CSE 401/M501 22Sp Homework 4

Due: Thursday, June 2 by 11 pm. **Late submissions only until June 4, 11 pm,** even if you haven't exhausted your "late days", so that we may publish a solution before the final.

As before, submit via Gradescope. If possible, don't split the solution to a problem across a page break, and please keep it legible. Show your work to help us award partial credit if appropriate, and for TA sanity. You should do this assignment individually.

1. (leaders, basic blocks, and control flow graphs) (Appel) In the program at right, given as a sequence of 3-address instructions:
(a) List the numbers of the instructions that are *leaders* (a first instruction in some basic block) . Hint: recall that the first instruction (#1) is a leader, the target of every branch/jump/goto is a leader, and every instruction following a branch/jump/goto is a leader.

(Hint: The discussion of basic blocks and how to identify leaders in the Intermediate Representations (IR) lecture might be helpful.)

```
1.  m = 0
2.  v = 0
3.  if v >= n goto #15
4.  r = v
5.  s = 0
6.  if r < n goto #9
7.  v = v + 1
8.  goto #3
9.  x = M[r]      # contents of element r of array M
10. s = s + x
11. if s <= m goto #13
12. m = s
13. r = r + 1
14. goto #6;
15. return m
```

(b) Draw the Control Flow Graph (CFG) for this program. The nodes in the graph should be *basic blocks*. Each basic block starts with a leader instruction and should contain all subsequent instructions up to but not including the next leader that begins a different basic block. There should be edges from each basic block to each of its successors.

Each basic block should show in sequential order the instructions contained in it. Write both the instruction(s) and their original instruction number(s) in the CFG graph nodes.

Please number the basic blocks in your CFG as follows so the graphs will be consistent for grading: The first basic block beginning with instruction #1 should be basic block #1. The remaining basic blocks should be given sequential numbers 2, 3, … in ascending order of their leader instruction numbers. In other words, the first instruction in block 2 should have a larger instruction number than the first instruction in block 1 and a smaller instruction number than the first instruction in block 3, etc.

2. (dataflow analysis – live variables) **(a)** Use the dataflow analysis framework described in class to compute the set of variables that are live at the beginning of each basic block in the control flow graph from question #1.

Recall that the live variable dataflow problem is formulated from the following sets. For each basic block *b*, define

- use[*b*] = variables used in *b* before any def
- def[*b*] = variables defined in *b* and not killed later in *b*
- in[*b*] = variables live on entry to *b*
- out[*b*] = variables live on exit from *b*

To compute the variables that are live at the beginning of each basic block *b* (i.e., in[*b*]), first initialize

in[b] = out[b] = ∅

Then iteratively solve the following set of equations by updating the in and out sets for each block until no further changes occur:

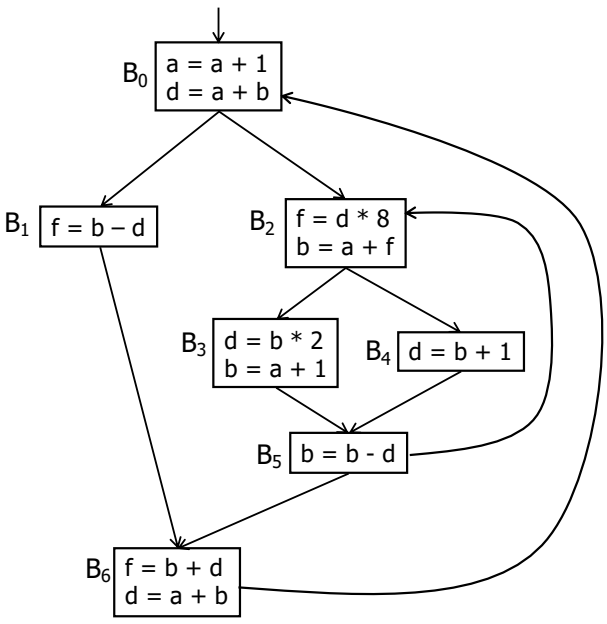
- out[b] = $\cup_{s \in \text{succ}[b]}$ in[s]
- in[b] = use[b] ∪ (out[b] – def[b])

Your analysis should include all of the scalar variables in the original program, but not the array *M*. Show your results for use/def and the in/out iteration in a table like the one in Dataflow lecture slides O-32..O-34 (approximately), but show one row per block. List, and process, blocks in reverse order of their numbers, as in the lecture example, and calculate out[b] before in[b] for each block b. Include enough columns to show that a fixed point has been reached; I think 3-4 in/out column pairs should suffice.

Use the instruction and block numbers and the control flow graph from your answer to problem 1.

(b) Are any variables in the program uninitialized? (that is, potentially used before they are defined?). Justify your answer in terms of the results of the live variable dataflow analysis from part (a).

3. (dominators) (based on Cooper/ Torczon ex 6. p. 536-7) **(a)** Compute the dominator tree for the control flow graph at right., then **(b)** compute the dominance frontier of each node.



4. (ssa) Translate the CFG from problem # 3 into SSA form. You only need to show the final code after Φ-functions have been added and variables have been renamed. (If you want to edit your answer on a computer, the course assignment calendar page contains a link to the original ppt slide with the diagram. However, don't feel obligated to do this – it might be a time sink compared to just drawing the result.)

Your answer should include all of the Φ-functions required by the Dominance Frontier Criteria (or alternatively the path convergence criteria, which places the same set of Φ-functions), but no additional ones. In other words, you need to include all of the Φ-functions to satisfy the criteria but should not have extra ones that are not required. It should include all Φ-functions that satisfy the Dominance Frontier Criteria even if some of those are assignments to variables that are never used (i.e., dead assignments). Answers that have a couple of extraneous Φ-functions will receive almost full credit, but answers that, for example, use a maximal-SSA strategy of placing Φ-functions for all variables at the beginning of every block will not be looked on with favor.