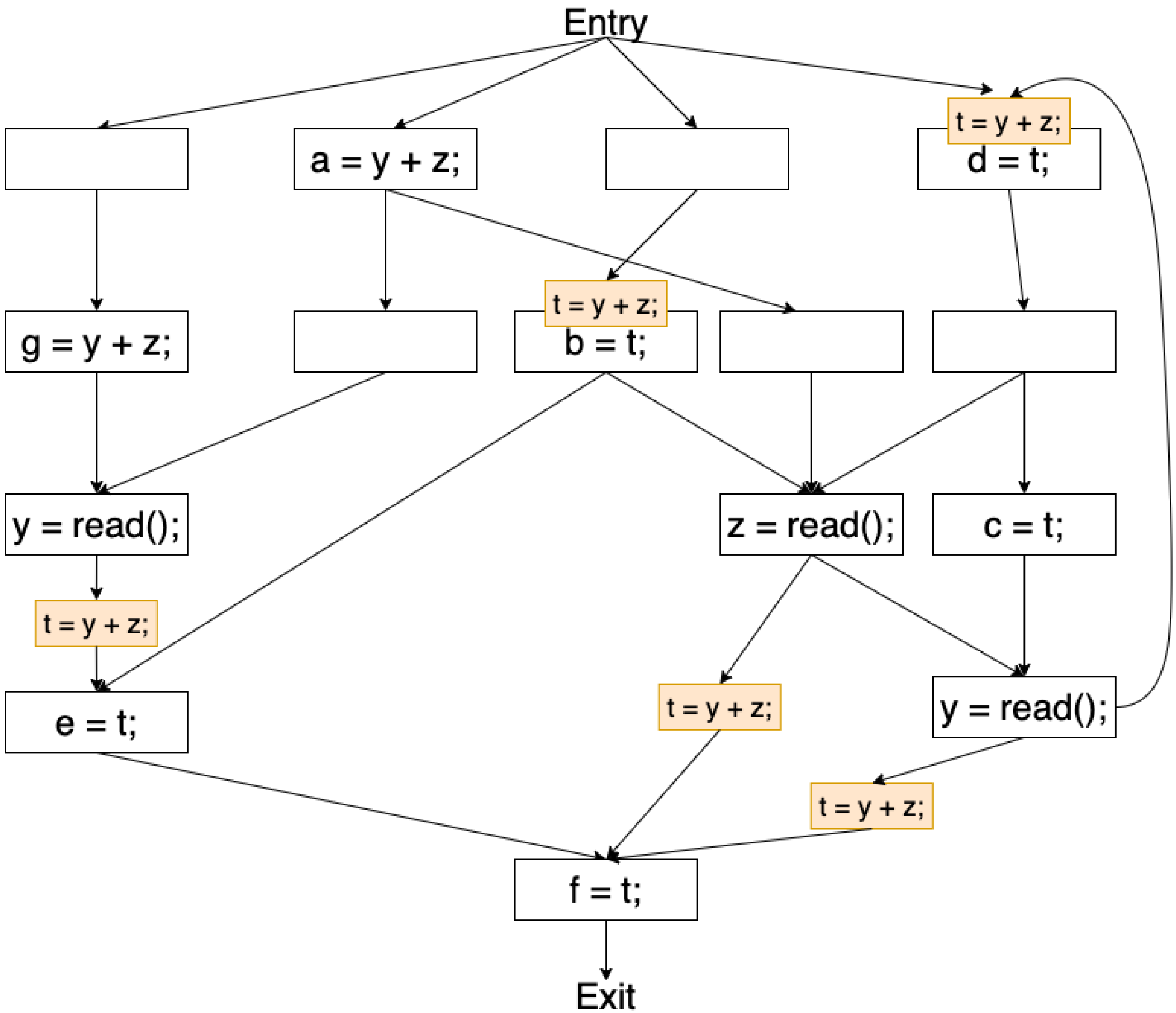
Direction of your analysis (forward/backward)	Forwards.
Lattice elements and meaning	Subsets of $\{1, 2\}$ corresponding to the values that each variable can take on. The product of the semi-lattices of all variables in the program will be the full semi-lattice.
Meet operator or lattice diagram	Union.
Is there a top element? If yes, what is it?	$\{\}$.
Is there a bottom element? If yes, what is it?	Universal set.
Transfer function of a basic block	`x = c`: $\text{out}[\text{b}][\text{x}] = \{c\}$. `x = y`: $\text{out}[\text{b}][\text{x}] = \text{in}[\text{b}][\text{y}]$. `x = 3 - y`: $\text{out}[\text{b}][\text{x}] = \{3 - y' \mid y' \in \texttt{in[b][y]}\}$ $\text{out}[\text{b}][\text{x}] = \text{in}[\text{b}][\text{x}]$ otherwise.
Boundary condition initialization	Universal set.
Interior points initialization	$\{\}$.

For any basic block b, to find the minimum value a variable can take on before or after the basic block, just compute min `in[b][x]` or min `out[b][x]`.

Problem 2. Apply PRE to the following program. Assume that variables a, b, c, d, e, f, g, y, and z are used (but not redefined) in any of the basic blocks shown. 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, even if they appear to be).



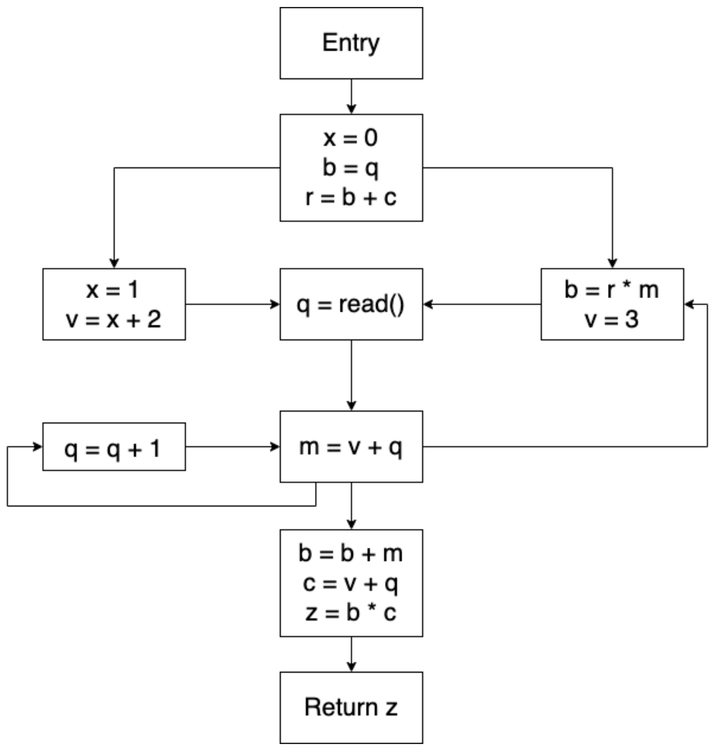
Solution:


Entry
a = y + z;
t = y + z;
d = t;
g = y + z;
t = y + z;
b = t;
y = read();
z = read();
c = t;
t = y + z;
e = t;
t = y + z;
y = read();
t = y + z;
f = t;
Exit

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):

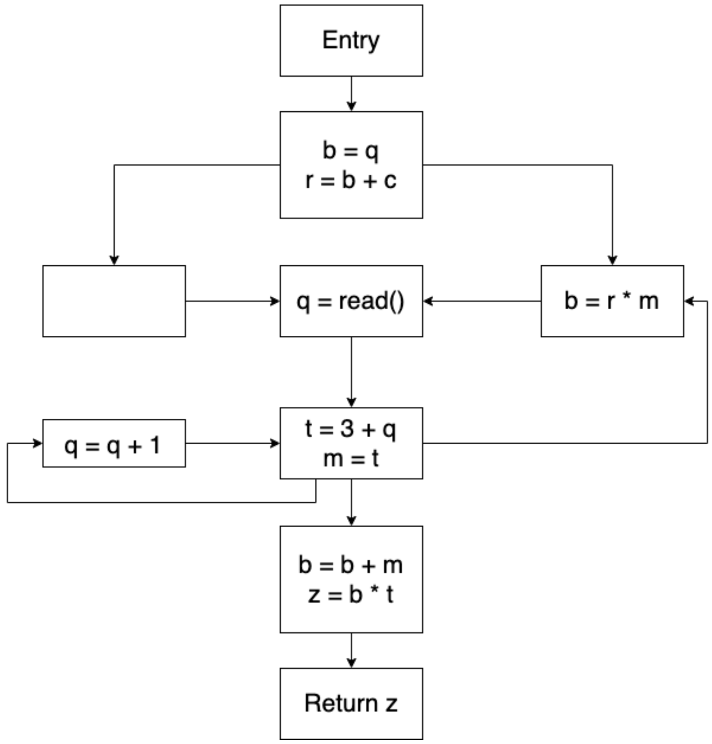
- PRE (as discussed in class)
- Constant Propagation (as discussed in class)
- Copy Propagation (as discussed in Section 9.1.5. of the textbook)
- Dead Code Elimination (liveness analysis, as discussed in class and in Homework 2)

You cannot modify the control flow graph or eliminate empty basic blocks, except to preprocess it for PRE. As in JoeQ, assume that expressions can take both registers and constants.



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.
2. What is the final optimized program?

First run constant propagation and DCE get rid of almost all code. After that, apply PRE to expression 3 + q. Next, apply copy propagation to make c dead. Finally, run a pass of DCE to eliminate c. You could end up doing a lot of passes with a different ordering. All orderings are accepted, as long as it produces the desired output.


Entry
b = q
r = b + c
q = read()
b = r * m
q = q + 1
t = 3 + q
m = t
b = b + m
z = b * t
Return z

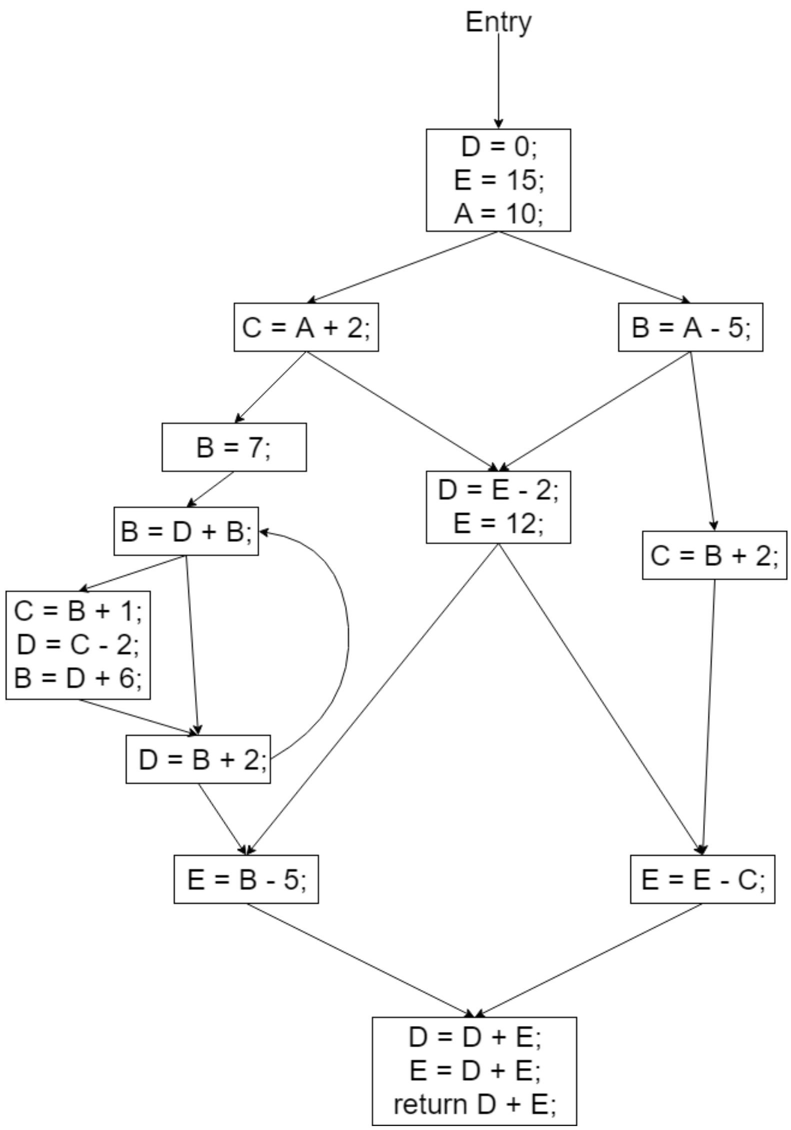
Problem 4. For the following control flow graph, perform register allocation. Show the results of the following steps.

1. Show the merged live ranges for the following program. Identify the different live ranges with unique variables (e.g. A_1, A_2, C_1, etc.). See graph below.

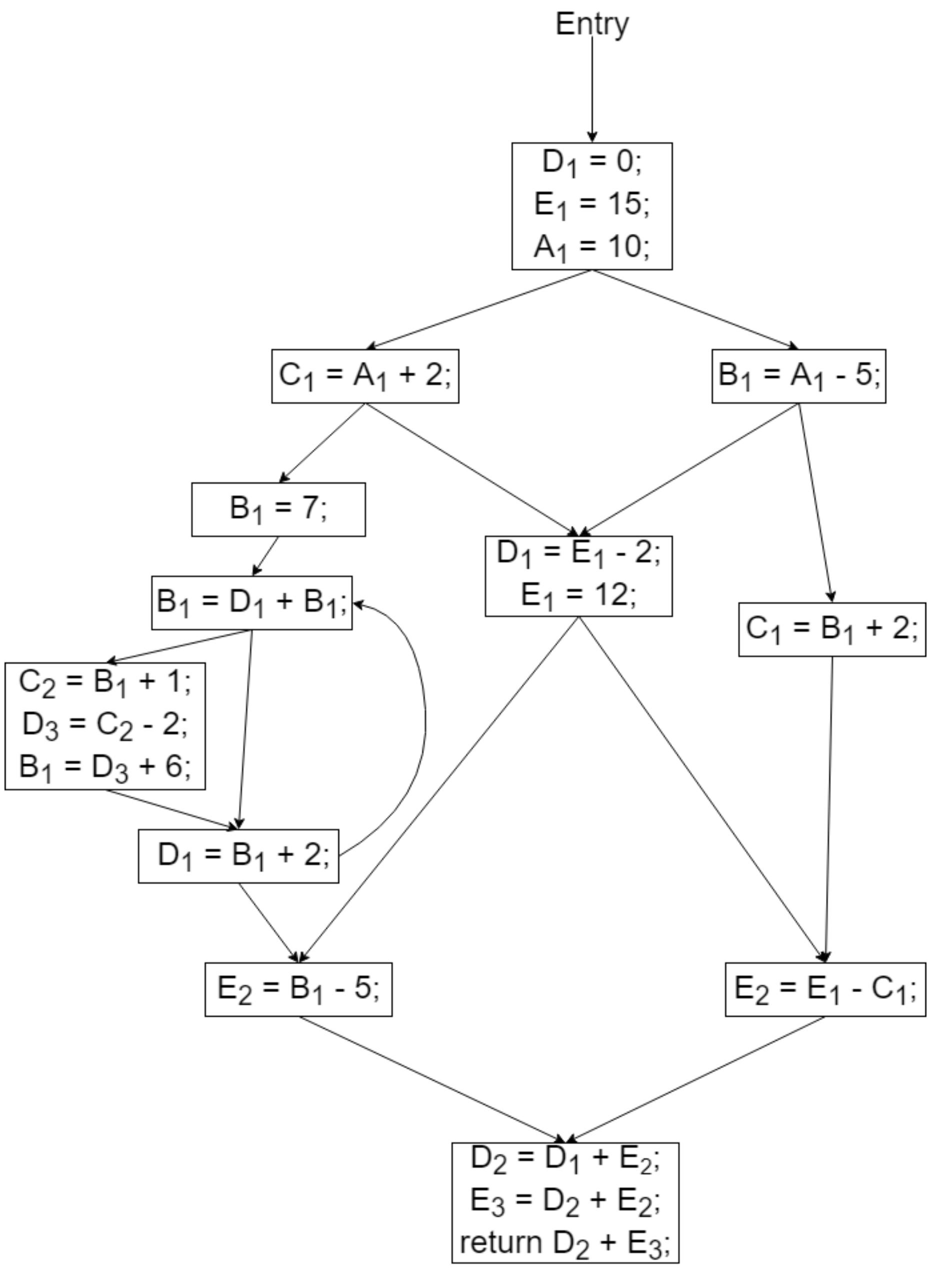
2. Draw the register interference graph with edges between nodes that represent merged live ranges. See graph below.

3. Apply the coloring algorithm for a machine with 3 registers to the interference graph from the previous part. Show the resulting "stack" of registers and show which ones, if any, are marked as spilled. There are many valid answers, one possible ordering: E3 D2 E2 A1 C1 C2 D3 B1 E1 D1.

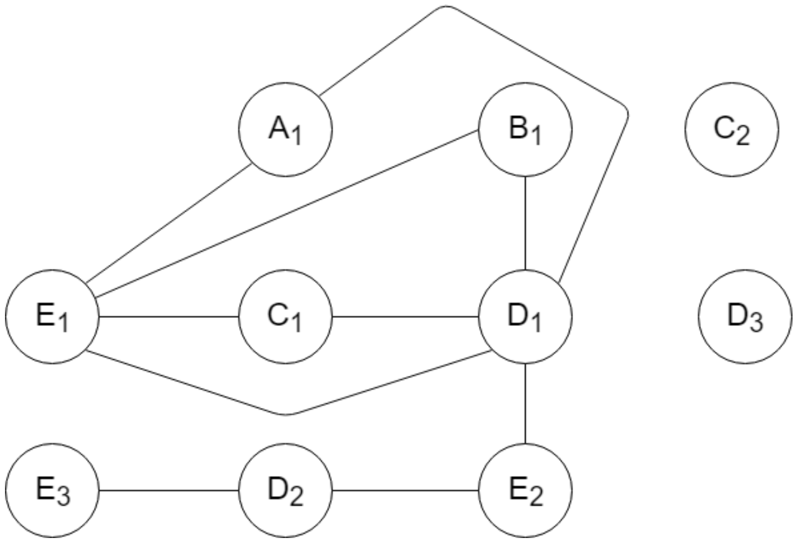
4. Assign the live ranges to registers. There are many valid answers, one possible assignment:
   (D1, R0), (E1, R1), (B1, R2), (D3, R1), (C2, R1), (C1, R2), (A1, R2), (E2, R1), (D2, R0), (E3, R1).



Live ranges graph:



Interference graph:


A1
B1
C2
E1
C1
D1
D3
E3
D2
E2

Problem 5. 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?

   Two live ranges.

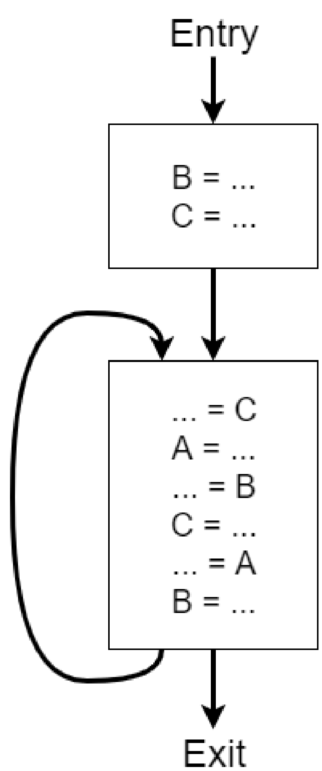
2. What is the minimum number of registers you need in order to successfully assign all variables without spilling?

   Three registers, due to the interference graph odd-length cycle between the live ranges of A, B, and C.

   Labeling the instructions from top to bottom as L1 .. L8. B_1: B1,B2,B3,B4,B5.in. B_2: B8.out, B3, B4,B5.in. B: B1,B2,B3,B4,B5.in, B8.out C_1: B2, B3.in. C_2: B6, B7, B8, B3.in. C: B2, B3.in, B6, B7, B8. A: B4, B5, B6, B7.in.

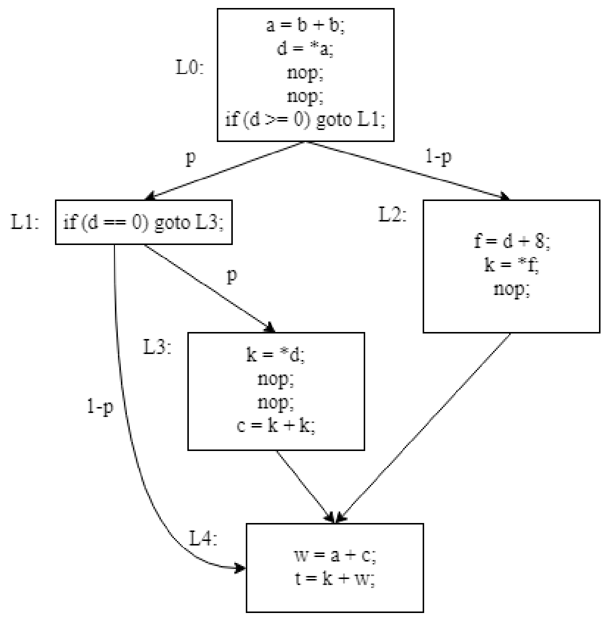
3. Now, imagine that, as part of register allocation, you can insert `MOVE x y` operations that copy a value from register x to another register y. Can you allocate all of the variables with fewer registers than before? If so, how many would it require?

   Two registers. For example, you can insert a `MOVE` after `...  = B` in the middle of the second basic block to swap A's register, which allows you to reuse the now free register for the allocation of C, breaking the cycle.



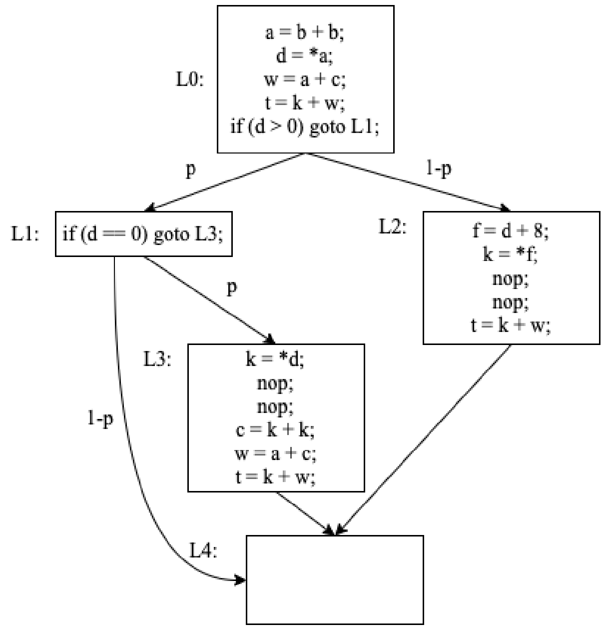
Problem 6. Global Instruction Scheduling

Assume you have a statically scheduled machine 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:

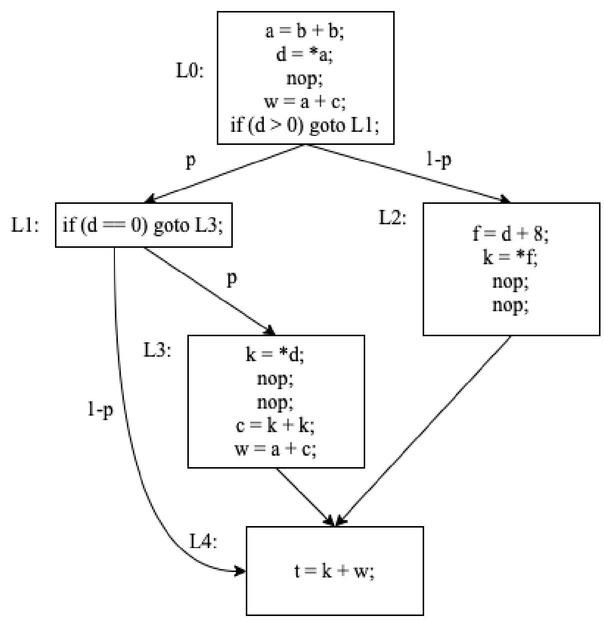


Assume that only t is live at the end of the program. Each branch in the flow graph is labeled with the probability that it is taken dynamically. To answer the following, you may apply any of the code motions discussed in class, but no other optimizations.

1. Is this the best globally scheduled code that can be generated given that p = 0.2? If not, provide the improved code along with its expected execution time and percentage improvement compared to the original program.



Push both t = k + w and w = a + c: Path1: L0,L1,L4: 6 instructions
Path2: L0,L2,L4: 10 instructions
Path3: L0,L1,L3,L4: 12 instructions.
The expected execution time is 6 * 0.16 + 10 * 0.8 + 12 * 0.04 = 9.44 clock cycles.



Only push w = a + c: Path1: L0,L1,L4: 7 instructions
Path2: L0,L2,L4: 10 instructions
Path3: L0,L1,L3,L4: 12 instructions.
The expected execution time is 7 * 0.16 + 10 * 0.8 + 12 * 0.04 = 9.6 clock cycles.

The original program with p = 0.2: 8*0.16 + 12*0.04 + 10*0.8 = 9.76
Therefore, the reduction in runtime is (0.32)/9.76 = 3.2% for proposal 1, or (0.16)/9.76 = 1.6% for proposal 2.

2. Repeat part 1 given that p = 0.5.

   Proposal 1 : 6 * 0.25+ 10 * 0.5 + 12 * 0.25 = 9.5.
   Proposal 2 : 7 * 0.25+ 10 * 0.5 + 12 * 0.25 = 9.75.

   The original program with p = 0.5: 8 * 0.25 + 10*0.5 + 12*0.25 = 10 cycles. Therefore, the reduction in runtime is (0.5)/ 10 = 5% for proposal 1, or (0.25) / 10 = 2.5% for proposal 2.