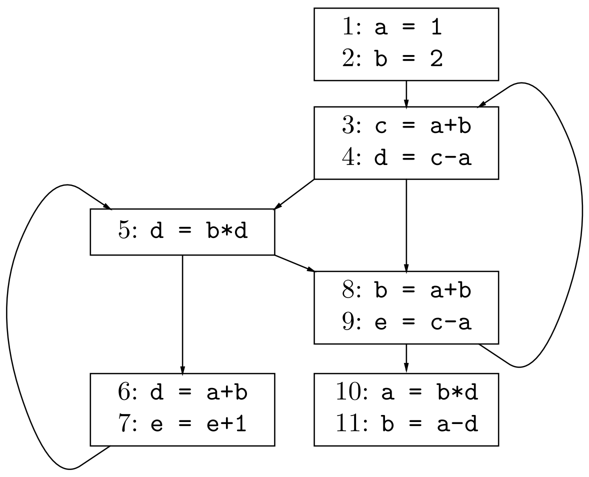
Assignment 5: Optimization

15-411: Compiler Design
David McWherter (cache@cs) and Noam Zeilberger (noam@cs)

Due: Thursday, Nov 15, 2007 (1:30 pm)

Problem 1 — (Very) busy work (30 points)

In this problem, we refer to the following flow graph:



Questions (adapted from Aho, Sethi, and Ullman):

(a) Is any constant folding possible? If so, do it.

(b) Are there any common subexpressions? If so, eliminate them.

(c) An expression e is said to be *very busy* at location ℓ if no matter what path is taken from ℓ, the expression e will be evaluated before any of its operands are redefined. Give dataflow equations for computing all very busy expressions. Is the propagation forward or backward?

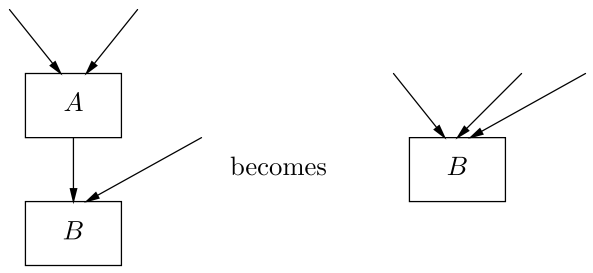
(d) If expression e is very busy at location ℓ we can *hoist* e by computing it at ℓ and preserving its value for subsequent use. (Note: This optimization does not usually save time, but may save space.) Give an algorithm to hoist very busy expressions.

(e) Are there any expressions that may be hoised in the flow graph? If so, hoist them?

(f) Where possible, propagate out any copy steps introduced in the optimizations of (a), (b), and (d).

Problem 2 — A useless problem (30 points)

Many different optimizations modify a program's control flow graph, with the result that often the optimized code will contain "useless" control flow. This presents an opportunity for simplification. For example, suppose that the block A is empty except for a single, unconditional jump to B. Then we can replace any branch to A in our program by a branch directly to B, and eliminate the useless node A:



(a) Give an L3 example for which optimization could produce the above form of useless control flow.

(b) Another form of useless control flow: a conditional branch out of block A, but both targets of the branch are to the same block B. Explain how this situation could arise, and how we should transform the code.

(c) Another form of useless control flow: two basic blocks A and B, where A ends with an unconditional jump to B, but A is B's only predecessor. Explain how this situation could arise, and how we should transform the code.

(d) Another form of useless control flow: an unconditional jump to block A, which is empty except for a *conditional* branch to B or C. Explain how this situation could arise, and how we should transform the code.

(e) The above transformations can be iterated. How would you order the different transformations in order to reach a fixpoint more quickly? In what order would you traverse the control flow graph?

(f) How do these transformations affect dataflow analyses (reaching definitions, available expressions, etc.), in terms of both accuracy and efficiency?